Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l'autorisation écrite du Directeur général du CERN, titulaire du droit d'auteur. Dans les cas appropriés, et s'il s'agit d'utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s'opposer à toute revendication qu'un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

ISSN 0304-2898

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use.

If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.

ABSTRACT

These Proceedings contain written versions of most of the lectures delivered at the 1988 CERN School of Computing. Five lecture series concerned different aspects of parallel and vector processing: advanced computer architectures; parallel architectures for neurocomputers; Occam and transputers; vectorization of Monte Carlo code; and vectorization of high-energy physics code. Software engineering was the topic of three series of lectures: formal methods for program design; introduction to software engineering; and tutorial lectures on structured analysis and structured design. Lectures on data-acquisition and recording were followed by lectures on new techniques for data analysis in high-energy physics. Computer-assisted design of electronic systems, and silicon compilation and design synthesis for digital systems, were the topic of two other, closely related, lecture series. Lectures on accelerator controls and on robotics are also recorded in these Proceedings. Various other aspects of computing were covered in lectures on high-speed networks, document preparation systems, interpersonal communication using computers, and on Fortran 8x. Two general lectures gave an introduction to high-energy physics at CERN.
PREFACE

In August 1988, the 11th CERN School of Computing was held in the Queen's College in Oxford. Sixty-two students from 13 countries stayed for two weeks in the College, and followed 47 lectures and participated in the tutorial sessions.

The School was organized thanks to the efforts and enthusiasm of Professor Erwin Gabathuler and Dr. Paul Jeffreys, with the active support of Dr. G. Kalmus. We owe them our gratitude for making this successful School possible. We gratefully acknowledge financial contributions from the Particle Physics Committee of the Science and Engineering Research Council and from the Rutherford Appleton Laboratory.

Paul Jeffreys did an excellent job in taking care of the local organization, skilfully assisted by Mrs Marjorie Sherwen. They arranged for the School to be held in the beautiful buildings of the Queen's College, where the Bursar, Dr. M.S. Gautrey, excelled in hospitality and helpfulness. We thank all three very warmly for their efforts.

The lecture programme was well balanced and highly appreciated by the participants. Two lecturers from Oxford University should be thanked in particular: Professor C.A.R. Hoare for his special opening lecture, and Professor Jeffrey Marshall for his very amusing and interesting speech on Oxford University life.

Stephen Fisher took care of the tutorial lectures and the exercises in the use of tools for structured analysis and structured design, actively helped by Philip Ovary, Peter Chiu and Douglas Gingrich. David Kelsey and Jason Leake helped in the installation of the equipment. The efforts of all of them are highly appreciated.

Visits to the Joint European Torus at Culham and to the Rutherford Appleton Laboratory were organized. We thank Dr. H. van der Beken and his assistants for the former, and Marjorie Sherwen and the RAL Visitors Centre for the latter.

Digital Equipment Corporation, UK, graciously provided equipment for the practical sessions; Instrumatic UK, Ltd., provided the software for the exercises in SASD; and IBM UK, Ltd., invited the participants to a party in the Nun's Garden of the College. These three firms are warmly thanked for their generous support.

Dr. Gautrey and his staff of the Queen's College left no doubt in the minds of the participants as to the high standard of their kitchen and services. They made our stay a memorable experience!

A visit to London followed by a Promenade Concert in the Royal Albert Hall will remain an agreeable souvenir.

Finally, it is nice to be able to rely each time on the competent help of CERN's editorial section and of Mrs Ingrid Barnett.

C. Verkerk
Editor
ADVISORY COMMITTEE

J.V. ALLABY, CERN, Geneva, Switzerland
B. CARPENTER, CERN, Geneva, Switzerland
R.F. CHURCHHOUSE, University College, Cardiff, Wales (Chairman)
E. GABATHULER, Liverpool University, U.K.
P.W. JEFFREYS, Rutherford Appleton Laboratory, Chilton, U.K.
R.P. MOUNT, Caltec, Pasadena, U.S.A. (presently at CERN)
J.J. THRESHER, CERN, Geneva, Switzerland
A. VAN DAM, Brown University, Providence, U.S.A.
C. VERKERK, CERN, Geneva, Switzerland (Scientific Secretary)
P. ZANELLA, CERN, Geneva, Switzerland
I. BARNETT, CERN, Geneva, Switzerland (Administrative Secretary)
# CONTENTS

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td>v</td>
</tr>
<tr>
<td>FORMAL METHODS IN COMPUTER SYSTEM DESIGN</td>
<td>1</td>
</tr>
<tr>
<td><em>C.A.R. Hoare</em></td>
<td></td>
</tr>
<tr>
<td>SOFTWARE ENGINEERING</td>
<td>8</td>
</tr>
<tr>
<td><em>G. Kellner</em></td>
<td></td>
</tr>
<tr>
<td>THE PRACTICE OF SA-SD</td>
<td>34</td>
</tr>
<tr>
<td><em>S.M. Fisher</em></td>
<td></td>
</tr>
<tr>
<td>DOCUMENT PREPARATION SYSTEMS</td>
<td>56</td>
</tr>
<tr>
<td><em>B. Levrat</em></td>
<td></td>
</tr>
<tr>
<td>INTERPERSONAL COMMUNICATIONS USING COMPUTERS</td>
<td>63</td>
</tr>
<tr>
<td><em>A.J. Casaca</em></td>
<td></td>
</tr>
<tr>
<td>ADVANCED COMPUTER ARCHITECTURE</td>
<td>96</td>
</tr>
<tr>
<td><em>Ph.C. Treleaven</em></td>
<td></td>
</tr>
<tr>
<td>PARALLEL ARCHITECTURES FOR NEUROCOMPUTERS</td>
<td>104</td>
</tr>
<tr>
<td><em>Ph.C. Treleaven</em></td>
<td></td>
</tr>
<tr>
<td>TRANSPUTERS AND OCCAM</td>
<td>127</td>
</tr>
<tr>
<td><em>A.J.G. Hey</em></td>
<td></td>
</tr>
<tr>
<td>HIGH-SPEED NETWORKS</td>
<td>162</td>
</tr>
<tr>
<td><em>A. Danthine</em></td>
<td></td>
</tr>
<tr>
<td>SOFTWARE TOOLS AND METHODOLOGIES FOR THE DESIGN OF DIGITAL ELECTRONIC SYSTEMS</td>
<td>184</td>
</tr>
<tr>
<td><em>M.F. Letheren</em></td>
<td></td>
</tr>
<tr>
<td>SILICON COMPILATION AND DESIGN SYNTHESIS FOR DIGITAL SYSTEMS</td>
<td>234</td>
</tr>
<tr>
<td><em>J. Rabaey</em></td>
<td></td>
</tr>
<tr>
<td>CERN AND ITS HIGH-ENERGY PHYSICS PROGRAMME</td>
<td>260</td>
</tr>
<tr>
<td><em>J.J. Thresher</em></td>
<td></td>
</tr>
</tbody>
</table>
This note expounds a philosophy of engineering design which is stimulated, guided and checked by mathematical calculations and proofs. Its application to software engineering promises the same benefits as have been derived from the use of mathematics in all other branches of modern science.

1. Specification

A specification of an engineering product or component can in principle be expressed as a predicate describing the desired properties of the observable behaviour of the product when put into service. Observable values are represented by free variables occurring in formulae, equations, inequations, or other predicates of the specification. The predicates may also include any definable concepts of the relevant branches of pure or applied mathematics, for example derivatives or integrals. A combination of requirements is expressed as a logical conjunction (and) of the individual predicates describing each requirement; for example, the general differential equations for a dynamic system are conjoined with the boundary conditions describing the particular product to be designed. Conjunction is the connective that permits a large and complex specification to be structured from its smaller and simpler components.

If correct operation of the product will depend upon proper conditions of use, then these too can be expressed as a predicate, and correct behaviour is required only when this precondition is true. If the environment or user fails to meet the specified precondition, the specification is vacuously satisfied, without placing any constraint on the behaviour of the product. Thus the overall specification is expressible as an implication, in which the preconditions are listed as the antecedent and the desired behaviour as the consequent. The explicit formalisation of preconditions for each component of a design, together with a proof of their sufficiency, can be surprisingly laborious; but the extra labour is directed at avoiding errors at the interfaces between the real world and the major components of a system, where mistakes are most likely to occur, most difficult to find, and most expensive to correct.

It is most important that the specification of a new product should be based on an accurate assessment of physical reality and of engineering feasibility; and also that it should describe the customers' genuine requirements. But there can never be any formal or mathematical technique that permits a proof of this. A proof can only start with a statement of the theorem to be proved: this statement would have to include an independent formal description of the real world and of what the customer really needs. But if such an independently believable description were available, it is this that should have been used as the original specification; and the question will again arise whether it is
correct. The only way of avoiding this infinite regress is to make the very first description so obvious, and organise it in such a transparent structure, that it appeals immediately to the good judgement of the experienced engineer and his more sophisticated customers. Any proofs at this stage should be devoted to clarification of the details and consequences of the specification, and to reinforce confidence in its adequacy. Clarity of specification is our only defence against the embarrassment felt on completion of a large engineering project, when it is discovered that the wrong problem has been solved.

There are, however, two important questions that can profitably be asked of a formal specification at the earliest possible stage. Firstly, is the specification feasible? In other words, considering all possible values of the controlled (input) variables, is it certain that there exist consistent values of the uncontrolled (output) variables such that all parts of the specification are simultaneously satisfied? This fact can fortunately be established by mathematical proof; and it is a good idea to do so before proceeding further with the design, because any project based on an inconsistent specification is doomed to failure.

The second question: is the specification complete? In other words, for each permitted combination of the input variables does there exist only one consistent combination of values for the output variables, which are therefore determined uniquely by the specification? For a complete specification, there is no point in adding new clauses describing new requirements. Each new clause is either consistent with the specification, in which case it can and should be proved from it; or it is not consistent, in which case it should certainly not be adopted. Proof of completeness of a specification acts as a check that the specification phase of the project is over. It also permits subsequent steps in the design to be performed by a kind of calculus, similar to that provided by mathematicians and used by engineers in the solution of differential equations and other well-defined mathematical problems.

But there is no need to insist on completeness of a specification for its own sake. An incomplete specification gives the designer valuable options to postpone decisions until there is a better understanding of the consequences of those decisions on the cost and the effectiveness of the product. It also assists in the design of a range of compatible products, which share a common design philosophy but differ in implementation detail.

2. Design

When the specification has been proved consistent and (to an appropriate degree) complete, an engineering project moves into the design phase. The outcome of the design is also expressed in some formalised notation, often including scale drawings. The design notation is usually quite different and much more restricted and cumbersome than that of the specification. This is because it does not directly describe the behaviour of the product, but rather some technologically sound method for its manufacture. Nevertheless, if the technology is well understood, it is possible to reinterpret the design as an indirect description of the range of behaviour of any product made in accordance with the design. This predicate is the strongest specification satisfied by the design, in the sense that it is the conjunction of all specifications satisfied by it, and so implies them all. The correctness of the design can then be proved before manufacture by showing that the design predicate logically implies the original specification; from the meaning of logical implication, it will follow that every observation of every product manufactured in
accordance with the design will also accord with the specification. That is exactly what we mean by correctness of design.

The approach expounded in the previous paragraph presupposes that the completed design is available before the proof starts. For a non-trivial project, this is much too late. Far greater value can be obtained from calculations and proofs conducted throughout all stages in the progress of the design. At each stage, the designer plans to build a component of the product out of several smaller sub-components. The assembly method is decided, and each component is carefully specified. Before proceeding with the design of the sub-components, a proof should be given that the eventual assembly of sub-components (meeting their individual specifications) will meet the original specification of the complete component. This too is done by interpreting each component of the incomplete design as a predicate; and these are then combined by predicate transformers reflecting the assembly technique which will eventually be used to combine the implementations of the components. Such proofs can be repeated at every stage, in the hope of eliminating one of the most serious problems in a large project, namely the diagnosis and elimination of errors detected after assembly of the manufactured components. This hierarchical design philosophy is encapsulated in the slogan “design right - first time”.

3. Abstraction

The relationship between an assembly and its components provides an excellent method of splitting a large design task into subtasks which can be delegated to teams of designers. The subtasks can be carried out concurrently or sequentially, in accordance with project time-scales and staffing. A further method of partitioning each design task or subtask is by levels of abstraction. The highest level of abstraction is the initial specification, expressed in unrestricted notations; the lowest level is the eventual design — machine code, layout masks or manufacturing instructions. Each intermediate phase of design takes as input the entire design document produced at the preceding phase, and elaborates it by incorporation of design decisions particular to the given phase. The concepts, notations and the methods of calculation and proof may differ radically from one level of abstraction to another. It is the goal of the mathematical engineer to ensure that the whole spectrum of phases and methods are mutually consistent and faithful to engineering reality, so that accurate observance of design procedures at each level will lead to a working product. Ideally, the design procedures should be based on calculation rather than proof.

In digital hardware design, there is a well-established hierarchy of levels of abstraction, often including register transfer machines, microcode, combinational logic, wiring and layout, gate design, and electronic circuit design. These tasks are carried out by designers with specialised expertise in each area, communicating with each other by a well-understood terminology. One of the problems of software design is the absence of such a hierarchy; though techniques of data refinement (e.g. VDM) are beginning to meet this need.

Getting a hardware design correct on time is of great commercial importance. The account that follows is based on figures from a real domestic product, but details and names have been changed. In 1983, Acme Ltd. announced that from 1st September it would start selling widgets for £200 each. It soon got sufficient orders to justify a
prediction that 5000 widgets could be sold each week. Acme was thus hoping to take in £1 million per week, of which 15% i.e. £150,000 would be profit. Unfortunately, the original widget design could not be got to work. Three unplanned design iterations were needed; each of these lasted six weeks, so it was not until the middle of January 1984 that widgets went on sale. Since eighteen weeks of widget sales were lost, the total loss of profit was £2,700,000. Similar losses can result from delay in delivery of software, but they are usually borne by the customer rather than the supplier.

4. The Life Cycle

The specification and design of a single product to be implemented and remain unchanged through its working life is difficult enough. Far more difficult and important is the specification of the architecture of a range of compatible products, capable of adaptation and evolution over a period of many decades. This is the problem that faces the major manufactures of aeroplanes, automobiles and computers. It also faces the designer of every large application or systems program, which must be structured from the beginning as a member of the family of programs which it is most likely to evolve into.

Fortunately, the structure of a family of products can be conveniently explored and clearly formalised as a family of predicates. The most general and persistent features of the architecture are expressed in highly abstract terms by a general predicate; the more specific details of specialised subranges and individual products are expressed as separate predicates which can be proved to conform in an appropriate degree to the more general ones. Clarification of the structure of the design space is a serious intellectual challenge; but it provides an opportunity to plan for the multiple use, throughout the working life of the architecture, of the early design steps, the partial implementations, and their interfaces, as well as the completed components.

There is usually a price to be paid for splitting a design into modules with clear general specifications and reasonably simple narrow interfaces. The price is often paid in the form of an increase in the number of components or lines of code, and a reduction in execution efficiency. If the sub-assemblies are not intended for disassembly during use, the price may be reduced by subjecting the design at some suitable stage to a series of correctness-preserving optimisations, which disregard and over-ride the initial modular structure of the design. Such optimisations may be applied automatically, as in many compilers for a high-level level language, or under human guidance. In either case, the validity of the optimisations must be guaranteed by algebraic equations or inequations, which are proved sound for the logical theory in which the design is expressed.

5. Computer Programs

An example of our design philosophy is provided by modern techniques for the design of algorithms and programs for a general-purpose computer. A conventional sequential program is specified by a predicate, whose free variables denote initial and final values of the variables manipulated by the program. The precondition on the environment is permitted to mention only the variables denoting initial values.

The end product of program design is expressed in a very formal notation, usually a programming language. The products described by the program are rather intangible;
they are the executions of the program. These executions correspond so closely to the structure and content of the program that their elaboration is entrusted to a mindless computer, and require no further human intervention. In addition to this mechanical interpretation, there are now available mathematical methods for deriving from the text of a program (expressed in certain restricted languages) the strongest specification which will be met by every execution of the program. The program can be proved correct before execution by showing that this predicate implies the original specification. The correct program can then be optimised if necessary by algebraic transformations which are known to be valid for the programming language in which it is expressed.

In practice, these proofs should be conducted piece-meal during the design of the program. Suppose at some stage it is decided to implement a component with specification $R$ by the sequential composition $(X; Y)$, where $X$ and $Y$ are unknown, because they have not yet been designed. Before the design begins, they are carefully specified by means of an intermediate predicate $S$, which is intended to be true on termination of $X$ and on initiation of $Y$. The specification of $X$ has $S$ as its post-condition and the same pre-condition as $R$; the specification of $Y$ has $S$ as its precondition and the same post-condition as $R$. Now it is obvious that if $X$ and $Y$ meet their respective specifications, then their assembly $(X; Y)$ will meet the overall specification $R$. That is a general, trivial, but useful theorem of the theory of programming. The predicate $S$ (together with appropriate resource allocations for space and time) serves as a provably complete specification of the interface between $X$ and $Y$; if all goes well, no further communication will be needed between the implementors of these two components.

6. Discussion

In our philosophy, design starts with a specification of requirements expressed with the aid of the full power of mathematics and logic. This power is needed to make it obvious that the true requirements have been captured, and not something else. The same power of abstraction is used throughout the design to specify components, formalise preconditions, and standardise interfaces. It is only the final outcome of design (the program or the mask layout files) that needs to be expressed in a restricted notation, so that it may be directly and automatically implemented. And even the final design document is given a mathematical interpretation, so that its correctness can be established by proof.

There is an alternative philosophy of formal design that insists that all or most of the specifications and design documents should be expressed in a notation that is directly executable by computer. This has the advantage that specifications and intermediate design documents can be tested by running them on small examples — though such testing should never take the place of proof. The disadvantages are:

1. The natural way to structure a complex specification is by conjunction, disjunction, quantification, and even negation of predicates describing individual requirements. Only logic provides these vital connectives, and they cannot be implemented on a computer without serious loss of generality or efficiency.

2. The proof that a program meets a specification also expressed as a program can be no easier in general than a proof of equivalence of two programs. Such proofs tend to be intricate: think of proving equivalence of bubblesort with quicksort, without knowing
that they are both implementations of a more abstract specification, expressed in a much less computable notation.

(3) To require a specification or design of a program to be executable is hardly less absurd than requiring the specification of a building to habitable or the blue-prints of a car to be driveable.

7. Conclusion

The novelty of software and software engineering has led to many strange beliefs and unfounded hopes. I therefore conclude with a warning which applies to all branches of engineering: there is no fool-proof methodology or magic formula that will ensure a good, efficient or even feasible design. For that, the designer needs experience, insight, flair, judgement, invention, and even good luck. Formal methods can only stimulate, guide, and discipline our human inspiration, clarify design alternatives, assist in exploring their consequences, formalise and communicate design decisions and help to ensure that they are correctly carried out.

8. Acknowledgement

I am grateful to Mike Gordon for the story of Acme.

9. References


SOFTWARE ENGINEERING

G. Keilner

CERN, Geneva, Switzerland

ABSTRACT

Many studies by academic institutions, government organisations, and industry, have attempted to define improved methods to control the software development process over the past 20 years. Following a brief historic overview, the concepts of software life-cycle, modelling of functional descriptions, modelling of data, diagramming techniques, amongst others, will be described in some detail. Software Development Methodologies - which combine various methods, management procedures, and tools - have been developed in recent years and are now widely used in industry. A short review of the major methodologies will outline highlights and problems. Research is actively pursued to improve the current methods and several of the major research areas will be reviewed.

1. INTRODUCTION

In this paper a brief overview will be given of various basic concepts applied in modern techniques of software development. There exists no unique methodology which could be applied equally well to development of software for engineering, scientific, or business applications. Existing methods are being adapted and new concepts are being introduced to improve the process of software creation and maintenance. Many of the key concepts appear in most of the existing methodologies, however.

Examples shown will mainly refer to one methodology, Structured Analysis / Structured Design (or SA-SD), which is fairly widely used in the engineering environment [1]. However, no attempt will be made to provide a coherent overview of SA-SD. Examples can be found in contributions by J. Harvey [2] and S.M. Fisher [3].

2. WHAT IS SOFTWARE ENGINEERING?

The role of basic sciences - mathematics, physics, chemistry, geology, and others - is to uncover basic principles. Existing knowledge is gathered and interpreted in models to provide new predictions. Experiments are performed to prove or disprove these hypotheses, eventually leading to refinements in existing models or to entirely new insights.

Engineering disciplines are founded on one or several of the basic sciences. Well founded principles are applied to design and implement specific applications (e.g. bridges, space
probes, superconducting magnets, drugs, home appliances). The intention is certainly not to uncover new principles or disprove a prediction. Rather one is interested to create products within prescribed standards of quality, time and cost.

Software Engineering is based on general principles of mathematics and computer science. It is a comparatively young discipline compared to Mechanical or Civil Engineering. Computers started to get widely used in the early 1960's. Many of the most important notions of modern software development concepts were being developed in this period, namely top-down design, stepwise refinement, modularity, structured programming, new programming languages, data modelling. Few, if any, of these concepts were applied in isolation for specific problems during this period.

It was realised very soon that costs for software production were escalating rapidly and that software was notoriously late and unreliable. It became evident that a major cause of difficulty was the lack of a systematic approach to software design and development.

Major new ideas on Structured Design [4] and on Structured Analysis [5] were introduced in 1973 and 1977, respectively. A significant new impetus came from the introduction of the Software Life-cycle concepts in 1975. These concepts are well known from manufacturing in which products are conceived, specified in detail, designed, built, and then maintained until they are no longer supported. A very comprehensive collection of original papers, reviews and comparative studies can be found in [6]. The application of the concepts of the product life-cycle to software development made it possible to develop frameworks which bring together many of the known techniques with appropriate management practicies in Software Development Methodologies.

The goals of Software Engineering are: to introduce an "engineering-like" discipline to improve the process of software production, to improve the quality of software products, to produce software which satisfies precise needs and to manage the complexity of large software systems. Improving software productivity is imperative to satisfy the ever increasing demand for software and to manage the increasing cost of software development as has been demonstrated by B.W. Boehm [7].

![Fig. 1 Software cost trend](image1)

![Fig. 2 Growth in software demand](image2)
Figure 1 shows recent and predicted software cost trends in the USA and worldwide, while Fig.2 shows the growth in software demand for several US spaceflight programs as an example.

A very extensive discussion on Software Engineering and various approaches to solve the problems can be found in [8].

3. SOFTWARE LIFE-CYCLE

The period between the initial decision to implement a software system and its final end of utilisation is called the software life-cycle. The concept allows to structure the process of software creation into several distinct phases. The number of phases and their names may vary with different models but boundaries between different phases are clearly defined. The major deliverable products (documents, code, test results, reports, etc.) are identified and provide an essential means to measure actually achieved goals against predictions. A typical life-cycle model may include the following phases:

User Requirements or Feasibility Study phase. Some people consider this phase outside of the life-cycle. However, it is a necessary preliminary step in order to define the needs of the users of the software system. It is essential to get a good understanding of the problem to be solved. Generally this is a rather informal document.

Software Requirements or Analysis phase. The aim is to produce a functional specification of the requirements of the system. It includes the user requirements, external requirements, and constraints (e.g. external interfaces, performance, environment, safety). Requirements analysis provides the software designer with a representation of information and function that can be translated into data, architectural, and procedural design. This creates a logical model to show what the system should do and defines the information domain that will be treated by software. As the size of the problem grows, the complexity of the analysis task also grows. Therefore the system must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion. In many instances it may be difficult to specify requirements in detail, especially if a system is to be used by a wide spectrum of users with different (often conflicting) requirements. In such cases a model of the software (or a subsystem), called a prototype, is constructed to assess alternatives. This should finally result in formalized requirements.

Design phase. The aim is to translate the functional specification of the system derived during the Analysis phase into a solution in terms of software modules, i.e. how the system should be implemented. From the project management point of view software design is conducted in 2 steps. Overall design (also called Architectural design) is concerned with the transformation of requirements into data and software architecture. This creates a hierarchy of components with their interfaces. Detailed design focuses on refinements to the architectural design that lead to detailed descriptions of data and data structures and algorithmic representations in software modules. The quality of the decomposition is evaluated against a set
of criteria (e.g. coupling and cohesion of modules). Optimisation and repackaging are used to satisfy certain constraints (e.g. timing constraints, external packages or libraries, interface routines).

**Implementation and Testing** phase. The aim is to translate representations of software derived in the previous steps into a form that can be "understood" by the computer. This includes coding of modules, using principles of structured programming and coding standards. Testing includes testing of individual modules, testing of groups of modules in incremental and phased approaches, and acceptance testing of overall system.

**Operation and Evolution** phase. Often this is also called the Maintenance phase. Its main purpose is to cater for operation of the software system in the production environment. Potentially this can be a very long period and past experience has shown that over 60% of all software cost may be spent during this phase. The basic reason is that computer programs are always changing. There are bugs to fix, enhancements to add and optimizations to make. The environment or the application may also change requiring upgrades to software. These upgrades may require changes to code and documentation, changes to design issues, or even redefinition of requirements for the system. Various phases of the software life-cycle will be initiated to re-engineer the subset which requires updates.

4. **MODELS FOR SOFTWARE DEVELOPMENT PROCESS**

The primary purpose of models for the software development process is to define the order in which the various phases of the software life-cycle should be involved in the software development and maintenance. They allow to establish transition criteria for progressing from one stage to the next. These include criteria for alternative options and entrance criteria for the next stage [9].

A very simple model is being used since the earliest days of software development: write some code, fix the problems in the code. It is quite evident that there are many difficulties associated with this very simple approach. After a number of fixes the code becomes so poorly structured that even the author may have problems to know what is happening. Since the code is probably not too well structured to start with it gets increasingly expensive to maintain it and test modifications. Even before the introduction of the software life-cycle several attempts had been made to define an improved way to develop large software systems. The Software Requirements and Design phases introduced by the software life-cycle provided the necessary means for a better organisation of the development process.

Another software process model which is being widely used is the "waterfall model", shown in Fig. 3. It includes all phases known from the software life cycle. Each stage includes a step for verification ('ensure that the software correctly implements a specific function') or validation ('ensure that the software built corresponds to the user requirements'). The feedback loops between stages ensure that costs for rework are minimised. It had been shown that the relative cost to fix errors introduced in early phases of a project increase dramatically the later they are discovered in the project [9]. This is shown in Fig. 4.
Fig. 3 The waterfall model of the software life cycle

Fig. 4 Increase in cost to fix or change software throughout life cycle
The waterfall model has encountered a number of problems and criticisms. The main objection is that the model is too 'strict' - it requires that a stage is completely finished (with full documents and formal review) before the next stage can be started. In many larger projects it has proven quite difficult to accomplish this. Various extensions have been proposed to accommodate intermediate steps to include prototyping, evaluation or selection of alternatives, incremental and parallel development of subsets.

The "Spiral Model" of the software development process has evolved over the past few years, based on experience with various refinements of the waterfall and other models [10]. The model can equally well be applied to development and maintenance and it can accommodate most other models as special cases. Fig. 5 shows a rendering of the model as discussed in detail by B.W. Boehm in [10].

The main underlying concept is that there are several stages where objectives, alternatives and constraints are determined, associated risks are evaluated via prototyping, simulation, benchmarking, and other techniques, and resolved. Depending on the remaining risk factors a new cycle of the spiral may be started to evaluate and resolve more of these risks. Eventually the basic waterfall model with requirements analysis, overall and detailed design, etc. will be followed. Depending on the complexity of the project and the risks involved only a subset of all the potential steps may actually be implemented.

Fig. 5 Spiral model of the software process
5. **DIAGRAMMING TECHNIQUES**

"A picture is better than a thousand words". Though coined in a very different context this statement is equally well applicable to software development. Complex activities and procedures are easier to describe in graphical form than in standard prose (however well it may be organised). Diagrams represent an essential communications tool which enables developers to interchange ideas and uncover poor or wrong structuring early in the development process. At any stage diagrams can be discussed between different people (members of the team, managers, customers, hardware people) since they provide an up-to-date view of the current state of understanding. They represent an essential documentation aid in the maintenance of programs and for upgrades. Diagrams are well suited for interactive manipulation on computer graphics screens. This speeds up the process of creation and modification, it enforces standards through validity checks, cross references and calculations performed by the computer. It alleviates a lot the rather boring job of manual verification.

Many different diagramming techniques have been developed to describe process structures, control structures, and data structures. A detailed discussion of various diagrams has been provided by J. Martin and C. McClure [11]. Figure 6 shows examples of different diagramming techniques for description of data and activities.

![Data and Activities Diagram](image)

**Fig. 6 Areas in which different diagramming techniques are applicable**
Figure 7 shows a summary of the capabilities of diagramming techniques to describe certain functions.

<table>
<thead>
<tr>
<th>WHAT CAN BE DRAWN WITH THE TECHNIQUE?</th>
<th>DECOMPOSITION DIAGRAMS</th>
<th>DEPENDENCY DIAGRAMS</th>
<th>ENTITY-RELATIONSHIP DIAGRAMS</th>
<th>DATA NavigAtION DIAGRAMS</th>
<th>HPO CHARTS</th>
<th>STRUCTURE CHARTS</th>
<th>WARNER-JONES CHARTS</th>
<th>FLOWCHARTS</th>
<th>STRUCTURED ENGLISH</th>
<th>NASSI-SHENKER DIAGRAMS</th>
<th>ACTION DIAGRAMS</th>
<th>DATA BASE ACTION DIAGRAMS</th>
<th>DECISION TREES AND TABLES</th>
<th>STATE-TRANSITION DIAGRAMS</th>
<th>HOS CHARTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise model showing corporate functions</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional decomposition species I (tree structure only)</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional decomposition species II (tree structure plus input and output)</td>
<td></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional decomposition species III (axiomatic control of decomposition)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction between business events</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow of data</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonprocedural (compound) data-base actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Conditions</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Case structure</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Repetition</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Loop control</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Good for showing complex logic</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Designed for showing highly complex decisions</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Linkage to data model</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Linkage to fourth-generation languages</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Tree structured data</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Plex structured data</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Derived data items</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Corporate data models</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Data-base navigation</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Fig. 7 Summary of the capabilities of diagramming techniques

Although many more diagramming techniques are currently used we will show examples of a small subset only. We will restrict to engineering environments only.

5.1 Data Flow Diagrams

Data Flow Diagrams are used during the Analysis phase of SA-SD [12]. The DFDs provide a graphical representation of a model of the system to be produced. They show the flow of information through the system, the processes which transform input data flows to output data flows, the data stores as a repository of data, and the external sources and sinks of data. The DFD allow the functional decomposition of the complete system and all the interfaces between the processes to be shown in the form of a network.

Figure 8 shows an examples from our applications. It shows the overall view of the
ALEPH software: data acquisition, event reconstruction, physics analysis (including interactive graphics) and event simulation. The following features are noted at a glance: a) all programs make use of a common data base to extract constants and store bookkeeping information, and b) programs are connected since output from one program can be used as (partial) input to other programs. This provides consistency between different parts of the software, and eases the problems of maintenance of a large number of parameters. Common data structures are defined throughout the system to simplify communication and transfer of data between processes.

Fig. 8 Data Flow Diagram showing principle components of ALEPH software

5.2 Extensions for real-time Analysis

One of the basic concepts of Structured Analysis required that only data flows and processes are shown on DFDs and that all control information was eliminated at this stage. This looks quite reasonable for programs which are basically batch oriented, e.g. Monte Carlo Simulation or Event Reconstruction. However, it causes problems for the design of real-time systems (e.g. data acquisition) or highly interactive systems (e.g. interactive graphics). Since 1982 new concepts were gradually introduced who meet the same goals as the basic SA, but
put equal emphasis on process and control [13].

In a data acquisition environment the state of the whole system can change due to the detection of a particular event in the system's environment and this can be made to affect the way the system subsequently behaves. It is extremely important for the technical methods to handle these features correctly as they have a major influence on the behaviour of the system. This situation typically occurs in interactive applications where the operator can change the state of the system by the press of a button. Figure 9 shows an example of a CFD (Control Flow Diagram) describing that part of a system for providing operator control over datataking activities. Following the notations of Ward and Mellor [13] events are represented by dotted lines to distinguish from the data flows represented by solid lines and the control transformation ("Invoke DAQ Request") is represented by a dotted circle. The control transformation appears as a transaction centre which must keep track of which state the system is in and use this information together with the input control flow to determine which action procedure to activate.

Fig. 9 Control Flow Diagram of an interactive application (Ward-Mellor style)
The control logic for implementing the transaction centre can be extremely complicated and experience has shown that it is in this area that errors are introduced which are most difficult to detect in the code. State Transition Diagrams (STD) provide an ideal technique for specifying this control logic as it provides a method for representing the states of the system, the conditions for transitions between states and the actions to be performed on making the transitions. Figure 10 shows the STD for the example shown in Fig. 9.

![State Transition Diagram](image)

**Fig. 10 State Transition Diagram for specifying transaction centre ('Invoke DAQ Request')**

5.3 SADT diagrams

SADT, Softech's Structured Analysis and Design Technique, has been published in 1977 [5]. Since then the methodology has been applied to a wide variety of major projects in a broad range of application areas [14]. The graphical representation is shown in an example in Fig. 11 [15] which describes a model for software development. The diagram shows more details of process "development" from a higher level diagram. As in the case of DFD's this diagramming technique allows a hierarchical top-down decomposition of the whole system into easy to grasp pieces. In the activity diagram the boxes represent 'actions' or 'processes' which transform input and control to output. The arrows represent interfaces between the boxes for 'inputs' (enter box at left), 'output' (leave box at right), and 'control information' (enter from top).
Outputs from one box can get inputs or control information for other boxes. Arrows entering the box from below represent 'mechanisms', i.e. connections from other boxes anywhere in the system or external to the system. Note, that there are no data stores used in this technique. Another form of the same diagram, the data diagram, uses an orthogonal view - boxes describe the states of things and the arrows represent the activities. Both types of diagrams are required to describe a system completely.

![Diagram](image)

Fig. 11 SADT diagram (at first level of decomposition)

5.4 Entity-Relationship Diagrams

Entity-Relationship Diagrams (ERD) provide a comprehensive way to graphically depict data structures and specific properties by showing the entities and the relations between them. ERD have been included recently in SA. Since all information about data is contained in the data dictionary ERDs can be used to display any conveniently selected subset in graphical form.

Two of the LEP experiments (ALEPH and DELPHI) each have over hundred crates of FASTBUS readout electronics attached to their detectors. Figure 12 shows an extract of the FASTBUS data model describing the physical layout and addressability of FASTBUS modules.
The data used in the online system can be subdivided into several schemata. These include descriptions of the readout components (FASTBUS), the detector control components (Slow Control), the data taking environment (Run and Partition) and the overall Detector description. The example on Fig. 13 shows a partial Entity-Relationship Diagram for the ALEPH detector description with only two of the nine components shown explicitly.
5.5 Structure Charts

Structure Charts (SC) provide a model of an implementation of a system in terms of software modules (e.g. functions and subroutines in Fortran). Data flows and control flows between modules are shown and extra procedural information can be indicated. Objective criteria allow evaluation of the design. Specific questions of repackaging, access procedures and information hiding modules (e.g. for complicated access to database information), compromises of good design for efficiency reasons, real-time and hardware constraints can all be shown explicitly on the SC with a few additional symbols. An example of a (partial) SC is given in Fig. 14 which shows the program structure derived from the CFD of Fig. 9.

Fig. 13 Extract from the Entity-Relationship diagram for the ALEPH detector description

Fig. 14 Structure Chart illustrating the interactive application
6. **FUNCTIONAL DECOMPOSITION**

Complex systems can not be described in a single phrase or a single diagram. Rather, some overall decomposition of the system into smaller pieces has to be provided.

Usually the first step in the decomposition of the system is to delineate the context of its environment, i.e. everything with which the system must interact but which is not part of the system itself. This overall view will be refined in successive steps of decomposition which show more and more details of the functions and the input data which are transformed by these functions into output data, possible under some control. This decomposition is continued until the individual processes represent 'functional primitives', i.e. simple procedures, mathematical algorithms, etc. The result is a hierarchical, top-down decomposition of the whole into easy to grasp pieces which generally perform a single function.

Diagramming techniques are very well suited to structure large systems into manageable portions. They provide a 'blue-print' of the system to be built. Figure 15 shows schematically the decomposition process for data Flow Diagrams. Each process on a higher level may be decomposed further to show more details. The basic reason for this procedure is that diagrams have to be simple to read in order to be useful at all. A total of $5 \pm 2$ different objects on a single diagram seems to be a practical limit.

![Diagram](image)

Fig. 15 Notation for Data Flow Diagram decomposition into several levels of DFD

Figures 16 and 17 show examples of decomposition for SADT diagrams. Figure 16 represents the top level diagram describing the software environment for software produced for ESA [15]. The expansion of the process 'Development' has been shown in Fig. 11.
Fig. 16 SADT diagram for top level of project

Figure 17 represents a further level of decomposition of the first box of Fig. 11.

Fig. 17: SADT diagram for decomposition into next level
Examples for functional decomposition used by other diagramming techniques can be found in [11].

7. **FUNCTIONAL SPECIFICATION**

The lowest levels of a hierarchical decomposition represent 'functional primitives', i.e. single functions, in general. They can be specified concisely and independently of the other parts since all the data interfaces are defined. Various approaches to functional specification exist:

*Natural Language Specifications* use a semantics based on informal language. A widely used method is PDL (Program Design Language), also called pseudocode or Structured English. In general this is a restricted natural language (English mainly) with additional keywords to provide structuring (constructs familiar from structured programming are used). There are many ways to define pseudocode. The more relaxed form is easy to write but, being less precise, is less useful and more difficult to process by automated tools. In principle, such specifications are easier to understand for the non-expert but they are inherently ambiguous. All checking and tracing with requirements has mainly to be done 'by hand'.

*Mathematical Specifications* use a semantics based on a proof system. Proofs are used to discover inconsistencies and to derive consequences of the specification. These are helpful for validation and checking of completeness. A discussion on formal methods for program specification has been presented by C.A.R. Hoare [16].

*Operational Specifications* use a semantics defined in terms of an execution model. They are analyzed, checked for consistency, and validated by static analysis based on the execution model or by execution itself. Descriptions using State Transition Diagrams (see Fig. 10) or Petri Nets provide examples of this.

In practical applications for different methodologies these approaches are combined in various ways. In SA-SD some aspects are represented in an operational specification (e.g. State Transition Diagrams, Process Activation Tables) [17], others use informal language (e.g. process and module specifications in PDL). In VDM [18, 19] important components of a specification are both in English and in mathematical equations. Parnas [20] uses a combination of English, State Transition Diagrams and equations to specify the A7 project. Specifications in SREM [21] and PAISley [22] are primarily operational, but mathematical proofs are used for certain validation aspects.

A very comprehensive comparison of various methods used for Software Requirements Specification can be found in [23].

8. **DATA MODELLING AND COMMON DATA DICTIONARY**

Data-flow oriented analysis methods (SA-SD) provide functional decomposition of a system into lower level diagrams which show more and more details of the functions and their interfaces (see chap. 6). As the functions are decomposed also their associated data flows may get decomposed. The higher level diagrams show rather abstract 'composite' data flows. The
lowest levels show generally data flows which can no longer be decomposed into other dataflows.

It had been realised for quite some time that the gradual decomposition top-down did not quite correspond to a realistic situation. In general, much more was known about data and data structures then about the functional organisation at the start of a project.

Data-structure oriented methods focus on data structure rather than data flows. Examples for these methods are represented by JSD, Jackson System Development, by M.A. Jackson [24] and DSSD, Data Structured Systems Development, by J.D. Warner and K.T. Orr [25]. Each of these methods has a distinct approach and specific diagramming techniques. The common characteristics of such methods is, that they identify key information objects (also called entities or items) and operations (also called actions or processes). Each method assumes that the structure of information is hierarchical. Each provides a set of steps for mapping a hierarchical data structure into a program structure.

Data modelling concepts have been introduced into data-flow oriented methods in recent years. The Entity-Relationship model proposed by P.P Chen [26] is mainly used. General aspects of data modelling have been discussed by S.M. Fisher [3]. Examples of the graphical representation, Entity-Relationship Diagrams, have been shown in Figs. 12 and 13.

Data contents and data structures are generally recorded in a data dictionary. This contains definitions for all data on diagrams, in functional specifications, in the code, or in the data dictionary itself. This data dictionary acts as a unique source of all information on data. A computer readable data dictionary is invaluable to ensure consistency and completeness of data across all phases and products of the software life cycle.

9. MANAGEMENT PROCEDURES

Management procedures play an essential role in the overall software development process. In order to conduct a successful development project one must understand the scope of the work to be done, the resources in manpower required, the tasks to be accomplished, the milestones to be tracked, the cost to be expended, and schedules to be followed. Software project planning provides this understanding. Useful techniques for cost and schedule estimation and tracking of the estimates against actually achieved progress have been developed in the past years. Very detailed discussions of basic principles and procedures for software project planning, implementation planning, quality assurance, testing strategies, and configuration management, can be found in many books on software engineering [8, 9, 12].

Strategies for the use of such procedures are either defined by using a particular software development methodology (chapter 11) or are decided for a given project or a whole organisation. An example of this is provided by ESA [27]. Figure 18 shows the overall life cycle management scheme that has to be followed for every software project developed in-house, by contractors, or software companies.
Figure 19 provides an overview of the software life cycle management procedures to be followed during various phases of the development.

![Software life cycle management scheme](image-url)
10. **AUTOMATED TOOLS**

Nearly all activities associated with software production have been manual in the past. However, many of those activities could, and should, be replaced by computer supported facilities. Currently tremendous investments are being made to develop automated tools for all phases of the life cycle and to support various methodologies. The clear aim is to improve productivity and quality of the software being developed. The key concept is to maintain all information in a central data base which can be accessed by anybody who wants or needs information at any time.

Automated tools are essential in order to help people using the methods effectively. It is possible to draw a few diagrams with paper and pencil. However, it gets very boring and people tire very quickly of rubbing out sections of diagrams or redrawing them from scratch. Tools have to provide graphics editors which allow creating and modifying diagrams quickly and easily and allow for rapid navigation between levels of diagrams and related information.

Equally essential are formal verification for correctness and completeness of the information on diagrams, specifications, and data descriptions and their cross-relations. Manual checking of this information is very tedious and error-prone and it simply consumes too much time which should rather be used to create good designs. Integration of all tools via a common data base is essential to go back and forth between information relating to various phases of the project.

In the past 2-3 years there have been dramatic changes due to the availability of reasonably priced workstation hardware. Development of data bases and sophisticated user interfaces which exploit features of the new hardware allow creation of well integrated sets of software development tools which have now appeared on the market. This may finally provide the real breakthrough for a wider application of software development methodologies.

Rather few CASE tool packages were available commercially for the support of the full SA-SD methodology in 1984. Several more supported only the Analysis phase and extensions for the Design phase were just promised. Data modeling was not available then. This was even worse for other methodologies which had virtually no support tools. The situation has changed dramatically since then. A recent survey of CASE products and vendors lists 135 entries [28] and amongst those some 31 products available for the support of development of real-time systems [29]. These numbers do not include tools traditionally used in the coding phase (compilers, debuggers, source control, static analyzers, etc.)

A word of caution may be useful: the methods and tools allow to make beautiful designs and layouts but there is no way that they can check if all that makes really sense - this is still the domain of human beings, fortunately.

11. **SOFTWARE METHODOLOGIES**

The coherent combination of methods, management procedures, and automated tools is
called Software Development Methodology. Unfortunately there exists no single methodology which could be equally well applied to all projects or in all environments (e.g. software development in a University or Research environment will be quite different from a defense contractors environment). Some of the methodologies are better suited for a particular field of applications (e.g. embedded or real-time systems, business systems, transaction processing, telecommunications, scientific or engineering projects).

Surveys can provide very useful information for the selection of a methodology suitable for a given field of application and environment. A very thorough comparative survey [30] had been used for our own selection. Table 1 shows a condensed summary for the applicability of various methodologies available in 1982. A more recent survey can be found in [28].

Table 1
Methodology applicability

<table>
<thead>
<tr>
<th>METHODOLOGY APPlicability</th>
<th>Application Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methodology</td>
<td>Embedded</td>
<td>Sci/Eng</td>
</tr>
<tr>
<td>ACM/PCM</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>DADES</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>DSSAD</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>DSSD</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>EDM</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>OGIS</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>HOS</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>IBM/MPSD-SEP</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>IESM</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>ISAC</td>
<td>I</td>
<td>W</td>
</tr>
<tr>
<td>JSD</td>
<td>W</td>
<td>S/W</td>
</tr>
<tr>
<td>MERISE</td>
<td>W</td>
<td>S</td>
</tr>
<tr>
<td>NIAI</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>PRADOS</td>
<td>S</td>
<td>W</td>
</tr>
<tr>
<td>REMORA</td>
<td>W</td>
<td>S</td>
</tr>
<tr>
<td>SADT</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>SARA</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>SA-SD</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>SD</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SDM</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>SEPN</td>
<td>W</td>
<td>S</td>
</tr>
<tr>
<td>SREM</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>STRADIS</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>USE</td>
<td>I</td>
<td>W</td>
</tr>
</tbody>
</table>

Key:
- W = well suited
- S = satisfactory
- I = inappropriate
- x = suitable
- (W or S not specified)
- IE = insufficient experience
- ... = no answer
- $ = "Large is 250K lines, 75-100 effort years"
- blank = is not suitable

Sci/Eng = Scientific/Engineering
O/S = Operating Systems
DP/DB = Data processing, database
AI = Artificial intelligence
12. PRACTICAL EXPERIENCE

One can find many papers in the literature which relate (mainly positive) experiences with using the techniques discussed. In this section we would like to sum up our own practical experience in using SA-SD for about three years for several projects. Very similar conclusions have been drawn by other groups in High Energy Physics who have used SA-SD for a variety of projects (ALEPH, SPS and LEP control systems, Fastbus system management, OPAL, D0, H1, Zeus, Cleo).

Positive aspects of using the methodology:
- there is a common style of work adopted and understood by everybody which makes it easier to obtain a much more coherent product despite the many people, with their individual 'tastes', who contribute to it,
- the idea to spend a sizable fraction of the software development in analysis of the problem is really a very positive aspect because it forces people to get clear ideas what they intend to do and document it so that other people can discuss it,
- documentation via various diagrams and descriptions provide powerful means to abstract and show details wherever required,
- the fact that documentation is available even before code is very useful indeed,
- frequent walkthroughs and reviews provide verification and validation all along the development process, in principle one should have a better product at the time of startup of production,
- the partitioning of work for several teams working independently is useful in our current context where software development has to be done distributed geographically,
- the definition of all data in one unique source and in a standard way should bring enormous benefits in the long run since this is one of the more tedious problems,
- the methods are flexible enough to be adapted to specific needs and it is reasonably easy to learn and use them.

Inconveniences or problems encountered:
- one of the most serious problems was non-availability of adequate tools - either because they did not exist yet or were too expensive or required expensive hardware platforms. This situation has changed now and a number of suitable CASE tools are being offered,
- one has to invest money and manpower to teach courses, learn the methods well and make sure that a consistent standard of quality is kept by investing more manpower and time for the review sessions,
- while the methods and procedures used for Analysis are quite powerful and easy to use, especially with the recent extensions for real-time applications, the methods used during the Design seem to be much more tedious and labour-intensive since they can not so easily be supported via automated tools,
- the methods are flexible and can be adapted - but this sometimes creates problems because there are different ways to express things and one may waste time by 'fiddling' around.
13. **OUTLOOK TO THE FUTURE**

Software Engineering techniques have been successfully transferred from the research area into the real world and are now available in the form of comprehensive methodologies which are taught in seminars and are supported by automated tools. Many of the CASE tools have now some of the extensions needed for real-time design.

However, at present only few of these tools can handle concurrency and parallelism well enough to move smoothly from design to implementation. There are still gaps to move from DFD/CFD and STD to the details needed for actual coding. Extension of the tools will allow better automation of code generation and better simulation capabilities and several activities are under way. Amongst these a group including P.Ward, co-author of the real-time extensions now widely available [13], is trying to expand and combine existing notations to provide a more complete design discipline proposing ESML, the Extended System Modeling Language [31]. Very recently methods of real-time structured analysis have been adapted to support a combined hardware/software specification and design for an application specific system on a single chip [32]. Several CASE tools are now available with sophisticated support for programming languages like ADA, C, and C++ in the near future.

Though the driving force behind these rapid developments is the US aerospace and military establishment the methods are now also widely used outside this area for systems programming, telecommunications, on-line transaction processing, fault-tolerant data processing, robotics and others.

The methodologies are well suited for development of new software systems. However, a maybe surprising fact is that about 80% of all programmers are working on maintenance and upgrades of existing software and do not design software starting 'from scratch'. The basic problem is that one wants to preserve the investment in manpower, knowledge and resources that went into the development of the software. Productivity for rewriting is very poor since one generally has to derive actions and intents from existing code with incomplete or no documentation available. There is therefore strong demand for tools for reverse-engineering of existing code. This would allow to restructure and document existing programs and help the programmer to translate source-level code into specification-level descriptions. These can then provide the basis for an enhanced version, where existing CASE tools can be successfully used. Obviously this task is not easy but several tools are starting to appear [33].

14. **CONCLUSIONS**

Software Engineering techniques have been used within ALEPH and other groups in High Energy Physics for approximately three years for a variety of projects. The methods have been found to be extremely helpful in providing a clear understanding of the required functionality of the system and as a means of communicating ideas and suggestions both within the groups and to the Collaboration as a whole. Lack of convenient automated tools in the past has meant that SA-SD has been used more as a collection of useful techniques for producing reliable and
robust software rather than a rigorous methodology for proceeding from a requirements specification to a validated set of program modules. This situation has changed now and we expect that the methods will be used more widely in future.

*Should one use these methods NOW?* YES - even if there are new developments and new tools. One should not forget that the software stone-age ended just some 30 years ago! Many of the basic concepts will not change much. It is best to start with these methods as soon as possible since it has been convincingly shown that the software creation process is much improved. Adopting these new methods implies a major change of personal habits, changes to the methods can always be incorporated later.

*Which methodology to use?* This depends on the characteristics of the projects as mentioned above. SA-SD seems to be applied successfully to a wide variety of projects. It is also well supported by several CASE tools.

*Where should the methods be applied?* Definitely for medium to large projects where many people participate in development, projects of long life-time, and projects with an important evolution phase. Initially it is much better to use them for small projects since one passes much faster through the various phases and gets a better feeling which methods are well suited and which are not in the particular context. One should not forget the 'learning curve' - it takes about 2-3 projects before one is really familiar enough with the methods in order to improve also efficiency and productivity.

*Who should learn the methods?* All people who are involved in analysis and design of software. All people who code and test the software (if not the same people as above), but maybe they need to be less familiar with all details. The managers who are responsible for the overall project (hardware and software) - this is the best way that they can monitor progress of the development process.

*The important point is to start using a consistent way to produce software.*

**ACKNOWLEDGEMENTS**

Introduction of new methods and change of work style need the commitment of many people otherwise this would simply not be possible. I would therefore like to thank all people who went to the pains to learn about the new methods and employ them successfully. However, I would also like to thank those people explicitly who contributed a great deal of enthusiasm and constructive ideas to adapt the methods to our environment and projects, namely P.Palazzi, S.M.Fisher, W.Zhao, J.Knobloch, M.Green, J.Bunn, J.Harvey, R.McClatchey, T.Charity, A.Putzer, the ADAMO team in general and F.Dydak in particular for his foresight.

* * *

**REFERENCES**

C. Gane, T. Sarson, Structured Systems Analysis, Prentice-Hall (1979),


[20] D. Parnas et al., copies of several papers in Naval Research Laboratory Memorandum Reports, Washington, D.C. received from author.


K.T. Orr, Structured Requirements Definition, Ken Orr & Associates Inc., Topeka, KS


Sigsoft Engineering Notes, vol.8, no.1 (Jan.1983) 33, and M. Porcella, P. Freeman,
A.I. Wasserman, ADA Methodology Questionnaire Summary, idem, 51.

Language Based on the Data Flow Diagram, ACM SIGSOFT Software Engineering

449.

THE PRACTICE OF SA-SD

S.M. Fisher

Rutherford Appleton Laboratory, Chilton, Didcot, UK

Abstract

These notes on the practice of SA-SD describe the history of what is now included under the SA-SD umbrella, and show how the methods may be applied for different types of project.

1 Introduction

A model is a representation of a system which behaves like the real system. Structured Analysis and Structured Design (SA-SD) is a collection of modelling techniques, which are useful when constructing a software system. Much of the development of SA-SD has involved linking existing methods. As this development is continuing, there is no agreement upon which modelling techniques are now part of SA-SD, though there is a tendency to appropriate any useful technique.

These lectures are based on my understanding of the methods and the way I have applied them and seen other people applying them. Though I have been much influenced by the books of Ward and Mellor [1] and by talking to my colleagues in ALEPH, this is rather a personal view of the subject.

Practice in SA-SD is essential before the method can be appreciated. It is easy to see that the methods look 'reasonable', but to be able to apply them takes some effort. The methods of SA-SD are basically simple, and the diagrams are, or should be, easy to read. To gain maximum benefit from reading these notes, examine the diagrams carefully and try applying the methods to something of your own as you go through.

Three main examples are used here to demonstrate different points. The first, running a farm, has been borrowed from a paper [2] describing a different methodology. The second is based on the stop-watch function of a digital watch. This describes a real watch, but was invented to demonstrate most of the principles of a system in which control is important. The third example, track finding, was chosen as a contrast because for this control is not important.

1.1 The model in SA-SD

To be useful the model must show the aspects of the system which are important to the person using it. For example, a model train is of little interest to children if it does not have wheels which turn, even though the scale, materials and colours are very accurate.

Different models of a system are appropriate at different stages of development. In particular one must distinguish between the Logical and the Physical (or Implementation) model. As the model of a system is constructed, understanding of that system increases. The steps in building a system should be:

Concept $\rightarrow$ Logical Model $\rightarrow$ Physical Model $\rightarrow$ Physical System

The concept may start off life as a narrative document, or it may be less tangible. It is frequently rather poorly defined.

The logical model shows as simply as possible what the system should do. This is analysis. As far as possible, design decisions should be avoided at this early stage.

Many physical models may correspond to a logical model according to the compromises chosen to satisfy the physical constraints. This is the design stage where the system is described in some detail.
Building a working system from the physical model should then be straightforward as much of the hard work has been done. This is the task of \textit{coding}.

If the physical system is to do what was originally envisaged then it is clear that the various steps must be traceable. Testing has not been identified as a separate phase, rather it is something which should be going on all the time. Checking the various stages by formalised review procedures is essential to ensure that the desired system is coded.

2 Development of the components of SA

This section describes the history of the various methods which have gone into the current SA and indicates the direction in which the methods are going.

2.1 Process modelling: the Data Flow Diagram

Structured Analysis and Design Technique (SADT) was formalised by Ross in 1974 [3]. Fig. 1 shows the various processes involved in running a farm. Each rectangle is a process and data inputs to a process are shown on the left of the box and outputs leave on the right. SADT distinguishes control inputs by showing them entering the box at the top. Note that many of the outputs are themselves fed back as control inputs to other processes. Control and data lines are drawn the same way. It is only the function that a particular input to a process has that distinguishes it. Further the distinction between data and control inputs seems to rely upon intuition.

![Figure 1: SADT style diagram of a FARM](image)

De Marco in 1978 [4] described dataflow diagrams (DFDs) which were similar in function to the ‘datagrams’ of SADT but all unimportant details eliminated. The control data shown on an SADT diagram would then tend to be regarded as unimportant. Referring again to Fig. 1 the process \textit{Run Household} may not be regarded as essential to running a farm, so if we eliminate this along with the various control data such as \textit{Weather} and \textit{Prices} we are left with a much simpler diagram; see the top half of Fig. 2 where nothing is left but the main data flow through the system.
Figure 2: A data flow diagram of a farm in early SA style

The data store was introduced to go between processes which were essentially asynchronous. The external, i.e. sources and sinks of data was also introduced. This leads to a diagram like the one at the lower part of Fig. 2. Notice that the diagram can be interpreted as showing the flow of objects: pound notes, bags of seed, and sacks of carrots, or it can be interpreted as the data describing the flow of these objects: requests to transfer money, inventories, orders, etc. Until the dataflow contents are properly defined this will not be clear.

Since then the trend has been to put detail back in [1]. In particular if the models are not consistent, checking tools spot inconsistencies produced by omitting 'unimportant details' in a higher level diagram. We also see in Section 2.3 that control is no longer disregarded.

2.1.1 Drawing DFDs: Rules

The DFD is a representation of a system which shows a network of processes transforming data. Each process may be expanded into a new DFD to show more detail, as shown in Fig. 3. Repeating this procedure leads to a tree of simple diagrams rather than one enormous, complex diagram.

Sources and sinks of information (external to the system) are shown by rectangles. They should be labelled with what they are rather than the data they deal with. If a source and sink are the same eg Bank they may be shown once or twice. Fig. 2 shows Bank twice. I tend to show the source and sink as distinct rectangles only when they are the ends of the principal flow through the system.

Processes are shown as circles, labelled with a number (either a global number or relative to that diagram), and containing some text. The text should take the general form of a verb followed by object(s), for example Buy Supplies, where the objects are visible in the dataflows connected to the process. Each process should have at least one input and at least one output.
Figure 3: DFD Notation
Data stores are used to indicate that data should be stored. They should be labelled with what they contain. They may be repeated on a diagram to avoid confusion from crossed dataflows. Readability is enhanced by repeating stores on lower level diagrams, though some Computer Aided Software Engineering (CASE) tools regard this as an error. Dataflows are shown as lines. It is good practice to arrange the diagram such that the most important data flow from top left to bottom right. Dataflows should be labelled. The label may, and to reduce clutter should, be omitted when it would be the same as that of a data store into which or from which it is flowing. A dataflow must have a process at one end, and the other end may connect to any object or to nothing. An unterminated dataflow means that the flow crosses the boundary of that diagram and so should be visible in the next higher level of diagram connected to the process being shown in detail. This is known as dataflow balancing. It is often necessary to show dataflows crossing the boundary of a diagram which are not the whole data flow shown on the parent diagram, but only a component of that flow. This tree structure of the dataflows should be recorded in a data dictionary.

The highest diagram should have just one process (the context diagram) and is the only diagram which may have sources and sinks. It is often convenient to allow it also to have one or more data stores.

All other diagrams should aim at $7 \pm 2$ processes on them, to maximise comprehension of the diagrams.

The lowest level processes may be described by text in what is know as a process (or transformation) specification.

### 2.1.2 Logical and Physical Models

The distinction between the logical and the physical model has been introduced in Sec. 1.1. It is important to understand the difference and to be sure which type of model you are building. One model cannot do both jobs, and much time can be wasted trying to build a model which sways between the physical and logical views of the system;

logical how things would be in a world with computers with infinite memory, and zero cycle times.

physical shows how they really are (or will be)

The logical diagram is useful for understanding what a system does and the physical diagram how it does it. It follows then that the logical model is good for user documentation and to explain what the system will look like; but the physical view is essential for building the system.

### 2.2 Data Modelling

Tschritzis and Lochowsky [5] state at the beginning of their book on data models (my italics):

For data to be useful in providing information, they need to be organized so that they can be processed effectively. ... In data modelling we try to organize data so that they represent as closely as possible the real world situation, yet are still amenable to representation by computers. These two requirements are often conflicting. To determine how best to organize data for a given application, we need to understand the characteristics of data that are important for capturing the essence of their meaning. These characteristics allow us to make general statements about how data are organized and processed. A consistent, formal set of such statements defines a Data Model.

E. F. Codd in his 1981 Turing Award Lecture defined a data model as:

A data model is, of course, not just a data structure, as many people seem to think. A data model is a combination of at least three components:
1. a set of data structure types;
2. a collection of operators which can be applied to any valid instance of the data types listed in (1), to retrieve, derive or modify data from any part of those structures in any combination desired;
3. a set of general integrity rules, which define the consistent database states - these rules are general in the sense that they apply to any database using the model.

Because a data model includes Definition, Manipulation and Validation of data it is very useful at all stages of software production:

*analysis* to understand the fundamentals of the problem,

*design* to produce clear data structures

*coding* using the operators of the model

### 2.2.1 The Entity-Relationship model

The Entity-Relationship (ER) model was proposed by P. Chen in 1976 [6] and since then there have been many theoretical extensions and practical applications of the ideas. The model, which is now part of the ACM and IEEE recommended curricula, is simple enough that the basic concepts can be readily learned, yet powerful enough to be useful. It maps readily onto the relational (tabular) model.

The real world consists of entities and relationships among them. An entity\(^1\) is a ‘thing’ which can be distinctly identified, for example a person, a car, a subroutine, a wire or an event. A relationship is an association among entities. For instance: ‘person *owns* car’ is an association between a person and a car, and ‘person *eats* dish in place’ is an association among a person, a dish and a place.

The information about one entity is expressed by a set of (*attribute*,*value*) pairs e.g. a car model could be:

\[
\begin{align*}
\text{Name} & = \text{R1222} \\
\text{Power} & = 7.3 \\
\text{Seats} & = 5
\end{align*}
\]

Values of attributes belong to different value-sets or domains, for example in the case of a car, *Seats* is an integer between 1 and 12.

Entities may be grouped into entity sets where each entity of the set is of the same type. Entities are of the same type if they have the same list of attributes and relationships. For example the table

<table>
<thead>
<tr>
<th>Entity Set: CarModel</th>
<th>Name</th>
<th>Power</th>
<th>Seats</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1222</td>
<td>7.3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>HZ893</td>
<td>6.8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>R1293</td>
<td>5.4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

represents the entity set CarModel with three entities. Because they are all in the same set they must of course be of the same type. No relationship information is represented in this particular table.

\(^1\)Fowler, in a dictionary of Modern English Usage, warns us:

Entity. The word is one of those regarded by plain people, whether readers or writers, with some alarm and distrust as smacking of philosophy.
2.2.2 The ADAMO variant of the ER model

The ADAMO [7] system for software development stresses the data view. It allows ER structures to be defined, manipulated and validated, primarily within the FORTRAN environment. There are a number of tools accessing a data dictionary, and a subroutine package to support data manipulation and validation. Its use has been described in [8,9]. The model is explained very briefly below, but for more complete information consult the ADAMO documentation [7].

The ADAMO variant of the model and its diagraming notation are compact and highly expressive. Many to many relationships must be replaced by an intermediate entity set and two relationships. This allows a direct mapping onto the tabular model, which is very convenient for actually making use of the model. For the same reason of simplicity, but in this case to simplify navigation between entity sets, only binary relationships are permitted. A form of generalised relationship is also allowed. The attributes of the ER model should be components of the dataflows of the DFDs as explained in Sec. 3.1.

```
<table>
<thead>
<tr>
<th>CarModel</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Name</td>
<td>*SerNum</td>
</tr>
<tr>
<td>Power</td>
<td>Colour</td>
</tr>
<tr>
<td>Seats</td>
<td>MakeDate</td>
</tr>
<tr>
<td></td>
<td>RegNum</td>
</tr>
</tbody>
</table>
```

The entity sets are represented by rectangles with the entity set name separated from the attribute names by a horizontal line. The attribute names are sometimes omitted.

The relationships are represented by an arrow. The double arrow head means that a CarModel may have many Cars and the bar (|) indicates that it is possible to have a CarModel for which no Car exists.

The asterisk (*) is used to show a particular entity in a set may be uniquely identified. For example, a CarModel is identified by its name, but a particular Car needs both the relationship to its CarModel (hence the asterisk on the arrow at the Car end) and its own Serial Number (SerNum). This allows Honda and Renault to allocate serial numbers without reference to any central body.

2.2.3 Examples of standard data structures

Fig. 4 shows the ER representation of a number of standard data structures.

1. The hierarchy (which must have distinct levels). A town has many roads and a road has many houses. Note that genuine hierarchies are rare in the real world, though it can be convenient to contort the truth sometimes. Many roads are common to more than one town and some houses are on the intersection of two roads, yet this form of model is used successfully by the postal services.

2. The tree, unlike the hierarchy, has all its nodes equivalent. The tail end of the arrow shows that each node has one parent except for the root which has none (indicated by the bar). The double headed arrow shows that a node may have many child nodes. The bar next to the double arrow is because leaf nodes have no children.

3. The list is a special case of a tree. This modelling of a list does not allow a member of a list to be a list.

4. A directed graph (e.g. a DFD, a Petri Net or an ER diagram). Here the many to many relationship between the nodes is replaced by an entity set to represent the arcs. Two one to many relationships are then required: one shows the subset of arcs going from a node, and the other defines the arcs going to a node.
2.3 Modelling Control aspects

So far we have looked at DFDs to give a process oriented view, and ERDs to give a data view. Now we consider control. The control model should show the relationship between control signals into a process and the signals the process produces. We consider first the case where the control signals are event signals occurring at a fixed point in time and with no information content. They may be visualised as a single bit arriving.

### 2.3.1 Finite State Machines

The Finite State Machine (FSM) is a simple model able to describe this kind of behaviour. My watch has a stop-watch mode where two buttons may be used to control 'Start', 'Stop', 'Clear' and 'Split time' (i.e. freezing the display without stopping the clock). Fig. 5 is a State Transition Diagram (STD) showing the control features of this stop-watch mode.

Each state which the watch can be in is represented by a rectangle. An arrow from one rectangle to another indicates that such a transition is possible. The arrow is labelled with the event triggering the transition and with the action to be performed as the state changes. If these actions are themselves control signals we have a clean description with just events in and events out. For example, from the initial state Not Counting & Normal Display, an event 1 (pressing button 1) generates Start and Beep signals and moves to a state Counting & Normal Display. Note that there are 4 states, but that their names indicate that they are really compound states. The multiplication leads quickly to a very large number of states as the size of the system increases.

As an alternative description of a State Transition Diagram, a State Transition Matrix (STM) can be used as shown in Fig. 6. The correspondence between the STD and the STM is obvious from the example. The diagram looks simpler than the matrix if there are few transitions possible, but the matrix allows one to check easily that all possibilities have been considered. In this example any button has an effect in any state.

A FSM is very simple to represent on a computer. A state can be represented by either the position of the program counter (STD style) or a variable, where the states to go to for each condition are tabulated (STM style)
Figure 5: Control Stop-watch: STD

<table>
<thead>
<tr>
<th>Condition</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Counting &amp; Normal Display</td>
<td>Counting &amp; Normal Display</td>
</tr>
<tr>
<td></td>
<td>Start, Beep</td>
<td>Start, Beep</td>
</tr>
<tr>
<td>Not Counting &amp; Normal Display</td>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>Counting &amp; Normal Display</td>
<td>Counting &amp; Normal Display</td>
<td>Counting &amp; Frozen Display</td>
</tr>
<tr>
<td></td>
<td>Stop, Beep</td>
<td>Stop, Beep</td>
</tr>
<tr>
<td>Not Counting &amp; Frozen Display</td>
<td>Counting &amp; Normal Display</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td>Start, Beep</td>
<td></td>
</tr>
<tr>
<td>Counting &amp; Frozen Display</td>
<td>Counting &amp; Normal Display</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stop, Beep</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: Control Stop-watch: STM
2.3.2 Petri Nets

Petri, in his thesis, described a net with a set of Places, Transitions, an Input Function and an Output Function. Fig. 7 shows a Petri Net representation of the states of my watch. The transitions are shown by heavy lines, the places by circles, the input functions by arrows from a place to a transition and the outputs from the transition to the place. Each place may be occupied. A transition is enabled if all of its inputs come from occupied places. Any enabled transition is allowed to fire at random. When a transition fires its inputs become unoccupied and its outputs all become occupied.

![Petri Net Diagram]

Figure 7: Control Stop-watch: Petri Net

To allow the Petri Net to communicate with the outside world, one allows certain places to be associated with event signals. So placing a token on the place labelled 1 corresponds to pressing button 1. Each transition may be associated with an action which may be the generation of an output signal. Consider the system with a token on Not Counting and one on Normal Display; this corresponds to the starting state of the FSM which was a compound state. This is now clear as it requires two tokens to describe the system. If a token is now dropped onto place 1, then exactly one transition is enabled, the transition will fire removing its two input tokens and set an output token on the place Counting. The Normal Display token is untouched. A Petri net showing the effects of the 4 buttons on my watch would not be unreasonably complex.

In the original Petri net, a transition would only fire if all its input places were occupied and all its outputs were empty. This condition is now often relaxed. Any FSM can be written as a Petri net. Various other extensions have been made to make them more useful, some of these are just notation to simplify drawing, but the significant thing is to be able to test for
empty places. Ref. [10] provides both an introduction to Petri Nets, and a description of their application in modelling.

Ward and Mellor [1] describe a useful hybrid: a dataflow diagram which can be executed as a Petri net and which contains certain nodes whose behaviour is that of an FSM. Fig. 8 shows a DFD of the stop-watch with this notation. The dotted lines are control flows, which correspond to events happening at a moment in time. So the input signals labelled 1 and 2 are control flows which are the inputs for process 6 Control Stop-watch. The outputs of this process are all control flows, so the process is a control process and is shown also by a dotted line. The behaviour of this control process has already been described by the FSM of Fig. 5, which accepted event type signals and the only action of which was to generate such signals. Process 2, Gate Clock, is a simple FSM with 2 states: Pass Clock Signals and Block Clock Signals. Some dataflows are continuous, for example a voltage level. This is represented by a double arrow head, as on the Time output from process 3 which may then be sampled by processes 4 and 5.

Control, which had been eliminated in the move from SADT to SA, is now a fairly respectable part of SA.

2.3.3 Extended Systems Modelling Language (ESML)

Hatley [11] developed methods similar to Ward-Mellor, but one major difference was that he generally takes the control flows to be levels instead of signals. Sometimes this is convenient, sometimes not.

Ward [12] and 3 other authors have produced yet another promising hybrid: ESML, which combines the features of Hatley with the old Ward-Mellor style. The basic extension is to have continuous non-value bearing flows and to allow the control processes to react to them, as well
as to values in stores (I refer to both as levels) and to events. So a transition is labelled by a number of conditions which must all be true. One of them must be an event, but there may also be any number of continuous levels to test. This can produce something very similar to a Flow Chart!

This modification allows the STD to be split in a natural way as we can now make the reaction to button 2 depend upon what state button 1 has left the system in. This is shown in Fig. 9 and 10 where we see that the condition C (counting) generated by Control Start and Stop is continuously available to Control Clear and Freeze

![Diagram](image)

**Figure 9: Stop-watch ESML style: DFD**

![Diagram](image)

**Figure 10: Stop-watch ESML style: STDs**

ESML also defines a depletable store where read is destructive, it supports a set of special event signals: Enable, Pause etc, with predefined meanings and it allows the state of the outputs for a disabled process to be defined.

### 2.3.4 Comments on the real time methods

Some extensions to the simple DFD are essential to be able to describe a real time system adequately. The Ward-Mellor notation has been in existence for a while and a path through to coding is defined. The latest Ward variants (ESML) do allow the control system to be divided further and, by providing a wider vocabulary, make it easier to model a system. Of course the
new vocabulary has to be learned. I have noticed that users of the Ward-Mellor notation often cheat a little, in the direction of ESML. So, though I like the look of ESML and believe that it is the right direction, I will use the 'standard' Ward-Mellor notation for the rest of these notes.

3 From concept to a physical system

It is now time to see how we may go along the path to building a system as introduced in Section 1.1. Some steps may be trivial, it depends upon the project.

Concept → Logical Model → Physical Model → Physical System

3.1 Construction of the Logical Model

To build a logical model, produce one large diagram, concentrating on control. Around the edge of this diagram will be all the external event flows and data flows. It is important to be clear where the boundaries of the system you are modelling lie. Use ERDs to describe all the data flowing. Note that many data flows will be composite, and only when broken down will the basic attributes be apparent. Nothing should appear in these diagrams that could be missed out. If there are any value bearing dataflows linking processes then merge the processes. Having done this the context diagram can be drawn (the system as one process), and the diagram broken down into a set of tree structured diagrams. It is easier to do the job this way round rather than to do a top down design for the overall system, though in both cases the result is a set of tree structured diagrams.

Consider the watch; we already have the flat logical diagram as in Fig. 8. As it already has seven processes there is no need to divide it up. The context diagram can then be readily drawn. This is shown in Fig. 11

![Figure 11: Context Diagram of Stop-watch](image)

Now consider a major element of a reconstruction program Find Tracks. Fig. 12 shows a context diagram for a system which takes 3-D points from the detector and produces tracks and some book-keeping information. We can also draw the ER Diagram, as in Fig. 13, to describe the data of most interest: the Point and the Truck. As there are no control issues, the flat logical diagram is the same as the context diagram, i.e. there are no processes which are essential to the task because there are no control signals. We should note that one of the SADT rules says that every box should have a control input, otherwise the boxes should be merged; so analysis has much to do with control. We can draw a logical diagram but it will not be the 'Essential Logical Diagram' because it involves making design decisions. It is useful, nevertheless, to produce a diagram showing some detail, because these high level design decisions are important, and it is good to make them in a way which is not influenced by physical constraints so that all concerned people can judge the overall processing strategy at an early stage.
Figure 12: Find Tracks: Context Diagram

Figure 13: Find Tracks: ER Diagram

All the smallest elements of the data flows should be attributes of entities in the ER model, except for data structures which are already defined. So to produce the ER model, one way is to get all the attributes and group them to define entity sets. Identify the relationships between the entities. In general, entity sets with one to one relationships between them should be replaced by a single entity set, as should entity sets which are only pointed to by one other entity set and have only one attribute. In practice the ER model is often built in parallel with the dataflow analysis; it will need refining when the physical model is constructed.

The data should be described using a Data Definition Language. There are a number of variants. I will describe the ADAMO DDL [7], which is comprehensive, allows documentation to be generated automatically and allows data structures to be created which can be manipulated in the FORTRAN environment. The DDL for Find Tracks, shown in Fig. 14, may be most readily generated on a VAX with the Language Sensitive Editor for which templates for the ADAMO DDL are available, it may then be processed by the ADAMO tools.

3.2 Construction of the Physical Model

The logical model was the ‘best’ way of doing things in the ideal world, so one should minimize distortion of the logical model when deriving the physical model from it. One must take account of multiple CPUs running multiple tasks, and finally make use of functional decomposition to complete the design of each task. One must meet the practical constraints imposed by finite speed of execution and by finite sizes of memory and of disk while going through the steps.

3.2.1 CPU Allocation

Here the task is to partition the model between CPUs to balance CPU power and to minimize movement of data between CPUs. The cost of a mistake at this stage could be high if the CPUs are very different from the software point of view.

One must think about data really flowing, and consider the time it takes. The data flows
SUBSCHEMA FindTracks

: 'Logical model of data for track finding'

DEFINE ESET

Point  = (Coordinate(3) = REAL : 'Space coordinates of point',
          Error(6) = REAL : 'Compressed 3 by 3 covariance matrix')
        : 'A point in space';

Track  = (Coordinate(6) = REAL : 'Track coordinates',
          Probability,
          Length)
        : 'A track in space';

END ESET

DEFINE RSET

(Point [0,1] -> [3,=*] Track)
     : 'A track is associated with at least 3 points';

END RSET

DEFINE ATTRIBUTE

Probability = REAL [0,1.000]
              : 'A probability of something';

Length     = REAL [0,=*]
              : 'A length must be positive';

END ATTRIBUTE

END SUBSCHEMA

Figure 14: Find Tracks: DDL
between the CPUs should be clear from the new model. Processes on the logical diagram may need to be split. Fig. 15 shows the CPU allocation to build a stop-watch which will behave as the logical model. I have chosen to allocate all of the processes to a 'mini-computer' except for the task of producing the clock signals, which is at process 1 of Fig. 8. The only communication between the devices is the clock signals.

![Image of the Stop-watch CPU Allocation diagram]

Figure 15: Stop-watch: CPU Allocation

### 3.2.2 Task Allocation

The procedure is similar to CPU allocation. Though it would seem attractive to have simply one task per process the system would be limited by context switch time, communication delays and the maximum allowed number of tasks; also simulated concurrency can result in more lock-ups than would occur with real concurrency. So again one must compromise.

It is good to keep data and control transformations separate, though small data transforms driven directly by a control transform may occupy the same task as the control transform. Multiple control processes within one task should be merged, and continuous data flows and transformations must be eliminated. Generally add a flag datastore to multiple data transformation processes within one task, to indicate which function they are supposed to be performing, unless they are simple processes sharing a task with their control process.

Fig. 16 shows a task allocation of the Soft Part of the Stop-watch, where Format Display is a task with one process of the same name and the other processes go into a task: Maintain Time Stores. The Saved Time datastore is to eliminate the continuous data flow into Format Display. This task is activated by the Normal and Frozen signals which select the information to be displayed, and by Resume which is generated by Maintain Time Stores when it is time to update the display, e.g. every new minute.

Fig. 17 shows inside the task Maintain Time Status, the two control processes of Fig. 8 have been merged into one which then controls the other minor processes within the task. Merging the control processes produces the State Transition Matrix shown in Fig. 18, which ensures that the Clock signal is transmitted if the system is in one of the two counting states, otherwise the signal is ignored.
Figure 16: Stop-watch (soft part): Task Allocation

Figure 17: Stop-watch: Maintain Time Stores
### Figure 18: Stop-watch: Control Stop-watch and Gate Clock

<table>
<thead>
<tr>
<th>Condition</th>
<th>1</th>
<th>2</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>State ↓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not Counting &amp;</td>
<td>Counting &amp;</td>
<td>Not Counting &amp;</td>
<td></td>
</tr>
<tr>
<td>Normal Display</td>
<td>Normal Display</td>
<td>Normal Display</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beep</td>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>Counting &amp;</td>
<td>Not Counting &amp;</td>
<td>Counting &amp;</td>
<td></td>
</tr>
<tr>
<td>Normal Display</td>
<td>Normal Display</td>
<td>Normal Display</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beep</td>
<td>Save Time,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frozen</td>
<td></td>
</tr>
<tr>
<td>Not Counting &amp;</td>
<td>Counting &amp;</td>
<td>Not Counting &amp;</td>
<td></td>
</tr>
<tr>
<td>Frozen Display</td>
<td>Normal Display</td>
<td>Normal Display</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beep</td>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td>Counting &amp;</td>
<td>Not Counting &amp;</td>
<td>Counting &amp;</td>
<td></td>
</tr>
<tr>
<td>Frozen Display</td>
<td>Normal Display</td>
<td>Normal Display</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beep</td>
<td>Clock</td>
<td></td>
</tr>
</tbody>
</table>

#### 3.2.3 Functional Decomposition

The remaining data transforms may need functional decomposition if they are not simple enough. Functional decomposition is *design*. One must think of a network of processes able to do the job specified by the parent process. Net input and output flows must match the parent. Try to avoid thinking of just a chain of processes.

For a logical model there is no point in decomposing beyond what is needed to make the function of the system clear, but for the physical model, the usual recommendation is to go down to a bubble which performs a simple task and which could correspond to one subroutine of say 10 to 100 lines of code (excluding comments).

There are problems in going down too far however, as one more level gives an eightfold increase in the number of diagrams to be maintained. Also the person coding is over constrained by the design given to him. Note that a detailed physical model contains a great deal of design (*how*) and not just analysis (*what*). Small changes to the code will result in the diagrams having to be continually modified if the right bottom level processes have not been correctly identified in the DFDs. And finally if one expands too far, the diagrams look either like a simple chain of processes or a flow chart according to the style of the author.

#### 3.2.4 Data Transform Specifications

You may wish to write specifications for lowest level data transforms of the logical model. They should anyway be written for the leaf processes after functional decomposition. It is good practice to use pre-conditions and post-conditions to enforce the *what* rather than the *how*. For example, to find the maximum of two integers, one could write:
Pre:  \( x \) and \( y \) are integers & 
    \( result \) is an integer variable

Post:  \( result \geq x \) & 
        \( result \geq y \) & 
        \( result = x \) or \( result = y \)

This does not indicate how the job should be done. In fact a practical algorithm must make 
early use of the last part of the post condition. As the conditions may be expressed in an 
informal manner, *Fit Tracks* can also be specified this way.

### 3.3 Building the physical system

It is now time to design and build the physical system; it is here that structured design (SD) 
should be used [13,14]. The outcome of SD is a structure chart, as in Fig. 19. Module A calls 
module B inside a loop, receiving a control flag and data \( \text{Data1} \) and \( \text{Data2} \), then calls either 
C, giving it \( \text{Data1} \), or D, giving it \( \text{Data2} \).

![Figure 19: Structure Chart Notation](image.png)

The idea is rather old; it was noted that successful programs had a shape rather like an 
onion, with the main routine at the top, widening out to a large number of routines, and then 
narrowing down to some service routines. The problem is to derive such a structure chart from 
the results of structured analysis. There is such a procedure, described for example in [15] but 
it does not lead to very attractive designs. Some of the principles are good, such as: ‘produce 
modules doing just one job and with simple interfaces’, or, ‘reduce access to common data stores 
by hiding them behind an access package to increase data independence’. But blind adherence 
to the rules is unlikely to produce a good design; in particular it is not going to result in clearly 
defined layers of software.

#### 3.3.1 Packages and Prototyping

Methods so far have left the ‘programmer’ nothing to do until all the design has been perfected. 
He need not be idle however, because at a very early stage it is possible to identify useful utility 
packages, and do some prototyping.

Designing a package early in the life cycle is in some respect bottom up design. But it may 
be thought of as the analyst making a quick mental analysis and set of designs, stirring in some 
experience and defining a package which he thinks will be useful. There is of course some danger 
that this may constrain the final design.

Prototyping is a nice idea, but hard in practice. For a commercial system the client may 
want so see, and try, the system at an early stage. If he is only concerned with the outward 
appearance, then this is not too hard. Consider an automatic bank teller machine which will
dispense cash, provide information on current balance etc. It should be easy to put together the
displays with the right tools, and then demonstrate the user interface to ensure that it is right.

Prototyping is really needed when you are not sure what you want, but this is often the case.
There is a danger that the existing prototype may compromise the design because ‘the code is
already written’.

3.3.2 Coding: ADAMO

We have seen that a data model includes definition, manipulation and validation of data.

Ward and Mellor [1, Vol. 3] in 1986 advocated a system for holding relational tables in mem-
ory (for speed), with their definitions derived automatically from the DDL. ADAMO provides
this and more (and did so before Ward and Mellor published their book).

Having completed the ER modelling with ADAMO, you have the data structures defined for
you. The TTable Package (TAP) provides routines for all the basic ER operations, input/output
of ER structures and validation. The table below shows some of the operations available:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Entity Set</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert into</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Replace in</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Delete from</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fetch from</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Select from</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Navigate</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

For those leaf processes defined by pre and post conditions as explained in Sec. 3.2.4 a subroutine
can be structured in 3 parts. The first to check the pre-conditions as far as possible, the second
to apply some algorithm, and finally if it is wished to check the algorithm the post-conditions
can be tested. The TAP itself makes use of pre-conditions; they may be switched off to save
time. Post-conditions are not coded.

4 Tool support for SA-SD

Many manual procedures are associated with software production. CASE tools help to con-
struct, control and correlate all the ‘documents’ needed to build a system. They should increase
productivity, and allow controlled measurable progress from the concept to the final system.
There are a quite a number of tools available, mostly now workstation based, though some
(with reduced functionality) are available on the IBM PC.

Unfortunately the tools are expensive and it is hard to find one which offers sufficient flexi-
bility. The methods are changing, and the suppliers of CASE products cannot hope to keep up
unless those products are designed correctly. I consider it essential that the tool allows definition
of diagrams and the constraints to be applied to these diagrams. At least one example should
be supplied for each major type of diagram as a starting point.

I also consider it essential that the tool should be able to correlate elements of different
models of the same system, otherwise, for example, it is hard to be sure that the physical model
corresponds to the logical.

As CASE tools are introduced to HEP we must consider standardisation, which always slows
things down. However I hope we do not have to wait much longer to see CASE tools on every
workstation, for it is only when they are taken for granted, like the FORTRAN compiler —
which must also be paid for — that the full benefits may be realised.
5 Conclusion

I hope I have shown that the use of SA-SD encourages clear thought, communication, good documentation and finally good programs. It is very much in the spirit of SA-SD to adapt the methods for your own needs. I believe that we are currently hindered by not having easily affordable, flexible CASE tools with the right functionality.

It is important to know when to use which methods; I would suggest:

‘Off-line’ programs e.g. Reconstruction - This type of program has more or less no control issues. ER and DFD are found to be important.

Packages - In developing a package, we are developing modules which may be used as the leaf nodes of many DFDs. As we deal only with leaves and their use of a data structure, ER is important.

‘On-line’ programs - These do little processing of data. The data structures are again important, but DFDs are very simple. It is essential to distinguish between the logical and the physical model as they tend to differ significantly. Control and timing issues are crucial in the multi-tasking environment.

I recommend that you try to apply at least some of the methods as soon as you get the opportunity.

6 Acknowledgements

I must thank John Harvey, Gottfried Kellner and Paolo Palazzi for useful discussions on SA-SD and, along with others, for providing constructive criticisms of these notes.

This text was processed by \LaTeX{} and the diagrams were prepared with the the SIGHT graphics editor on a VAXStation. The diagrams were then converted to PostScript by the RENDER command and slightly edited by hand to get the size and overall position right. Finally the whole lot was converted to one PostScript file by the PSPRINT program from Andrew Treverrow and sent to a PostScript printer. I am grateful to Dave Kelsey and Jason Leake for installing the software and running the hardware necessary for this work.

References


1. INTRODUCTION

What makes reading a book or an article pleasurable? The discussion in this paper will be limited to the production of scientific documents: the success of novels, poetry, newspapers, tabloids or advertisements being outside the scope of the School.

For most scientists, I would like to believe that the contents of the article come first. I always open with relish every new issue of *Scientific American*, hoping to find articles in which I can discover something new, especially in the fields in which I have kept an active interest like physics, astronomy, computer science, mathematics or engineering.

Some articles - it is also true of certain books - are easily read from beginning to end. One concept follows another in a well-constructed way. A well-structured document assists in the understanding of the reader. Printed material does imply a linear approach: having to move back and forth in trying to relate text, formulae and figures can be quite discouraging.

Pleasure also comes from a nice presentation. One can be attracted to a book by the quality of its printing, the readability of mathematical formulae and the beauty of illustrations. Sometimes, however, it can be overdone. I receive another monthly magazine called *National Geographic* in which the pictures are so beautiful and breathtaking that I rarely read any of the text apart from the legends of the photographs. I don’t mean to criticize the *National Geographic* which I enjoy very much, but to warn the scientific writer not to distract his reader from the essential message to be conveyed!

Economic considerations have changed the way scientific publications appear. In the old days (before 1965), scientific books and journals were usually typeset from manuscripts submitted by the authors. Those of us who published during that time remember all the cutting and gluing before submitting an article and the difficulty of having professionals draw the necessary illustrations in a timely fashion. If an index was required, it had to be done by hand, a very long, fastidious and error-prone process. After long delays, galley proofs would come back with a very short deadline for submitting corrections; extensive ones were generally forbidden but, if absolutely necessary, were charged to the author at a stiff price.

Let me quote Donald Knuth from his book "*T, X and Metafont, New Directions in Typesetting" [1]:

"A greatly increased volume of publication, together with the rising salaries of skilled personnel, was making it prohibitively expensive to use traditional methods of typesetting, and the [American Mathematical] Society eventually had to resort to a fancy form of typewriter composition that could simply be photographed for printing...

"At this point, I regretfully stopped submitting papers to the American Mathematical Society, since the finished product was just too painful for me to look at. Similar fluctuations of typographical quality have appeared recently in all technical fields, especially in physics where the situation has gotten even worse."

In the mid 70's and the early 80's, an all-time low was reached with the so-called camera-ready copies. In publications involving programming, people tried to include, without retyping, line-printer or teletype output which was almost impossible to decipher.

Improvements in output devices, from the daisy-wheel letter quality printer to the currently popular laser printers avoided the catastrophe which was looming over scientific publishing, namely to publish more and more papers that nobody would care to read.
During this time, the world of typesetting was undergoing its own revolution. Various kinds of phototypesetters were replacing the old linotypes and monotypes in spite of a strong opposition by labor unions responding to the laying off of entire classes of workers. The computer had to be hidden inside the casings and retrained typographers composed books by embedding low-level commands into running text. Except for the machinery being different, typesetting by professionals remained stubbornly conservative.

Once cathode-ray tubes were used in phototypesetters, it was an easy step to apply the algorithms for converting arbitrary sets of figures to pixels. Soon, all sorts of character fonts started to be generated "but it has been a huge culture-shock for computer scientists and mathematicians entering this field to discover that the 'obvious' method of using arcs and lines to outline a character is the subject of a patent which has already featured in a spectacular out-of-court settlement"[2].

The leading authority in this field is Donald Knuth who gave computer users true typesetting capabilities, with the language \( T_X \), (for \( \text{Tau, Epsilon, Chi} \)).

**II. OFF-LINE SYSTEMS**

Knuth was the first person who dared tackle the problem of typesetting in all its intricacies, including font generation with different sizes and styles (bold, italic, ...), justification which makes sense only with proper hyphenation, proportional spacing and kerning, and full page layout. As an example, this document is printed with a Times 10-point font. Knuth's approach is based on a proper mathematical model of typographical decisions. Other systems, sometimes with much less capabilities, are well-known to computer users who want to compose their own documents from their terminal connected to some mainframe. These are summarized in Table I.

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Author</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977</td>
<td>NROFF/TROFF</td>
<td>Ossana</td>
<td>Bell Labs</td>
</tr>
<tr>
<td>1979</td>
<td>TEX and METAFONT</td>
<td>Knuth</td>
<td>Stanford</td>
</tr>
<tr>
<td>1980</td>
<td>Scribe</td>
<td>Reid</td>
<td>CMU</td>
</tr>
<tr>
<td>1982</td>
<td>A typesetter independant TROFF</td>
<td>Kerighan</td>
<td>Bell Labs</td>
</tr>
<tr>
<td>1986</td>
<td>Standard Generalized Markup Language</td>
<td>Kernighan</td>
<td>ISO 8879</td>
</tr>
<tr>
<td>1986</td>
<td>Waterloo SCRIPT GML</td>
<td>Lampert</td>
<td>MIT (?)</td>
</tr>
</tbody>
</table>

Table I: Most significant off-line document preparation systems

Mainframe text processing systems are much older (see for example ref. [3]). Quoting van DAM who proposed Hypertext as early as 1969, we can state that: "we want systems that support scholars in their role as researchers and authors". The requirements for good systems have not changed, they are:

- **maintainability**, which means no corruption of the original text by editing commands but revisions or new editions are easy to make;

- **exchangeability** by the virtue of which several authors can work on the same document, publishers don't have to spend money for re-keying, proof reading of final texts becomes unnecessary and entire works can be included in textual databases (this in turn raises copyright issues which will not be discussed here);

- **portability** which offers some independance from printing devices and the possibility to move documents from one system to the other (between \( T_X \) and SGML for example);
facilities for structuring the document will help in handling footnotes, references, automatic indexing, setting up headers and including them in a table of contents;

tools for content preparation ought to be easy. Scientists are prepared to accept the rather clumsy page editors available on terminals designed for programmers, these are less palatable for secretaries, especially if they have access to a word processor. LATEX [4] was specifically designed to facilitate the use of T^E_X.

Off-line systems are quite good at describing mathematical formulae. Although Figure 1 was produced by a WYSIWYG system on a SUN, it could have been produced equally well on a VAX (Table II) or the IBM 3090 (Table III) by the code listed below.

$$Q_2(x) = \frac{x^{3/2}}{\pi^{3/2}} \sum_{n=1}^{\infty} \frac{U(n)}{n^{3/2}} \frac{\cos\left(2\pi \sqrt{nx} - \frac{7\pi}{4}\right)}{\sqrt{2\pi} \sqrt{nx}} + O\left(x^{3/4}\right)$$

$$= \frac{x^{5/4}}{\pi^{3/4}} \sum_{n=1}^{\infty} \frac{U(n)}{n^{7/4}} \cos\left(2\pi \sqrt{nx} + \frac{\pi}{4}\right) + O\left(x^{5/4}\right)$$

Figure 1: Example of a rather complex expression

One can see that editing of formulae can easily be done on off-line systems. They perform the fastidious tasks of aligning and balancing the different symbols, but it may take a few trips to the printer before the results are satisfactory. Handling complex mathematical expressions remains one of the stumbling blocks of word processing systems and convenient tools are only starting to appear for document preparation systems.

\section{mathematical formulae}
\begin{array}{r}
Q_2 (x) & \displaystyle \frac{x^{3/2}}{\pi^{3/2}} \sum_{n=1}^{\infty} \frac{U(n)}{n^{3/2}} \frac{\cos\left(2\pi \sqrt{nx} - \frac{7\pi}{4}\right)}{\sqrt{2\pi} \sqrt{nx}} + O\left(x^{3/4}\right) \\
& \displaystyle \frac{x^{5/4}}{\pi^{3/4}} \sum_{n=1}^{\infty} \frac{U(n)}{n^{7/4}} \cos\left(2\pi \sqrt{nx} + \frac{\pi}{4}\right) + O\left(x^{5/4}\right)
\end{array}

Table II: T^E_X input for formula in Fig. 1

Table III: Waterloo GML/SCRIPT [5] input for formula in Fig. 1
III DESKTOP PUBLISHING

"Everybody understands what you mean when you say desktop publishing,... The term is not old, and its origins are readily traceable. Paul Brainerd of Aldus, father of PageMaker, gets credit for coining the phrase. Apple Computer, looking for a vehicle to dramatize the Macintosh's capabilities, had the perspicacity to go all out on a promotional campaign. Without the Apple LaserWriter, desktop publishing would never have existed - desktop publishing didn't really make its debut until Apple announced the LaserWriter in January 1985"[6]. Desktop publishing had a pre-history briefly summarized below in chronological order.

It originated in 1973 with the Alto, a multi-function workstation developed at XEROX Palo Alto Research Center (PARC) and never announced as a product although 1200 were built. Alto included a coupling to the first Ethernet, a high resolution display and a mouse-pointing device. Various software packages were developed for it including Bravo, Gypsy, Markup and Draw for text editing and formatting as well as for manipulating images. Later (1981), XEROX announced the STAR, derived from the Alto, but it offered only an expensive printer and it had no commercial success.

The next big event was an "affordable" laser printer, the LBP-10, announced by Canon in 1979. The IMAGEN company managed to print full page of text and graphics in 1981.

After seeing what was happening at XEROX-Parc, Steve Jobs, co-founder of Apple Computers, developed the Lisa with a complete set of office applications but only a dot-matrix printer. At about the same time, Niklaus Wirth designed and built a machine programmed in Modula-2 and called the Lilith. It had a powerful document preparation system called Andra but required an expensive laser printer.

Later, Apple moved away from Lisa and introduced the Macintosh (1984), but it was not until January 1985, when the LaserWriter made its debut, that the "desktop publishing metaphor" captured the fancy of an ever increasing number of users. PageMaker by the ALDUS company has led the way in offering professionally designed fonts fully supported by ADOBE's Raster-Image-Processor.

There are currently three main platforms for desktop publishing systems:

- Macintosh for which the BYTE survey of May 1987 lists 10 offerings with PageMaker the undisputed leader;
- IBM PC/AT/PS compatibles with 32 offerings, which should preferably be bought as turnkey systems because of the peculiarities of the graphics cards supported. PageMaker and Ventura (a XEROX product) are the current leaders but many word processors are being upgraded to a point where it will become difficult to tell the difference;
- UNIX based workstations which support some older, expensive products for professionals, like INTERLEAF - although today it offers more competitive prices - and a number of aggressive newcomers like FrameMaker and The Publisher (from ArborText).

In general, workstations are more adequate for preparing long documents: repaging a whole chapter can take quite a long time on a machine with slower disk access and less computing power. Access to Ethernet makes the sharing of information a natural feature while the multi-window system makes it trivial to import stuff from other systems by just clicking the mouse. But once one is hooked to a given system, one tends to favor it over other, less familiar products.

The list of what I consider desirable functionalities of document preparation systems stem from a study of available systems which was conducted in the Spring of 1987 as part of a procurement process which eventually led the University of Geneva to buy over 100 SUN workstations and 30 laser printers installed in different departments. The document preparation system is FrameMaker for which 65 floating licences have been bought. The fact that we are quite pleased with this solution does not mean that the same study, conducted today, could not lead to a different solution.

Without much training, it is possible to produce visual aids of quality incorporating text and graphics. Figure 2 is an example which was used in the lecture. It took only a few minutes to prepare. To use all the functionalities offered by the system is a longer process. It has to be decided whether to become a really proficient typesetter, possibly at the expense of other activities. Leaving the specific roles of scientists, technical writers and secretaries to studies in computers and society, let us concentrate on the functionalities.
DESKTOP PUBLISHING

Content preparation

word processor
include computer output
scan and edit images

spelling checker
search and replace
copy stuff
printer code (postscript)
pixel editor
scale, rotate, ...

Structure

anchored footnotes, pictures and legends
automatic indexing
headers and table of contents

Presentation

laser printers
device independence
WYSIWYG

many fonts available
postscript
what you see is
(more or less)
what you get

Figure 2: Example of a slide prepared with FrameMaker
1. Input

All the functions of a good word processor are needed. Workstations under UNIX may have problems with the direct key-ing of European languages. Scholars in the humanities would like to work with ancient Greek, Arab, Cyrillic or Hebrew, implying that corresponding fonts should be available on the screen.

Importing files from other word processing systems should be easy. The files are sent over Ethernet via ftp and they are operated on by filters which incorporate their content in the document according to the suffix of the files (.asc for ASCII, .pc for PC-DOS, .dif for Document Interchange Format).

Some graphics preparation tools are usually included. Getting in graphics from other sources (bitmap, GKS, PostScript) is tending to become a standard facility. Some skill is required to include snapshots of parts of the screen in the current document.

2. Manipulation

Search and replace are always found but do not always work on special symbols. They do not involve the text entered as part of a picture. Hyphenation is very language specific. Even in English, intelligence could be built into the system to avoid cow-orker or the-rapist among others.

Editing pictures or mathematical formulae in a WYSIWYG fashion is quite difficult on some systems, if not impossible, but things are changing quickly and the latest versions from several vendors are responding to the challenge of the older "off-line systems".

A lot of effort has gone into the correct handling of hangers, widows and orphans, to give powerful ways of preparing multi-column documents. In the same way, anchoring footnotes and images to text works well. Making an automatic table of contents or an index requires a careful study of the reference manuals, which most users refuse even to consider.

3. Output

Two media must be supported concurrently in a WYSIWYG system: the screen and the printer. Many fonts can be designed for a 300 DPI (Dots Per Inch) laser printer but it will be a challenge to make them appear as good as they are on the screen with only 70 DPI. Good fonts are hard to design and it would be overly costly to implement them independently on different pieces of hardware.

PostScript [7], brought a solution to this problem. It is a fully developed Page Description Language to be interpreted by the processor which is part of the laser printer (see Figure 3). Introduced with the Apple LaserWriter in 1985 by Adobe, it is becoming a de facto standard in spite of earlier work done at XEROX Parc which announced Interpress in 1982 and a Document Description Language (DDL) supposed to correct PostScript main weakness: a very long time to process complicated pictures.

![Figure 3: The PostScript files are interpreted in the laser printer](image)
IV EXPECTATIONS

One must recognize that desktop publishing has come a long way in only 3 years. I expect many pleasant surprises in the near future, especially for handling mathematical formulae.

The UNIX world promises a full 8-bit version with open look, a much better human interface, and X.11 which should handle PostScript on the screen, removing many of the current restrictions on font transformations.

New hardware is very likely to come up with faster processing for PostScript, introducing some parallelism. The main progress for workstations could come from the video industries with new TV providing better screens and read/write video-disks storage space for vast libraries of images.

Artificial Intelligence may offer help with grammar and style, document composition and picture enhancement. Voice input and automatic translation are somewhat further away.

ACKNOWLEDGMENTS

I wish to thank Jean Bunn-Richardson for improving the manuscript. She also did most of the work concerning the evaluation of the document preparation systems, on which I borrowed heavily. Paul Bartholdi and Stephen Franklin contributed to my understanding of the issues involved.

My grateful thoughts to my SUN workstation and its copy of FrameMaker for not causing any major catastrophies during the writing of this article.

REFERENCES


Reference Manuals
PageMaker (Macintosh, PC)
Ventura Publisher (PC)
FrameMaker, (SUN)
Interleaf, (SUN, APOLLO, DEC, HP)
The Publisher (SUN).
INTERPERSONAL COMMUNICATIONS USING COMPUTERS

A.J. Casaca

INESC, Lisbon, Portugal

ABSTRACT

The most important telematic services for interpersonal communications supported in computer networks are studied by overviewsing their main features and the present state of standardization development; this study includes Electronic mail, Document interchange, Directory service, Group communication and Videotex services. Finally, the concept of the emerging Integrated Services Digital Network is introduced and a reference is made to its impact on interpersonal communications.

1. INTRODUCTION

The use of interpersonal communication services supported in computer networks is already well established in some communities of users. This activity is being fostered by the increasing availability of wide area computer networks in many regions of the globe, and a further use of these services is foreseen in connection with the progressive integration of data and voice communication networks.

A computer network consists of a number of computers that can communicate among themselves by using the same protocols over a transmission medium. Wide area computer networks, which may cover long distances between the users, and whose utilization is many times offered as a public service, are particularly suited to interpersonal communication as they assure a widespread connectivity among the users.

The diagram of a wide area computer network (WAN) is shown in Fig. 1. The WAN is physically constituted by a set of switching nodes that establish the different communication paths between the user terminals and/or the computers, and by the transmission media, which may include cables, optical fibers, microwave circuits and satellites.

A WAN can be either circuit-switched or packet-switched. In a circuit-switched WAN there is a physical path established between the users for the complete duration of the communication, much like in the Telephone network. In a packet-switched WAN it is not necessary to establish a dedicated path between users. Every message to be sent is divided into blocks, called packets, which are passed along the network from node to node until they reach the destination where they are reassembled. A public owned WAN is normally called a Public Switched Data Network (P SDN), and the majority of the existing PSDN are packet-switched.

The user access to the network is made through a communication subsystem, existing either in the terminal controller or in the computer. To guarantee a uniform access to the network
to all users, the PSDN has a standardized user-network interface: X.25 in a public packet switched data network (PPSDN) and X.21 in a public circuit-switched data network (PCSDN).

The importance of WAN in general, and of PSDN in particular, for interpersonal communications, is dependent on the type and number of services available in the network. The services offered in a data network are mainly oriented for text, graphics and data communication; this type of services, i.e., non-voice services, are called telematic services [1].

Two different types of telematic services exist: network oriented and terminal oriented telematic services.

The network oriented telematic services have the following characteristics:
i) the communication is done via a network storage, implying that there is no direct end-to-end communication between the user terminals;
ii) the addressee is always a person to whom a mailbox is allocated;
iii) the message is stored in the network, and the user has to access the store to get the message.

On the other hand, the terminal oriented telematic services have the following characteristics:
i) there is a direct end-to-end communication between the user terminals;
ii) the addressee is always the number of the called terminal;
iii) the received message is in the terminal.

There are five network oriented telematic services that assume particular importance for interpersonal communication. These services are the Electronic mail, Document interchange, Directory, Computer conferencing and Videotex.

In Electronic mail, users at distinct points of the network exchange interpersonal messages.
In the Document interchange service, documents containing text and graphics are exchanged between users, keeping the same document format at both ends, being also possible for the receiver to edit and process the document as at the originator's site. The Directory service provides a structured storage of information on the network resources and gives the possibility to the users of accessing that information in a friendly way. Computer conferencing is a tool for communication between more than two people, in which the communication is organized into a number of distinct communicating groups. Videotex is a service that allows the users to access a remote computer, containing particular types of databases, such as stocks, share prices, travel information, news reports, etc.

There are also other network oriented telematic services that are being standardized and should be offered in data networks in the future, although they are not specifically oriented for interpersonal communication. They are the File Transfer and Management (FTAM) [2], Virtual Terminal Protocol (VTP) [3] and Job Transfer and Management (JTM) [4].

The terminal oriented telematic services that are presently available as public services are the Telex, Teletex and Facsimile.

Telex is an old service, originated in 1933, which uses a specific network. It is presently the second telecommunication service in the number of subscribers, after the telephone service. However, its low transmission speed (50 bit/s), the use of a restricted character set and the requirement for a special purpose network are serious limitations for further expansion of this service.

Teletex is a new service that may be considered an evolution of telex, in which the complete character set of an office typewriter may be exchanged between similar terminals at a speed higher (2400 bit/s) than in the telex service. The present offering of this service is very limited, and some interrogations exist concerning its future, as it requires special purpose terminals and it overlaps the operating environment of Electronic mail [5].

Facsimile is a facility for transmitting scanned images of documents electronically between similar terminals. It is a service in expansion, which is presently based on the public switched telephone network (PSTN), therefore operating at low speed. The next generation of facsimile terminals (Facsimile Group 4) will, however, have better resolution and require the support of a data network capable of transmitting at higher speeds [6].

This paper has the main aim of surveying the network oriented telematic services primarily used for interpersonal communication, by giving an overview of their main features and of the state of standardization development. In chapter 2, the layered architecture of the OSI model is introduced, as it provides the framework for the study of the different telematic services, being followed by an overview of the main activities on standards development. In chapters 3 to 7, Electronic mail, Document interchange, the Directory, Group communication (including Computer conferencing) and Videotex are respectively explained. Finally, chapter 8 presents the main concepts and state of evolution of the emerging Integrated Services Digital Network, which will integrate data, voice and image communication in the same network, and in chapter 9 some conclusions will be drawn.
2. COMPUTER NETWORKS AND THE OSI MODEL

2.1 - The OSI concept

The Open Systems Interconnection (OSI) concept is fundamental for the description of the telematic services. The aim of OSI is to provide communication-based user services that operate between computer systems, which may be located in different countries and be supplied by different manufacturers [7] [8].

The communication between two computers, called end systems in the OSI nomenclature, may be modelled as shown in Fig. 2. The final goal is to provide communication between two user’s application processes running on end systems A and B, through a data communication network. This data communication network can be a public or private WAN; it can also be a LAN when both computers operate in a local environment.

Due to the fact that different computers frequently have distinct operating systems and different forms of data representation, the communication between the application processes in the two end systems needs to be standardized. Special purpose hardware and software is also required in both systems to handle the requirements of establishing a communication channel across the network and of having flow and error control in the channel during the communication.

The International Standards Organization (ISO) has introduced a reference model for OSI (OSI Basic Reference Model) that provides a basis for the development of standards. In this
model, every end system is structured into seven layers each of which performs a well defined function. Peer layers in two distinct systems communicate through a communication protocol.

The basic functions of each layer are:

Application layer (7) - it provides access to the OSI environment for user application processes running on the end system;

Presentation layer (6) - it provides a common representation of the application information for the communication between the two systems;

Session layer (5) - it manages the dialogue between the two end systems during the communication;

Transport layer (4) - it provides the session layer with a reliable message transfer facility, independently of the network type;

Network layer (3) - it establishes and clears a network connection between the two systems, including the network routing facility;

Data link layer (2) - it provides the network layer with a reliable information transfer facility, including error control and flow control;

Physical layer (1) - it provides the data link layer with a means of transmitting a bit-stream between the two systems. It is concerned with the physical and electrical interface between the end system and the network termination.

The lower three layers constitute the network environment. They provide the so-called bearer services, which are application-independent and are only concerned with the provision of a data communication mechanism, independent of the type of network used for the exchange of information. The OSI environment consists of all the seven layers; it includes the network environment and the additional four upper layers that allow the two end systems to communicate at the application level, providing the telematic services.

Each layer has an interface between itself and the adjacent layers. The implementation of a layer functionality is independent of all the other layers, which permits changes to be made in one layer without affecting the others. A layer operates according to a certain communication protocol by exchanging protocol data units (PDU), consisting of user data and additional control information, with a peer layer in the other system. Within an end system, each layer provides a set of services to the layer immediately above it and, in turn, uses the services from the layer immediately below it to carry the protocol data units. Therefore, although conceptually each layer communicates with a peer layer in the other system according to a certain protocol, in practice the protocol data units of the layer are passed by means of the services provided by the next lower layer.

1) In this case, the term service is used in the context of the OSI model. It is different from the concept of service used in the context of network operation, as are the cases of bearer and telematic services.
2.2 - The development of standards

The OSI model is a basis for the development of standards. Standards are defined for each layer and each standard is described in two documents: service definition document and protocol specification document.

The service definition document contains a specification of the OSI services provided by the layer to the layer above it. The protocol specification document contains a precise definition of the protocol used for communication between two layers and also the specification of the OSI services used by the layer to implement the protocol.

A number of standards is associated with each layer. They are sometimes defined by different bodies and may offer different levels of functionality. They are called the base standards.

For a certain OSI environment a selected set of standards has to be defined for use by all systems in that environment to allow an open communication to be established among all the systems. This is called a functional standard or profile. A functional standard states exactly which base standard is to be used in each layer in that environment, explains the less clear points of the base standards to which it refers and establishes the values of the options existing in the base standards, making them ready for implementation [9].

The most important international bodies producing standards for computer networks are the International Standards Organisation (ISO), the Consultative Committee of the International Telegraph and Telephone (CCITT) and the Institute of Electrical and Electronics Engineers (IEEE). Typically, the ISO and IEEE are interested in producing standards for use by computer manufacturers, whereas the CCITT is interested in producing standards, called “recommendations” in the CCITT terminology, for connecting equipment to the different PTT networks.

There are other organizations that are mainly active in defining functional standards. Some of the most well-known organizations working in this field are, the Comité Européen de Normalisation (CEN), the Comité Européen de Normalisation pour Electrotechnique (CENELEC), the Comité Européen de Postes et Telecommunications (CEPT) and the Standards Promotion and Application Group (SPAG) in Europe, the Corporation for Open Systems (COS) in the United States, and the OSI Promoting Conference (POSI) in Japan.

As an example, Fig. 3 shows a selection of CCITT recommendations (base standards) to be used for Electronic mail. At layer 7, the X. 400 series of recommendations (X. 400, X. 401, X. 410, X. 411, X. 420, X. 430) specify Electronic mail systems at the application level. At layer 6, recommendations X. 408 and X. 409 respectively specify the encoding information for the Electronic mail systems and the communication protocol at the presentation level. At layers 5 and 4, X. 215 and X. 214 define the services provided by the session and transport layer respectively, whereas X. 225 and X. 224 specify the session and the transport protocols respectively.

In this example, the recommendations indicated for the network environment, assume that an X. 25 packet data network is used for the communication. X. 213 specifies the services provided to layer 4, and the other three recommendations, X. 25 (layer 3), X. 25 (layer 2) and X.21 specify the X. 25 protocol.
3. ELECTRONIC MAIL

3.1 - General aspects

An Electronic mail system is a software package that, installed on a computer, provides facilities for the users of this computer to generate, send, read and file messages. In a general sense Electronic mail allows persons to exchange electronically all the information that they would otherwise exchange by conventional mail.

Several Electronic mail systems can be interconnected via an Electronic mail network which provides means to transfer the messages. An Electronic mail network has a set of rules that specify how the messages are to be transferred; these rules are the Electronic mail protocols.

A number of different Electronic mail systems have been in existence for some time. Examples are the VMS Mail over Decnet, RFC 822 over EARN and the Grey Book implementation over JANET. However, all these systems are incompatible, and gateways are needed for the different systems to communicate between themselves [10] [11].
The X. 400 series of recommendations launched by the CCITT in 1984, specify a complete application service based on the OSI model and opens the way to a uniform Electronic mail service to be provided [12] [13]. The CCITT will issue a new series of recommendations in 1988, which adds some extensions to the 1984 recommendations.

The first X. 400 interconnection was established between the University of British Columbia in Canada and KDD in Japan, in 1985. Since then, many manufacturers started developing X. 400 products and a number of PTTs committed already themselves to offer the service soon. As most of the X. 400 implementations appearing in the near future will still be made according to the 1984 recommendations, the following X. 400 description is primarily based on them.

3.2 - X. 400 functional model

Electronic mail systems are known as Message Handling Systems (MHS) in the X. 400 nomenclature. The X. 400 functional model for MHS is shown in Fig. 4 and embodies two levels of service. At the lowest level, the Message Transfer System (MTS) operates as a general purpose carrier of messages across the network. At the higher level, the MHS uses the MTS as the underlying carrier. It provides the users with facilities to use the MTS and also assists them in constructing and interpreting messages.

In X. 400, a user is referred as an originator when it is sending a message, and as a recipient when it is receiving a message. A message can be sent to more than one recipient. An MHS is a store-and-forward system, and to send a message from the originator to the recipient(s) there are five phases involved: message preparation, submission interaction, relaying interaction, delivery interaction and message reading.

In the first phase, the user prepares the message in an agreed structure and syntax with the support of the User Agent (UA). A UA is a set of processes in a computer that are used to create,
inspect and manage the storage of messages; there is one user per UA. In the second phase, the message is then passed to the attached Message Transfer Agent (MTA). An MTA is also a set of processes that are used to carry the message along the network. In the third phase, the linked set of MTAs operate together, according to a defined protocol, to convey the submitted message to the specified recipient MTA. The latter then delivers the message to the recipient UA, in the fourth phase. The message is typically stored in the recipient mailbox until the recipient requests to read it, to complete the operation. If a message cannot be delivered, the originating UA is notified.

The structure of the message exchanged in the MTS is well-defined. It is composed of an envelope and a content, as shown symbolically in Fig. 5.

![Fig. 5 - Basic message structure](image)

The envelope contains the address information used for transferring the message within the MTS. The content is the actual message to be delivered to the recipient.

### 3.3 - Interpersonal Messaging System

According to the type of message content that they can handle, the UAs are grouped in classes, and those that belong to the same class are called "Cooperating UAs".

To satisfy the need for communicating messages from person to person, the CCITT defined in X. 400, a set of rules for a class of cooperating UAs, the Interpersonal Messaging System (IPMS). This is the only class of cooperating UAs defined until now, but others may be defined in the future, for instance, for electronic funds transfer, library services and office applications.

The IPMS comprises the MTS and a specific class of UAs, as shown in Fig. 6; in addition, to make the IPMS accessible also to users of other services, access protocols have been defined for teletex terminals (X. 430) and the access to telex and other telematic services is under study. The UAs represented in the figure with an asterisk, are examples of UAs that use the MTS, but do not support the IPMS.

The content of an interpersonal message is structured into a heading and a body as shown in Fig. 7. The heading contains the IPMS indicators and the body the actual information. In
addition, the body can be composed of a sequence of body parts, each encoded according to any one of a certain set of encoded information types, such as text, voice, facsimile and graphics. The type of the encoded information of each body part is conveyed along with the body part itself.

Fig. 6 - The Interpersonal Messaging System model

Fig. 7 - The structure of an interpersonal message

3.4 - Physical and organizational mapping

In general, UA implementations will provide storage in which users can manage incoming and outcoming messages. While processing messages, the user interacts with the UA via on I/O device. A UA can be physically implemented as a set of application processes in a multi-access computer system or in a personal computer.
A UA and an MTA may be implemented in the same system (co-resident) or in physically independent systems. Both cases are documented in the two systems represented in Fig. 8 a. On the right hand side of the figure there is no standard protocol for the interaction between the UA and the co-resident MTA. On the left hand side, the communication between the UA and the MTA is done via a standard protocol, called P3. A more complex configuration is given as an example in Fig. 8 b.

![Diagram](image)

**Fig. 8 - Physical mapping of the MHS**

The organizational mapping of the MHS is centered around the concept of Management Domain. A Management Domain is a collection of at least one MTA and zero or more UAs owned by a PTT Administration or an organization. In the first case, we have an Administration Management Domain (ADMD) and in the second one we have a Private Management Domain (PRMD). The interconnection between Administration and Private Management Domains is shown in Fig. 9.

73
COUNTRY A | COUNTRY B

Fig. 9 - Administration and Private Management Domains
A PTT Administration may provide access for its subscribers to the ADMD at one or more of the following boundaries: i) user I/O device to Administration supplied UA, ii) private UA to Administration MTA, iii) private MTA to Administration MTA. An important role of the ADMD is to act as naming authority for all the organizations which are within its region of authority. The ADMD is concerned only with administering the top level of names; the responsibility for the naming in the PRMDs is normally allocated to the organization itself.

3.5 - Protocols

The message handling protocols are located at the level of the application layer in the OSI model. This layer, in the case of MHS is considered to be divided into two sublayers with different message handling functions. The higher sublayer is the User Agent sublayer (UAL), which contains the UA functionality associated with the contents of messages. The lower layer is the Message Transfer sublayer (MTL), which contains the MTA functionality and provides the message transfer service. The division into sub-layers and the MHS communication protocols are depicted in Fig. 10.

There are three distinct MHS protocols: P1, P2 and P3. The P1 protocol basically defines the operations undertaken by the MTAs in relaying the messages along the MTS (X. 411). The P1 units of exchange are called Message Protocol Data Units (MPDU). There are two types of MPDU: the User MPDU, which carries messages submitted by a UA for transfer and delivery and the Service MPDU, which carries information about the messages that are exchanged between MTAs.

The P2 protocol defines the syntax and semantics of the interpersonal messages content being transferred (X. 420). The units exchanged in P2 are called User Agent Protocol Data Units (UAPDU), and they comprise the contents of the messages exchanged between UAs.

The P3 protocol enables a UA that is remote from its MTA to obtain access to the services of the MTL, through a special entity called Submission and Delivery Entity (SDE), which is co-resident with the UA (X. 411). The access includes the transfer of messages from the SDE to the MTA during submission and from the MTA to the SDE during delivery. Operational Protocol Data Units (OPDU) are the units of exchange between an SDE and an MTA. They contain the information needed to require an operation or report the result of that requirement.

![Fig. 10 - MHS application layer model](image-url)
3.6 - Naming and addressing

When a user submits a message, the MTS must be informed of the identity of the recipients. This identification is done by a name, called Originator/Recipient (O/R) name.

An O/R name has two components, at least one of which must be present: i) O/R address and ii) directory name.

The O/R address contains information that an MTA can use for routing a message to its destination. The directory name is intended to be a more user-friendly and more stable form of name than the O/R address, as it will be independent of the physical configuration of the MHS.

If a user originates a message addressed to an O/R name which consists only of a directory name, then the MTS has to consult the Directory to discover the corresponding O/R address. However, if the originator supplies an O/R address in the O/R name, the MTS will use it directly to route the message to the recipient. For the time being, as the Directory recommendations have not yet been issued, every O/R name consists only of an O/R address.

An O/R address is structured as an ordered list of attributes, each of which consists of a type and a value. A typical set of attribute types that form an O/R address is the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country name (C)</td>
<td>mandatory</td>
</tr>
<tr>
<td>Administration domain name (A)</td>
<td>optional (at least one attribute)</td>
</tr>
<tr>
<td>Private domain name (P)</td>
<td></td>
</tr>
<tr>
<td>Organization name (O)</td>
<td></td>
</tr>
<tr>
<td>Organizational unit name (OU)</td>
<td>must be selected</td>
</tr>
<tr>
<td>Personal name (S)</td>
<td></td>
</tr>
</tbody>
</table>

For example, the following is my complete X.400 O/R address:

C = pt; A = ctt; P = utl; O = ist; OU = deec; S = ajc

meaning that the user ‘ajc’, belongs to the Department ‘deec’, in the Faculty ‘ist’, in the University ‘utl’, in the region of authority of the Administration named ‘ctt’, situated in the Country ‘pt’ (Portugal).

3.7 - X.400 implementations

One of the first implementations of X.400 was the EAN (Electronic Access Network) developed at the University of British Columbia, Canada [14]. It is the mail system used at CDNet, the Canadian academic network. There is also a Europe-wide EAN network for the academic and research community, which is connected to EAN networks existing in other parts of the world. There are gateways between EAN and older mail systems. For this purpose a set of gateways at CERN allow the flow of mail between EAN and EARN, EUNET and DECnet [15]. A gateway between EAN and JANET exists in London. Some of these networks have already plans to migrate their mail systems into X. 400 in the near future, therefore eliminating the need of gateways.
Other known implementations of X. 400 are the GIPSI [16], developed at INRIA, France, in 1985, running on Bull machines and the KOMEX [17], developed at GMD, Germany, running on Siemens machines.

A large number of computer manufacturers have already announced X. 400 products to run on their machines, and some Administrations started already offering an X. 400 public service, such as the recently announced ATLAS 400 [18] in France.

4. DOCUMENT INTERCHANGE

4.1 - General aspects

The interchange of documents by means of data communication plays an important role in office automation. Documents are items such as memoranda, letters and reports, and they may include text, tables and graphics. Documents which are interchanged in electronic form should not only be printable at the receiver's site, but it should also be possible to edit and reformat the received documents in the same way as at the sender's site.

In order to facilitate such an interchange a document architecture needs to be defined. The document architecture should allow different types of contents to exist in the same document and also permit that the intentions of the document originator with respect to editing, formatting and presentation can be communicated to the receiver.

For this purpose a model has been developed which describes how documents are structured. This model is called Open Document Architecture (ODA) and is being standardized in a combined effort by CCITT (series of recom. T. 410) and ISO (DIS 8613) [19] [20]. Associated to ODA, it is also defined the Open Document Interchange Format (ODIF), which indicates the format of the data stream used to transmit ODA documents.

4.2 - Open Document Architecture

ODA describes a document in terms of a document profile and a document body.

The document profile is used for the document management. It specifies the characteristics of the document that apply to it as a whole. The profile includes the title of the document, the date, the name of the author and an indication of the main architecture features that are used in the document, such as the specification of the character sets, character fonts and types of contents.

The document body consists of the logical structure and the layout structure. The structure of a document consists of the division of the contents of the document into a number of parts, called objects. The logical structure associates the contents of the document with the document's logical elements, such as, chapters, sections, paragraphs and figures. The layout structure associates the contents of the document with elements related to the presentation media, such as pages, columns and blocks.
The logical and layout structures provide alternative but complementary views of the same document. Both these structures are shown applied to a simple document, a business letter, in Figs. 11 and 12 respectively.

In the logical structure of the business letter, the header and the main part of the letter can be distinguished at a first level. The header is considered to be subdivided in the date, subject, addressee and summary of the letter. The main part consists of a set of paragraphs and probably figures, and at the end there will be a formula for greetings and the signature.

The layout structure of the same letter would consist of the front page which would be divided into blocks, where the items of the header would be written, and of a number of other pages, each of them divided into blocks, where the text and figures would be inserted.

The logical structure and the layout structures of a document are defined in different ways. The logical structure is determined by the author during the editing process and the layout structure is determined by a formatting process, which is controlled by some directives associated with the logical structure, such as the requirement for a chapter to start on a new page, the requirement to underline the section title or the indication of the indentation for the start of paragraphs.

Depending on the application, either one or both structures can be transferred. If only the logical structure is transferred, although the contents of the document can be subsequently altered, its layout will be determined by the recipient’s system. If only the layout structure is transferred, the layout of the document will be as intended by the originator, but the contents will be in image form, which inhibits further logical processing at the recipient’s system.

An MHS can be used as the support for the interchange of ODA documents. An ODA document, encoded as indicated in O DIF, may constitute a body part in an interpersonal message in X. 400, and then be transmitted by using the X. 400 protocols.

![Diagram of logical structure of a business letter]

**Fig. 11 - Example of the logical structure of a document**
4.3 - ODA implementations

The first ODA products are expected within the next couple of years. In the meantime some pre-competitive implementations of ODA have been tried. One of the most relevant occurred in the ESPRIT PODA (Pilot ODA) project [21]. The partners of this project designed a generalised ODA software architecture for their systems and demonstrated the interchangeability of ODA documents among the different proprietary systems in the CeBIT 87 exhibition, in Hannover. In the continuation of this project a demonstration for the interchange of ODA documents containing text and graphics using X. 400 is foreseen.

5. THE DIRECTORY

5.1 - General aspects

The Directory is a collection of cooperating open systems that hold a database of information about a set of objects in the real world.

The main need for a Directory service arises from the desire to isolate the user from changes in the network, which implies the designation of a recipient by symbolic name (e.g. the O/R directory name in X. 400), rather than address; the utilization of symbolic names gives also to the user a more friendly view of the network.

There are three main Directory functions:

i) Name to attribute binding - it binds a name to an attribute related to the object referred by the name, as it is the case of name to address binding (white pages directory).

ii) Name to list of names binding - it binds a name to a list of names, which permits to designate a group of recipients by a single common name, as it happens in the distribution lists of Electronic mail.
iii) Attribute to set of names binding - it lists the names of objects which have a given attribute (yellow pages directory).

The Directory service is in the process of standardization; many of its characteristics are already established, and it will be defined in the X. 500 series of recommendations of the CCITT [22].

5.2 - Functional model

X. 500 adopts the approach of having a distributed Directory model; this model has the advantage that each address resolution request can, many times, be carried out locally, thereby increasing the speed of response to requests and reducing the amount of network traffic. The Directory functional model is depicted in Fig. 13.

The component blocks of the Directory are the Directory User Agent (DUA), the Directory System Agent (DSA) and the Directory Information Base (DIB). The DUA is an application process that accesses the Directory and interacts with it to obtain the service on behalf of a user; each DUA serves a user. The DSA is an application process whose role is to provide access to the collection of information to DUAs and/or other DSAs. The DIB is a data base which holds the collection of information to which the Directory provides access. As we are in presence of a distributed Directory, each DSA holds a local data base ('DIB'), which is part of the DIB.

All services are provided by the Directory in response to requests from DUAs. The DUA interacts with the Directory by communicating with one or more DSAs, which may use information stored in its local database, or alternatively it can interact with other DSAs to carry

![Diagram of Directory functional model]

**Fig. 13** - The Directory functional model
out the requests. The requests may be for Directory interrogation, such as read, compare and search operations, and for Directory modification, such as the operations of adding, removing and modifying entries in the DIB. The Directory must always report the result of each request that is made to it.

5.3 - Protocols

The Directory communication protocols are required to provide cooperation between the DUAs and the DSAs, and between the DSAs themselves; these protocols are located in the application layer of the OSI model, and they use the underlying layers to establish reliable connections between the individual systems independently of the network type.

There are two Directory communication protocols: Directory Access Protocol (DAP) and Directory System Protocol (DSP), as represented in Fig. 14.

The DAP defines the exchange of requests and responses between a DUA and a DSA. It contains protocol elements associated with interrogating and modifying the Directory. The DSP defines the exchange of requests and responses between cooperating DSAs.

![Fig. 14 - The Directory protocols](image)

5.4 - MHS and Directory interworking

The Directory service has a key role in the support of MHS. The way through which both systems will interwork, is shown in the functional diagram of Fig. 15.

Both UAs and MTAs may use the Directory. For example, one UA may submit the directory name of a recipient to the Directory and obtain from it the recipient's O/R address. The UA may then supply both the directory name and O/R address to the MTS. Another possibility is that a UA submits a directory name directly to the MTS. In this case the MTS itself asks the Directory for the recipient's O/R address and adds it to the envelope.

Each UA or MTA accesses the Directory via a DUA, which is, in many cases, physically co-resident with the corresponding UA or MTA. In any case, the communication protocol between a DUA and its user is not standardized, being dependent on the implementation.
5.5 - Directory implementations

The standardization work on the Directory is still in progress and therefore, there is a small number of Directory implementations. The most relevant one is being done in the ESPRIT THORN (THe Obviously Required Nameserver) project [23]. This project was launched in 1985, and it had as an overall objective to build a Directory service in line with the emerging standards, to be adequate for the Information Exchange Service used by the ESPRIT project teams. The work is in progress and a migration of THORN to the X. 500 recommendations is under way.

6. GROUP COMMUNICATION

Group communication is concerned with the provision of facilities to support the communication needs among groups of people. Two main systems for group communication may be presently considered: i) Distribution Lists in MHS and ii) Computer conferencing. Both these systems are explained in the following sections.

6.1 - Distribution Lists

X. 400 is mainly concerned with the Electronic mail among individuals, i.e., an individual user sends a message to one or more individual users by explicitly specifying all the recipients of the message. However, the 1988 version of X.400 has already a limited capability for group communication, which is the Distribution List (DL) concept [24].

DLs allow an originator to transmit a message to a group of recipients by using the O/R name of the group instead of having to enumerate each of the final recipients. More formally, a DL is a set of elements, called the DL members, each of which is an MHS user, a collection of such users,
or another DL. In the latter case we are in presence of a nested DL.

The use of DLs requires the existence of a Directory, which will store the identification of the members of the DL, the names of the users that have permission to submit a DL name and the identification of the DL owner.

When an originator addresses a message to a DL, the MHS with the aid of the Directory must expand the DL, i.e., replace its name with the names of the users referred in the DL. If the DL contains nested DLs, the expansion may be performed incrementally, for example, each of the MTAs involved in conveying the message may carry out only part of the expansion. To avoid the possible looping of messages in the case of nested DLs an expansion history field must be added to the envelope in the P1 protocol, tracing the various lists that were expanded in the distribution process. By inspecting this field, an MTA can check whether the expansion of the DL has already been performed and in the affirmative case, abandons the expansion.

An example of a DL expansion is given in Fig. 16. A DL O/R address specifies the MTA at which expansion occurs. A message that contains a DL recipient name is carried to the expansion point, where the set of member names of the DL are added to the list of recipients of the message. In the case of a member of the DL being itself another DL with a distinct O/R address, the message is routed to the next expansion point.

6.2 - Computer conferencing

The only use of Electronic mail with DLs is a rudimentary capability for Group communication. An efficient Group communication system would require besides that facility, other ones that would allow an organized presentation of the messages to the recipients, control the access of the different users to the information and give the possibility of forming special interest groups. MHS which support all these facilities as an addition to Electronic mail are called Computer conferencing systems [25].

The basic properties of a Computer conferencing system are then the following:

i) the messages are organized into a number of distinct communicating groups, called “Computer Conferences” or “Bulletin Boards”, and the originator of the message needs only to indicate the name of the Bulletin Board to make the message available to all the members of that Bulletin Board;

ii) the messages after being read must be stored by the system for some time, in order that new members of the Bulletin Board can read them;

iii) there must be the possibility of controlling the access to the information, therefore two classes of Bulletin Boards exist, the open Bulletin Board where new members can join without restrictions and the closed Bulletin Board, where participation is restrained to a selected group of people;

iv) users can establish interpersonal communication, through the Electronic mail facility, whenever they wish.

Besides these basic properties, Computer conferencing systems may include some other ad-
ditional features. Two examples of these extra features are the existence of a moderator to control the information displayed in the Bulletin Boards, and the occurrence of information retrieval facilities for the acquisition of stored messages according to certain search criteria, such as the author, number or date.

Although no standard exists yet for Computer conferencing, there are a few systems already working such as QZCOM, EUROKOM, FORUM and USENET NEWS. However, most of them are centralized, requiring that all the users log in a central computer, where all the conferences are organized. An example of a Computer conferencing system working along these lines is the

![Diagram](image)

1. Submission
2. Delivery after first expansion
3. Relaying
4. Delivery after second expansion

Fig. 16 - Example of DL expansion
EUROKOM. This system is located at the University of Dublin, Ireland, and was established by the EEC as an aid for the communications requirements of participants in the European research programs. It has all the basic facilities of Computer conferencing, and this can be illustrated by the indication of some of the EUROKOM user commands:

- LIST CONFERENCES ; it indicates all the conferences available
- MEMBER <conference> ; to become a member of an open conference
- JOIN <conference> ; to join a conference of which one is a member
- NOTICE ; to send a message to the current conference
- NEXT NOTICE ; to read the next unread notice in the current conference
- REVIEW ; to review entries in the conference
- WITHDRAW <conference> ; to finish being a member of a conference.

A few other Computer conferencing systems may have some support for distributed operations, but with only very limited facilities, like the USENET NEWS. The definition of a fully distributed Computer conferencing system is still at a research stage, and the possibility of extending the X. 400 and X. 500 protocols for that purpose, instead of developing a completely new protocol, is being investigated, namely in the framework of an ESPRIT project [26].

7. VIDEOTEX

7.1 - The structure and use of videotex

The term videotex may refer to two distinct systems, known as interactive videotex and broadcast videotex, respectively [27].

Interactive videotex is a bidirectional system in which the users can access a remotely located computer from their premises, and display the information retrieved from the computer on a specially adapted visual display unit. The flow of information between the users and the computer occurs in the Public Switched Telephone Network (PSTN) and/or Public Switched Data Network (PSDN).

Broadcast videotex is a broadcasting system which displays selected frames of information as they are being continuously recycled by the originator of the information. The information is prepared and stored digitally and is many times broadcast as part of the regular TV signal. Broadcast videotex is usually known as Teletext.

In this paper, only interactive videotex will be studied, as it is the most relevant system for interpersonal communication and also the one in more widespread use. In the following text, only the single term videotex will be used, meaning always interactive videotex.

The general structure of a videotex system is represented in Fig. 17. The communication infrastructure for the flow of information between the user terminal and the host computers containing data bases is normally the PSDN, but the access to this network can be done through
the PSTN, to guarantee a wide availability of the system. The videotex access points are special switching nodes that act as an interface between the two networks.

The use of a videotex system is better illustrated by a concrete example. The French videotex system, called Teletel, is chosen as the example, as it is probably the most well-known and one of the most widely used videotex systems in the world [28].

Every user has at his premises a special videotex terminal, called Minitel, connected to the PSTN. There is a range of Minitel terminals with different facilities, but every Minitel integrates as a minimum, a keyboard, a CRT display screen and a modem.

When a user wishes to access the Teletel, dials the access code in the telephone set and is then connected through the PSTN to the nearest videotex access point. The line is then transferred from the telephone to the Minitel and the video access point sends an initial menu to the Minitel display. The user can then indicate the wanted Teletel service by typing its name at the Minitel keyboard and access the respective host computer data bases, through Transpac, the French PSDN.

There is a large number of services available in Teletel, and this is one of the reasons for its success. Examples of such services are news reports, classified ads, income tax calculation, comparison of prices in different supermarkets, video games, airline and railway seat reservations and an electronic directory service containing information on the telephone subscribers.

Many other videotex systems are already in use, in a number of countries, however, they are often not compatible, as there are different representations for the text and graphics that can be followed. Examples of relevant videotex systems are Teletel in France, BTX in Fed. Rep. of Germany, Prestel in United Kingdom, Telidon in Canada and Captain in Japan.

7.2 - The coding of text and graphics displays

Three different options are usually considered for the representation of text and graphics in videotex systems: alphamosaic, alphageometric and alphaphotographic.

These three options are described in the CCITT recommendation T. 101 [29]. This recommendation is mainly concerned with the description of the Presentation PDUs and data syntaxes.
used for their coding. Three distinct data syntaxes are indicated in T. 101, which respectively define the alphamosaic, alphageometric and alphaphotografic systems. Alphamosaic is the most rudimentary solution in terms of screen image definition and alphaphotografic the most advanced; in Europe only alphamosaic systems are available as a public service, presently. Recommendation T. 101 also indicates that videotex will use the X. 215/X. 225 in the session layer and X. 214/X. 224 in the transport layer. An application layer protocol for user to data base access is not published yet.

7.2.1 - Alphamosaic systems

In the alphamosaic system the display frame is composed of defined character positions, which may be occupied by any of the characters of the repertoire. The default format of the frame is 24 rows of 40 columns.

The repertoire is composed of the alphanumeric repertoire and mosaic repertoire, in which the mosaic repertoire is formed by dividing the character space into a matrix of 2x3 elements. There is also a set of control characters.

Alphanumeric, mosaic and control characters are represented in different code tables in the terminal. A code table consists of 128 positions arranged in 8 columns and 16 rows. A code table entry is identified by a 7-bit code, in which the 3 msb define the column number and the 4 lsb define the row number.

If we consider, for example, the PLDS (Presentation Layer Data Syntax), which is a CEPT functional standard for the European alphamosaic systems, there are two alphanumeric tables (G0, G2) which include graphics symbols, three mosaic tables (L, G1, G3) and three control tables (C0, C1 series, C1 parallel) altogether. G0 and C0 are the default tables, and the evolution to the use of other tables is done through the occurrence of special control characters. The contents of four of these tables (G0, G2, C0, L) is shown in Fig. 18, as an example. Notice that the use of columns 0 and 1 is reserved for control characters.

The serial and parallel alphamosaic coding forms differ because of the way attributes are handled. Examples of attributes are the colour of a character, the colour of the screen background or the underlining of a sentence. In a terminal using a serial alphamosaic coding, the attribute codes are stored always with and precede in the same memory the characters to be displayed; attributes correspond to a position in the screen and make the cursor move. It is the case of the Prestel system.

In a terminal using a parallel alphamosaic coding, the characters comprising the display and their attributes are stored separately in different parts of the terminal memory: extra memory is required, but the attribute does not interfere with the display of the characters. It is the case of the Teletel system.
7.2.2 - Alphageometric systems

In an alphageometric system the display is composed of alphanumeric text and pictorial drawings defined in terms of geometric primitives transmitted to the terminal as drawing commands. This option for text and graphics representation has been adopted in North America and named NAPLPS (North American Presentation Level Protocol Syntax).

NAPLPS has five geometric primitives to draw a point, a line, an arc, a rectangle and a polygon, and a sixth one to draw a line or a polygon in an incremental way. A geometric primitive is composed of an opcode and zero or more parameters, which specify the coordinates needed by the primitive. The opcode is a 1 byte character that identifies the primitive or alternatively expresses a control operation. A repertoire of alphanumeric symbols is also available.

Fig. 18 - Examples of PLDS code tables
7.2.3 - **Alphaphotographic systems**

An alphaphotographic system is the videotex system that offers the best image resolution. An image is displayed as the result of the transmission of the individual picture elements (pixels).

This videotex system permits the storage and the display of photographs and other high-resolution images. The present limitations for its implementation result from the requirements for a large store in the terminal and high speed communication links. A pilot alphaphotographic videotex system, the CAPTAIN system, is being experimented in Japan.

8. **INTEGRATED SERVICES DIGITAL NETWORK**

8.1 - **The ISDN concept**

The Integrated Services Digital Network (ISDN) is a completely digitized communication network in which the same switches and paths are used to establish connection for the different services. As all the types of information can be digitized and transmitted in this form, the ISDN integrates voice, data, text and image communication in the same network [30]. The ISDN is being implemented by the progressive digitization of the public telephone network.

The physical configuration of the ISDN is shown in Fig. 19, consisting of: i) local exchanges, ii) transit exchanges with routing functions and iii) digital communication links.

The users may have simultaneous access to more than one service, therefore, multifunctional terminals will be typically used; the user access to the network will be done through a standard interface. Service providers, like for example videotex host computers, may connect directly to the network and a set of gateways for the ISDN to interwork with other types of existing networks have to be provided.

The CCITT I-Series of recommendations, published in 1984, define the general configuration of the ISDN, services, user-network interface structures, communication protocols and interworking with data networks [31]. Some pilot implementations have been done and a limited public ISDN service is starting to be offered in some countries. Although, due to its capabilities the ISDN can substitute both telephone and data networks, it will happen that at least in the near future the three types of network will coexist, because of network evolution strategies, rendering necessary the existence of gateways.

8.2 - **User-network interfaces and protocols**

The CCITT I-recommendations specify the types of user - network interfaces that may be used in an ISDN environment. There are two main interface structures specified: basic structure and primary structure.

In the basic structure, there are 2 B-channels plus 1 D-channel simultaneously available to the user. A B-channel is a 64 kbit/s, bidirectional, information carrying channel. A B-channel gives
to the user a transparent connection to the network, and it can be used for either circuit-switching or packet-switching at the user criterion. The D-channel is a 16 kbit/s, bidirectional channel, mainly intended for carrying signalling information associated with the B-channels. It always operates in a packet-switched mode, and as the signalling rate is usually less than 16 kbit/s, the channel can be multiplexed to be used as a third information carrying channel available to the user, although at a rate lower than the B-channels.

In the primary structure, there are 30 B-channels (23 in North America) plus 1 D-channel. The primary structure is typically utilized by users that have PABXs at their premises and therefore need to have simultaneously available a large number of channels. Each B-channel has the same characteristics as in the basic structure, but the D-channel, in this case, operates at 64 kbit/s.

As the ISDN is only concerned with the provision of communication facilities, it only covers the first three layers of the OSI model, respectively the physical, data link and network layers. Assuming the basic interface structure as an example, the use of the different ISDN protocols is indicated in the lower three layers of Fig. 20.

The I-recommendations define the communication protocol to be used at the physical layer (I. 430, I. 431), the data link protocol in the D-channel (I. 440, I. 441) and the network layer protocol for signalling in the D-channel (I. 450, I. 451). The users are free to choose the layer 2 and 3 protocols for the B-channels, and if the D-channel is also used to carry the user information, X. 25 must be used at layer 3 in that case. The data from the distinct B-channels and D-channel are multiplexed at layer 1 into a common frame for transmission to the local exchange.
The teleservices, e.g. Electronic mail, Directory, Videotex, Telephony, Fax, Teletex, Videotelephony, use all the seven layers of the model, as documented in the same figure. End-to-end signalling, i.e., user-to-user signalling is considered a special application that utilizes the ISDN signalling protocol at layer 3.

8.3 - ISDN evolution

The introduction of a public ISDN will accomplish a number of strategic objectives, bringing advantages to the users and to the network operators. The user will have a large number of telecommunication services available through a standard interface, at a speed higher than the one generally available in present public data networks. The user may also have some economic advantages, if as a result of factors of economy of scale, the ISDN tariffs reduce the cost of the services provided. A better quality of service is also expected, due to the digitization of the network, and a better planning for the network evolution can be achieved.

![Protocol structure at the user-network interface](Fig. 20)
Despite these advantages, ISDN will only be available in a widespread scale in a few years from now, due to the high investments that are necessary to make and to the need of harmonizing its introduction with the existing telephone and data networks. The success of its introduction will be also very dependent on the tariffs charged to the user and on the definition of ISDN functional standards that may guarantee the compatibility of the ISDN implementations in the different countries.

At present, ISDN starts to be available in some European countries, such as France and Germany and in the United States. It is foreseen that many other European countries, Canada and Japan will also start offering ISDN services in the next five years, although at a limited scale.

The evolution of ISDN towards higher speeds is at a research stage, presently. Work is being carried out, in order that speeds in the order of hundreds of Mbit/s can be achieved in a public network with integration of services, taking advantage of the high transmission speeds allowed in optical fibers, which will be more and more used as the transmission infrastructure in communication networks. These high speeds would allow the integration of all the services available in the ISDN and services that require higher speeds, such as, high definition TV distribution, video communication and high-speed file transfer, in the same network, called Broadband ISDN. Special switching and transmission techniques, network structures and new services definition for a Broadband ISDN are being investigated in Europe (RACE project), United States and Japan, being possible that the first pilot networks appear up to 1995 [32].

9. CONCLUSIONS

Interpersonal communications using computers is a subject of prime importance for the scientific community, due to the large international collaborations that are set up in many scientific fields, namely in High Energy Physics.

The scientific community is normally a forerunner in the use of new telecommunication facilities and services. This has been exemplified in the latest years through the use of many private academic and research networks, such as EARN, EUNET, DECnet, JANET, EUROPKOM and others, which incorporate network-oriented telematic services, not yet available in the public network [33]. Due to the lack of OSI standards, all these networks have their own proprietary protocols. In order to ensure compatibility among the networks, eliminating the need of gateways, there is now an effort to achieve the migration of the protocols existing in the different academic and research networks towards OSI protocols [34] [35]. This effort is being coordinated by the RARE (Réseaux Associés pour la Recherche Européenne) Association [36], which was founded in 1986 and groups more than twenty European countries and international organizations, including CERN.

The use of public telecommunication facilities, whenever available, is also encouraged by RARE, and this is well exemplified in the EAN network, which is running an X. 400 prototype implementation on the public X. 25 network, since 1986. It may be foreseen that, if tariffs are reasonable, the use of the public telecommunication networks by the scientific community may
significantly increase in the future, accompanying the gradual introduction of the different network oriented telematic services based on OSI protocols and of ISDN.

Acknowledgements

The author is grateful to his colleagues Drs. P. Veiga and V. Vargas, for their valuable comments in the production of this text.

REFERENCES


BIBLIOGRAPHY

[38] F. Halsall, Data Communications, Computer Networks and OSI, Addison Wesley, 1988.
ADVANCED COMPUTER ARCHITECTURE

Philip C. Treleaven

Department of Computer Science, University College London, London, UK

ABSTRACT

There is currently a veritable explosion of research into novel computer architectures, especially parallel computers. In addition, an increasing number of interesting parallel computer products are appearing. The design motivations cover a broad spectrum: (i) parallel UNIX systems (e.g. SEQUENT Balance), (ii) Artificial Intelligence applications (e.g. Connection Machine), (iii) high performance numerical Supercomputers (e.g. INTEL iPSC), (iv) exploitation of Very Large Scale Integration (e.g. INMOS Transputer), and (v) new technologies (e.g. Optical computers). This short paper gives an overview of these novel parallel computers and discusses their likely commercial impact.

1. PARALLEL COMPUTERS

In October 1981 Japan launched its 10 year national Fifth Generation project [9, 14] to develop knowledge information processing systems and processors. Since then other major industrial countries have started comparable national research programmes. In the United States the Strategic Computing Initiative, a $600 million programme funded by the Department of Defence, is investigating "Machine intelligence technology that will greatly increase national security and economic power". In the European Community the ESPRIT programme has a significant part of its $1.3 billion funding devoted to future computers. In addition, the individual European countries are funding major fifth generation programmes.

This competition between the national research programmes, to develop a new generation of computers, has been a catalyst for parallel computer research [1]. A major question for the design of future parallel computers is the choice of the parallel programming style. There are seven basic categories of computers (shown in Figure 1). They range from "low level" computers, such as Control Flow, that specify exactly how a computation is to be executed, to "high level" computers, such as Connectionist, that merely specify what is required. Associated with each category of computer is a corresponding category of programming language.

Firstly, there are control flow computers and procedural languages [13]. In a control flow computer (e.g. SEQUENT Balance, INMOS Transputer) explicit flow(s) of control cause the execution of instructions. In a procedural language (e.g. ADA, OCCAM) the basic concepts are: a global memory of cells, assignment as the basic action, and (sequential) control structures for the execution of statements.

<table>
<thead>
<tr>
<th>PROGRAMMING LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
</tr>
<tr>
<td>Languages</td>
</tr>
<tr>
<td>Languages</td>
</tr>
<tr>
<td>ADA, OCCAM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMPUTER ARCHITECTURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Flow</td>
</tr>
<tr>
<td>TRANSPUTER</td>
</tr>
</tbody>
</table>

Figure 1: Categories of Parallel Computers
Secondly, there are actor computers and object-oriented languages [16]. In an actor computer (e.g. APIARY) the arrival of a message for an instruction causes the instruction to execute. In an object-oriented language (e.g. SMALLTALK) the basic concepts are: objects are viewed as active, they may contain state, and objects communicate by sending messages.

Thirdly, there are data flow computers and single-assignment languages [13]. In a data flow computer (e.g. Manchester) the availability of input operands triggers the execution of the instruction which consumes the inputs. In a single-assignment language (e.g. ID, LUCID, VAL, VALID) the basic concepts are: data "flows" from one statement to another, execution of statements is data driven, and identifiers obey the single-assignment rule.

Fourthly, there are reduction computers and applicative languages [13, 16]. In a reduction computer (e.g. ALICE, GRIP) the requirement for a result triggers the execution of the instruction that will generate the value. In an applicative language (e.g. Pure LISP, SASL, FP) the basic concepts are: application of functions to structures, and all structures are expressions in the mathematical sense.

Fifthly, there are logic computers and predicate logic languages [16,17]. In a logic computer (e.g. ICOT PIM) an instruction is executed when it matches a target instruction. In a predicate logic language (e.g. PROLOG) the basic concepts are: statements are relations of a restricted form, and execution is a suitably controlled logical deduction from the statement.

Sixthly, there are rule-based computers and production systems languages [16]. In a rule-based computer (e.g NON-VON, DADO) an instruction is executed when its conditions match the contents of the working memory. In a production system language (e.g. OPS5) the basic concepts are: statements are IF... THEN... rules and they are repeatedly executed until none of the IF conditions are true.

Lastly, there are connectionist computers and semantic net languages [16]. A connectionist computer (e.g. Connection Machine) is based on the modelling at interneural connections in the brain. In a semantic net language (e.g. NETL) networks are used to define the connections between concepts, represented as structured objects.

However, since most parallel computers are still based on control flow, we believe that the best way to survey future parallel computers is by their application areas. Thus below we briefly examine each of the major application areas, namely: (i) fifth generation computers, (ii) numerical supercomputers, (iii) transaction processing systems, (iv) VLSI architectures, and (v) new technologies.

2. FIFTH GENERATION COMPUTERS

Fifth Generation computers are intended to be "knowledge-based" processing systems supporting AI applications. The design of Fifth Generation computers centres on the choice of the parallel programming style on which the computers are based. The three major approaches are: functional programming (e.g. Pure LISP), logic programming (e.g. PROLOG) and, what might be generally termed, knowledge-based programming including production system languages (e.g. OPS5) and semantic net languages (e.g. NETL). It is interesting to note that the main approach in Europe is reduction and data flow machines to support functional programming, whereas the main approach in Japan is logic computers and in the USA is rule-based and connectionist machines.

As an illustration of Fifth Generation computers we will briefly examine the MIT Connection Machine, based on the connectionist approach. Connectionists picture the brain as a densely-linked network of neurons (each neuron connected to as many as 10,000 others) capable of producing certain outputs when given certain inputs.

The Connection Machine [16] is designed to rapidly perform a few operations specific to AI, such as: (i) deducing facts from semantic inheritance networks; (ii) matching patterns against sets of assertions, demons or productions; (iii) sorting a set according to some parameter; and (iv) searching graphs for sub-graphs with a specific structure.
The Connection Machine is simply a collection of "intelligent" memory cells that are capable of connecting themselves to other such cells and hence representing some concept in the form of a semantic network. The initial design of the Connection Machine comprises 128K "intelligent" memory cells arranged as a uniform switching network. Each "intelligent" memory cell comprises a Communicator, Rule-Table, a State Register, a few words of storage, primitive ALU, and a message-register, as shown in Figure 2 below:

Figure 2: Connection Machine Node

The Communicators form a packet switched communications network being physically connected to only a few neighbouring communicators. When a communicator receives a message it decides (on the basis of the message address and local information) which direction the message should be routed, modifies the address and sends the message to the selected neighbour. The Rule Table is a simple table (shared with other cells) of rules that determine the behaviour of the "intelligent" memory cell under a received message. This behaviour may involve performing some elementary ALU operation, generating new messages or changing the state of the cell. The State Register is a vector of 10-50 bits storing so-called markers, arithmetic condition flags and the type of the cell. The storage area comprises registers of a total 384 bits of memory holding relative addresses of other "intelligent" memory cells.

A prototype Connection Machine is currently operational at Thinking Machines Inc. The prototype, uses a conservative LSI technology and comprises 64,000 nodes. In this prototype the processor has evolved from the initial design and looks less like a finite state machine and more like a general computer. It has been claimed by the designers that the 64,000-node processor has approximately 1000 times the logical-inference performance capabilities of current LISP workstations. Future connection Machines will be based on a custom LSI circuit suitable for use in a one million processor node machine.

3. SUPERCOMPUTERS

In the Supercomputer area, the 1970s was described as the decade of the single instruction-multi-data stream (SIMD) computer [5], which includes both vector computers (e.g. CRAY-1) and array processors (e.g. ICL DAP). However, the 1980s is becoming the MIND decade with computers ranging from a few CPUs (e.g. CRAY X-MP, ETA GF10) to parallel systems such as the Denelcor HEP and INTEL iPSC [5]. A number of the parallel numeric processors use the hypercube interconnection topology, which was pioneered in the CALTECH Cosmic Cube.
The Cosmic Cube [12] is a 2^6 Hypercube of 64 nodes, with each node comprising an INTEL 8086 and 8087 floating-point coprocessor, together with 128K bytes of local memory. The interconnection topology of the Cosmic Cube, with each node being connected to 6 neighbours, is shown below. Communication between the nodes is by queued message passing along the edges of the hypercube at a rate of 2M bits/s.

![Figure 3: Cosmic Cube – Hypercube (binary n-cube) Topology](image)

The programming model of the Cosmic Cube is based on concurrent processes that communicate by message passing, with a single node supporting a number of processes. Each process has a unique (global) ID that serves as an address for messages. All messages have headers containing the destination and the sender ID, and a message type and length. Messages are queued in transit, but message order is preserved between any pair of processes. Programs for the Cosmic Cube are written in conventional sequential languages (e.g. PASCAL, C) extended with statements and external procedures to control the sending and receiving of messages.

Even with current microelectronic technology, the 64-node Cosmic Cube is quite powerful for its cost and size. It can handle a variety of demanding scientific and engineering calculations 5-10 times faster than a VAX 11/780. A number of companies market versions of the Cosmic Cube [12]. The INTEL product is called the INTEL iPSC.

4. TRANSACTION PROCESSING SYSTEMS

An important new class of parallel computers has recently emerged in the marketplace, namely parallel UNIX machines. Examples include the ELXSI 6400, ENCORE Multimax, FLEXIBLE Flex-32 and SEQUENT Balance 8000 etc. [6]. These machines start in price at $60,000, ranging up to $200,000, and are aimed at the high performance end of the DEC VAX computers. These parallel UNIX machines are multi-processor systems, with between 2-20 processors, each with a local cache, and a global memory of up to 30M bytes, all connected by a common bus. In addition, the processors are typically the 32-bit NS32032. As an illustration we will examine the SEQUENT Balance 8000.

The SEQUENT Balance 8000 system consists of 2 to 12 NS32032 processors, a high speed 26.7M byte/s bus, and up to 28M bytes of global memory, as illustrated by Figure 4. There are three additional buses, namely the System Link and Interrupt Controller (SLIC) bus, the Small Computer System Interface (SCSI) bus and the 8-slot IEEE-769 Multibus. Each processor comprises five parts: the 32-bit CPU, a hardware floating point accelerator, a paged virtual memory management unit, a SLIC interface and an 8K byte cache. The system is managed by a
version of the UNIX 4.2 BSD operating system, enhanced to make the multi-processor invisible to any application.

![Diagram of SLIC Bus and System Bus]

Figure 4: Parallel UNIX SEQUENT Balance 8000

This parallel UNIX system centres on the concept of a "processor pool" with all code and data residing in the global memory. When a processor becomes idle it is allocated, from the pool, to the next process on the process list. As the processor executes the process, the code and data are fetched over the System Bus into its local cache, thus reducing communication overheads.

Parallel UNIX systems, such as the SEQUENT Balance, combine the benefits of UNIX's existing applications with the scalable power of a multi-processor, and they are likely to have a significant impact on the market for parallel computers.

5. VLSI ARCHITECTURES

The term very large scale integration (VLSI) is generally applied to a chip containing over 100,000 devices. VLSI has very different properties from the earlier microelectronic technologies: (i) design complexity is critical, (ii) wires occupy most space on a circuit, and (iii) non-local communication degrades performance. In the design of VLSI architectures to exploit parallelism two approaches are notable [11,15]. The first to design specialised parallel grids of processors such as Systolic Arrays [7]. The second to design general-purpose, reduced instruction set (RISC) [10] parallel microcomputers like the INMOS Transputers [2]. Below we examine the Transputer.

INMOS' Transputer comprises a family of 16- and 32-bit microcomputers, capable of operating alone as a 10-MIPS (million instructions per second) processor or as a component of a parallel network of Transputers. Each microcomputer, as shown below, consists of four main parts: a reduced instruction set processor, 2K bytes of static RAM, a 32-bit multiplexed memory interface and four INMOS standard serial links providing concurrent message passing to other Transputers. The processor has built-in support for multi-processing and parallelism. The execution state of each process is defined by six registers. These registers are arranged as a three-register evaluation stack, together with an instruction pointer, a workspace pointer, and an operand register. Instructions are eight bits, comprising a 4-bit function code and a 4-bit data value. Operands longer than four bits are built up four bits at a time in the operand register. Basic arithmetic instructions execute in 50 nsecs and a process switch takes only 600 nsecs.
Communication between Transputers is handled by the links. Each link implements two channels, an output and an input, over which messages are transmitted as a series of bytes.

Parallel programming in Transputers, and its OCCAM programming language [2], is based on communicating processes and message passing using explicitly defined channels. A network of Transputers corresponds directly to a network of processes, with each Transputer supporting one or more processes in a timeshared fashion.

6. NEW TECHNOLOGIES

Advances in technology have perhaps always constituted the driving force for developments in computer architecture. Three technologies that could have a big impact on future parallel computer design are: in the short term, Gallium Arsenide (GaAs) processors [8]; in the medium term, Optical computers [3]; and in the long term, Biological/Molecular computers [4].

GaAs technology [8] has made rapid progress in recent years particularly in the area of digital chip complexity. When comparing GaAs with silicon, its two main advantages are higher switching speed and greater resistance to adverse environmental conditions. But GaAs is inferior to silicon in terms of cost (of material and lower yield) and transistor count (related to yield and power considerations). However, for certain applications the advantages of GaAs are critical, leading to increasing interest in GaAs processors. A good discussion of processor architectures suitable for GaAs is given by Milutinovic et al [8].

Optical techniques for information processing have also made rapid advances in recent years. Within this area, the term Optical computing is defined [3] as: the use of optical systems to perform numerical computations on one-dimensional or multi-dimensional data that are generally not images. The goal of this work is to build an Optical binary digital computer which uses photons as the primary information carrying medium rather than electrons. The potential advantages of optical computers include (i) high space-bandwidth and time-bandwidth products, (ii) they are inherently two dimensional and parallel, (iii) optical signals can propagate through each other in separate channels with essentially no interaction, and (iv) optical signals can interact on a subpico-second time scale. Thus the potential for Optical parallel computers is clear. Discussions of the possible organisation of optical computers is given in [3].

Finally, in the longer term Biological or Molecular computers promise an exciting research area. Although no molecular computing device seems so far to have been constructed [4], the possibility of organic switching devices and conducting polymers may come about from current
developments in polymer chemistry, biotechnology, the physics of computation, and computer science. So far, however, there is no clear consensus as to the viability of biological/molecular computing or the best strategy for such computation. A good introduction to the topic is given in [4].

7. FUTURE TRENDS

Many factors support the adoption of a radically new generation of parallel computers. Firstly, the handling of non-numerical data such as sentences, symbols, speech, graphics and images is becoming increasingly important. Secondly, the processing tasks performed by computers are becoming more "intelligent", moving from scientific calculations and data processing, to artificial intelligence applications. Thirdly, computing is moving from a sequential, centralised world to a parallel decentralised world in which large numbers of computers are to be programmed to work together in computing systems. Lastly, today's computers are still based on the thirty-year-old von Neumann architecture.

A number of trends in computer architecture are already discernible. Firstly, there is the growing agreement that future parallel computers will be constructed from large numbers of identical units (each with processing, memory and communications) suitable for implementation in VLSI and wafer-scale technology. The best current example is the INMOS Transputer [2] microcomputer.

Secondly, there is the need to integrate symbolic and numeric computing. Thus it is to be expected that the architecture of Fifth Generation computers and numeric Supercomputers will converge.

Thirdly, there are the increasing numbers of interesting parallel computer products, such as the parallel UNIX machines, that are appearing in the market. I believe these parallel "operating system" machines will become an industry standard over the next three years for mainframes, minicomputers and workstations, leading to parallel computers becoming the accepted commercial norm.

Lastly, in the longer term say 10-20 years we have the stimulating prospect that Optical and Biological parallel might become available.

*   *   *

REFERENCES


PARALLEL ARCHITECTURES FOR NEUROCOMPUTERS

Philip C. Treleaven

Department of Computer Science, University College London, London, UK

ABSTRACT

Recent advances in "neural" computation models will only demonstrate their true value with the introduction of parallel computer architectures designed to optimise the computation of these models. There are three basic approaches for realising neurocomputers. Firstly, special-purpose neural network hardware implementations that are dedicated to specific models and therefore have potentially a very high performance. Secondly, neural network simulators utilising conventional hardware which are slow but allow implementation of a wide range of models. Lastly, general-purpose neurocomputers will provide a framework for executing neural models in much the same way that traditional computers address the problems of "number crunching", for which they are best suited. This framework must include a means of programming (i.e. operating system and programming languages) and the hardware must be reconfigurable in some manner.

This paper surveys current work on parallel neurocomputer architectures, concentrating on Special-Purpose hardware implementations and on General-Purpose systems.

1. BACKGROUND

Even a small child can recognize faces, whereas a Supercomputer is stretched to its limits performing such computations. In contrast, an inexpensive computer excels at a series of laborious calculations beyond most humans. This computational contrast between computers and humans is striking. Further it suggest two fundamental domains of computation: Symbol Processing (of computers) and Pattern Processing (of humans).

In crude terms, the brain [6,33,41] is a massively parallel natural computer composed of 10-100 billion brain cells (i.e. neurons), each neuron connected to about 10,000 others. Neurons seemingly perform quite simple computations. The principal computation is believed to be the calculation of a weighted sum of its inputs, comparing this sum with a threshold, and forming its output if this threshold is exceeded.

Yet the brain is capable of solving difficult problems of vision and language in about half a second (i.e. 500 milliseconds). This is particularly surprising given that the response time of a single neuron is in the millisecond range and taking into account propagation delays between neurons. Thus the brain must complete these pattern processing tasks in less than 100 steps.

The pattern processing class of problems, covering pattern recognition and learning applications, are trivial for brains but are far from readily solvable by traditional (symbol processing) computers. There has been a renewed belief that to solve demanding pattern processing problems, parallel computing systems are needed which emulate the organisation and function of neurons.

1.1. Neurons

Since the basis of Neurocomputers is consideration of the structure of brains, the key properties of neural systems will be reviewed.

The basic building block is the neuron [41]. A neuron (see Figure 1) consists of a cell body called a soma, dendrites which receive input and branch out, and an axon that
carries the output of the cells, one to another. Junctions between neurons, called synapses, occur either on the cell body or on spine-like extensions called dendrites. The neuron, in its simplest form can be considered a threshold unit that collects signals at its synapses and sums them together using its internal summer. If the collected signal strength is great enough to exceed the threshold, a signal is sent out from the neuron by way of its axon.

![Synapse Diagram]

**Figure 1**: The Neuron

### 1.2. Artificial Neural Networks

Artificial neural networks are neurally inspired mathematical models that use a large number of primitive processing elements (PEs) for pattern processing. Typically in neural networks, PEs are organised into layers with each PE in one layer having a weighted connection to each PE in the next layer. This organisation of PEs and weighted connections creates a neural network, also known as an artificial neural system (ANS). A neural network learns patterns by adjusting the strengths (weights) of the connections between PEs, analogous to synaptic weights. Through these adjustments a neural network exhibits properties of generalisation and classification.

A PE, such as the $j$th PE in Figure 2 comprises inputs $I_1 \ldots I_n$, with weights $W_1 \ldots W_n$, from the layer below, a summation function, a threshold function $f$ and the net value $OUT_j$, the output of the threshold function.

![Neural Network Processing Element Diagram]

**Figure 2**: A Neural Network Processing Element

Each component of the PE corresponds to a component of the neuron, as shown below:

<table>
<thead>
<tr>
<th>Neuron</th>
<th>Processing Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dendrites</td>
<td>Inputs</td>
</tr>
<tr>
<td>Synapses</td>
<td>Weights</td>
</tr>
<tr>
<td>Summer</td>
<td>Summation Function</td>
</tr>
<tr>
<td>Threshold</td>
<td>Threshold Function</td>
</tr>
<tr>
<td>Axon</td>
<td>Net Output</td>
</tr>
</tbody>
</table>

**Figure 3**: Correspondence of a Neuron and PE

There are many different classes of neural networks. Hecht-Nielsen states [24] that there are at least 30 different types of neural network models, currently being used in
research and/or applications, of which 14 types are in common use. Perhaps the best known neural network models are: the Hopfield model [26], the Boltzmann Machine model [2] and the Error Propagation model [39]. These models are discussed further in the next section.

In summary, neural networks [24] are massively parallel interconnected networks of simple (usually) adaptive processing elements, and their hierarchical organisations, which are intended to interact with the objects of the real world in the same way as biological nervous systems do.

1.3. Spectrum of Parallel Architectures

When considering the set of possible parallel architectures for the basis of a Neurocomputer, as shown by Figure 4 they range from dedicated hardware, analogous in complexity to RAMs, to simulations on conventional computers [40].

There are three distinct approaches currently being taken for supporting neural network models:

- **Special-Purpose Hardware** - specialised neural network hardware implementations that are dedicated to a specific neural network model and therefore have a potentially very high performance.

- **General-Purpose Neural Architectures** - generalised neural computers for emulating a range of neural network models, thus providing a framework for executing neural models in much the same way that traditional computer address the problems of "number crunching".

- **Simulations** - neural network simulators utilising conventional hardware which are slow but allow support of a wide of models.

![Figure 4: Spectrum of Neurocomputer Architectures](image)

2. CANDIDATE PARALLEL ARCHITECTURES

Research into parallel architectures for Neurocomputers largely falls into two camps:

- **Special-Purpose architectures** and

- **General-Purpose architectures**

This is mirrored by the structure of the paper. In this section we examine, firstly the candidate neural network models for direct hardware implementation and secondly the candidate architectures for general-purpose neurocomputers.

2.1. Special-Purpose Architectures

The approach for *Special-Purpose* neurocomputer architectures is to directly implement a specific neural network model in hardware to give a very high performance sys-
tem. Basically any neural network model could be chosen, although currently a Hopfield associative memory model is typically favoured, because of its simplicity.

In general, neural network (or Connectionist [4,14,15] models are of two broad classes, namely associative memories and categorisation or learning systems. With Associative Memories, information can be retrieved based on the content of the memory (auto-associator), or a relationship between remembered pieces of information (pair-associator). In addition, with both types of associated memory, a corrupted "key" will lead to a recall of the nearest stored event.

With Learning Systems, data is presented repeatedly according to a set of rules, and the task is for the system to extract the underlying patterns. Learning systems can be further classified into supervised and unsupervised learning. During supervised learning, as in the Boltzmann Machine and the Backpropagation of Errors algorithm, expected results govern the learning process. The Competitive learning algorithm is an example of unsupervised learning.

Hecht-Nielsen has identified [24] 14 neural network models in common use:

- **Adaptive Resonance (ART)** - a class of networks that form categories for the input data, and where the coarseness of the categories is determined by the value of a selectable parameter.
- **Avalanche (AVA)** - a class of networks for learning, recognising and replaying spatiotemporal patterns.
- **Backpropagation (BPN)** - a multilayer network that minimises mean square mapping error.
- **Bidirectional Associative Memory (BAM)** - a class of single-stage heteroassociative networks.
- **Boltzmann Machine (BCM)** - a class of networks that use a noise process to find the global minimum of a cost function.
- **Brain State in a Box (BSB)** - a single-stage auto-associative network that minimises the mean square error.
- **Cerebellatron (CBT)** - learns the averages of spatiotemporal command sequence patterns and relays these average command sequences on cue.
- **Counterpropagation (CPN)** - a network that functions as a statistically optimal self-organising lookup table and probability density function analyser.
- **Hopfield (HOP)** - a class of single-stage auto-associative networks without learning.
- **Lernmatrix (LRN)** - a single-pass, non-recursive, single-stage associative network.
- **Madaline (MDL)** - a bank of trainable linear combiners that minimise mean square error.
- **Neocognitron (NEO)** - a multilayer hierarchical character recognition network.
- **Perceptron (PTR)** - a bank of trainable linear discriminants.
- **Self-Organising Map (SOM)** - a network forming a continuous topological mapping from one compact manifold to another, with the mapping metric density varying directly with a given probability density function on the second manifold.

Two of the most important neural network models are the Hopfield model (i.e. auto associator), which is typically chosen for special-purpose hardware implementation, and the Backpropagation model, the most popular neural network learning model in use today. Below the Hopfield model is briefly described. (The Backpropagation model is described in the companion paper on Programming Languages for Neurocomputers [44].)

**Hopfield Model**

The resurgence of interest in neural networks is largely due to Hopfield and his 1982 paper [26] proving that a neural network of interconnected processing elements will seek an energy minima.
The Hopfield model acts on a binary input vector $I$ mapping it to a binary output vector $O$, both of $n$-elements, using a $n \times n$ weight matrix $W$. The model comprises two algorithms, namely for storage and for recall.

The Hopfield learning algorithm has the following steps:

1. Set the output vector to the input vector $O = I$
   
   $\text{for } i = 1 \text{ to } n \text{ do}$
   
   \hspace{1cm} $O_i = I_i$
   
   $\text{enddo} \hspace{1cm}$  \text{ex. } $I = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$
   
   \hspace{1cm} $O = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$

2. Initialise the weight matrix to zeroes $W = 0$
   
   $\text{for } i = 1 \text{ to } n \text{ do}$
   
   \hspace{1cm} $\text{for } j = 1 \text{ to } n \text{ do}$
   
   \hspace{1.5cm} $W_{ij} = 0$
   
   $\text{enddo} \hspace{1cm}$  \text{ex. } $W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

3. Generate the weight matrix $W_{ij} = (2I_i - 1)*2(I_j - 1)$ $i <> j$
   
   $\text{for } i = 1 \text{ to } n \text{ do}$
   
   \hspace{1cm} $\text{for } j = 1 \text{ to } n \text{ do}$
   
   \hspace{1.5cm} $\text{if } i <> j \text{ then}$
   
   \hspace{2cm} $W_{ij} = (2I_i - 1)*2(I_j - 1)$
   
   \hspace{1.5cm} $\text{endif}$
   
   $\text{enddo} \hspace{1cm}$  \text{ex. } $w = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \end{bmatrix}$

Next the Hopfield Recall algorithm which has the following steps:

1. Sum all connection strengths entering output $O_j$, $\sum W_{ij}$
   
   $\text{for } j = 1 \text{ to } n \text{ do}$
   
   \hspace{1cm} $\text{sum} = 0$
   
   \hspace{1cm} $\text{for } i = 1 \text{ to } n \text{ do}$
   
   \hspace{1.5cm} $\text{sum} = \text{sum} + W_{ij}$
   
   $\text{enddo}$
   
   $\text{NET}_j = \text{sum}$

2. Transform each element of $\text{NET}_j$ to binary value.
   
   $\text{for } j = 1 \text{ to } n \text{ do}$
   
   \hspace{1cm} $\text{if } \text{NET}_j > 0 \text{ then}$
   
   \hspace{1.5cm} $\text{NET}_j = 1$
   
   \hspace{1cm} $\text{else}$
   
   \hspace{1.5cm} $\text{NET}_j = 0$
   
   $\text{endif}$

And the vector $\text{NET}$ recalled from matrix, is the same vector the neural network was taught, depending on the number of stored patterns. Note that although the Hopfield model is designed to store a single binary vector, it can easily be extended to store several binary vectors.

The Hopfield model is typical of a class of single layer neural network systems, but many real-world problems cannot be represented by such neural networks. The solution to this problem is to introduce a third layer, called the hidden layer, between the input and output layers. The best known three layer model is Backpropagation.
2.2. General-Purpose Architectures

Design of general-purpose parallel computers that are candidates for neurocomputer architectures, centre around a small set of parallel programming models [43]. As shown in Figure 5, each programming model comprises a computer architecture and a corresponding category of programming languages.

<table>
<thead>
<tr>
<th>COMPUTATION</th>
<th>DOMAIN</th>
<th>Symbolic</th>
<th>Processing</th>
<th>Pattern Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAMMING</td>
<td>LANGUAGE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedural</td>
<td>OCCAM, ADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smalltalk</td>
<td>SMALLTALK</td>
<td>Single-Assignment</td>
<td>SISAL</td>
<td>Applicative Logic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pure LISP</td>
<td>PROLOG</td>
</tr>
<tr>
<td>Control Flow</td>
<td>COMPUTER</td>
<td>Object-Oriented</td>
<td>DOOM</td>
<td>Data Flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object-Oriented</td>
<td>MANCHESTER</td>
<td>GRIP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Parallel Computer Architectures

These parallel architectures and their associated programming languages have the following properties:

- **Control Flow** - in a control flow computer (e.g. Sequent Balance, Intel iPSC, INMOS Transputer) [17,43] explicit flows of control causes the execution of instructions. In their procedural languages (e.g. ADA, OCCAM) the basic concepts are: a global memory of cells, assignment as the basic action and explicit control structures for the execution of statements.

- **Object-Oriented** - in an object-oriented computer (e.g. APIARY, DOOM) [43] the arrival of a message for an instruction causes the instruction to execute. In an object-oriented language (e.g. SMALLTALK, POOL) the basic concepts are: objects are viewed as active, they may contain state, and objects communicate by sending messages.

- **Data Flow** - in a data flow computer (e.g. Manchester, MIT) [43] the availability of input operand triggers the execution of the instruction which consumes the inputs. In a single-assignment language (e.g. SISAL, ID, LUCID) the basic concepts are: data "flows" from one statement to another, execution of statements is data driven, and identifiers obey the single-assignment rule.

- **Functional** - in a reduction computer (e.g. ALICE, GRIP) [43] the requirement for a result triggers the execution of the instruction that will generate the value. In an applicative language (e.g. Pure LISP, ML, FP) the basic concepts are: application of functions to structures, and all structures are expressions in the mathematical sense.

- **Logic** - in a logic computer (e.g. ICOT PIM, BULL DDC) [43] an instruction is executed when it matches a target pattern and parallelism, or backtracking, is used to execute alternatives to the instruction. In a predicate logic language (e.g. PROLOG) the basic concepts are: statements are relations of a restricted form, and execution is a suitable controlled logical deduction from the statements.

- **Rule-Based** - in a rule-based computer (e.g. NON-VON, DADO) [43] an instruction is executed when its conditions match the contents of the working memory. In a production system language (e.g. OPS5) the basic concepts are: statements are IF...THEN... rules and they are repeated executed until none of the IF conditions are true.
- **Cellular Array** - in cellular array computers each processor is connected to its "near-neighbours" in a regular pattern that matches the flows of data and control in the target computation. Two subsidiary classes of cellular arrays are computational arrays (e.g. Connection Machine, MPP, DAP, CLIP) [17,25,43] and systolic array processors (e.g. WARP). A class of programming languages corresponding closely to the computational arrays, are semantic network languages (e.g. NETL, IXL).

When considering the above categories of parallel architectures as the basis of a neurocomputer, the most appropriate is Cellular Arrays. (The other six categories are based on far more complex computational models than required by the simple "threshold" models typical of neural computing.) The computational framework of Cellular Arrays is consistent with an idealised neural structure and supports well the distributed nature of data in these neural network models. An additional advantage is that frequently, even with current levels of VLSI processing, many processing elements can be fabricated on a single chip. Two specific Cellular Arrays, namely the Programmable Systolic Chip [16] and the Connection Machine [25,43], are indicative of a "general-purpose" neurocomputer.

**Programmable Systolic Chip**

Programmable Systolic Chips (PSC) [16] can be assembled into a number of regular topologies (e.g. linear, 2D array etc.) to support the family of systolic algorithms. Once the PSCs are connected, they are configured for a specific systolic algorithm by downloading identical code into each PSC. The PSCs then operate as a synchronous pipeline with the data being pumped from chip to adjacent chip.

![Programmable Systolic Chip](image)

Figure 6: Programmable Systolic Chip

A PSC processor, see Figure 6, consists of five functional units that operate in parallel and communicate simultaneously over the 3 buses. The five functional units are: a 64x60-bit microcode dynamic RAM and a microsequencer, a 64x9-bit dynamic DRAM register file, an ALU, a multiplier-accumulator (MAC), plus three input and three output ports. Next we briefly examine the Connection Machine.

**Connection Machine**

The Connection Machine [25] is designed for concurrent operations on a knowledge base represented as a semantic network. A semantic network is a directed graph where the vertices represent objects (e.g. sets) and the arcs represent binary relations (e.g. set membership) required by the knowledge to be represented. A Connection Machine comprises 64K identical "intelligent" memory cells connected as a hypertorus structure. A Connection Machine cell (see Figure 7) is a bit serial processor, comprising a few registers, an ALU, a message buffer, and a finite state machine. All cells are configured with the same program, known as the "Rule Table", which defines the next state and output functions of the finite state machine. A cell reacts to an incoming message according to its internal state and the message type, and performs a sequence of steps that may involve arithmetic or storage operations on the contents of the message and the registers, sending new messages, and changing its internal state.
Having introduced the candidate parallel architectures for Neurocomputers, we next examine some of the special-purpose neurocomputers under development.

3. SPECIAL-PURPOSE NEUROCOMPUTERS

When considering the implementation of neural network models, the basic corresponding hardware structure is the crossbar switch [28,29], shown in Figure 8a. This crossbar switch can be enhanced for neural network models by the introduction of lateral feedback. From this organisation it is possible to devise [29] a complete operational module for neural systems, as proposed by Kohonen [29] and illustrated by Figure 8b.

Figure 8: Neurocomputer Components
(from Kohonen [29])

Kohonen states [29] that the most natural topology of a neural network would be two-dimensional and the distribution of lateral feedbacks within the system could be the same around every neuron.

3.1. Overview

Developments of special-purpose, very-high-performance, typically analog circuits for neural networks are underway at a number of locations [19,20,42]. Leaders in this field are: Jackel at AT&T Bell Laboratories (Holmdel), Lambe at the NASA Jet Propulsion Laboratories in Los Angeles, Mead at CALTECH, and Goser at the University of Dortmund, FRG.
Implementing large numbers of individually primitive processing elements in VLSI technology is intuitively appealing. In addition, analog circuits generally occupies less area than the equivalent digital circuits. However, a number of technology-dependent limitations are encountered:

- **Cost** - chip area is the principal cost in VLSI.
- **Power** - analog processing elements typically require a high power consumption.
- **Parameter Variation** - fabrication of small devices introduces variations effecting the currents transferred.

### 3.2. CALTECH

Researchers at Caltech are investigating VLSI architectures for implementing neural networks [42], specifically networks based on the Hopfield model. The Caltech circuit for the Hopfield model is illustrated by Figure 9.

![Circuit Diagram for Hopfield Model](image)

This Hopfield circuit consists of three major components: amplifiers, interconnection matrix and capacitances. The collection of amplifiers (cf. neurons) with gain function \( V = g(v) \) are connected by the passive interconnection matrix which provides the unidirectional synapses, connecting the output of one neuron to the input of another. The strength of this interconnection is given by conductance \( Gi = Go Tij \). Lastly, the capacitances determine the time evolution of the system.

The Caltech chip, based on the above Hopfield circuit, contains 22 processing elements and a full interconnection matrix of 462 elements. The chip, fabricated in 4um NMOS technology, measures 6700um x 5700um, and has 53 I/O pads. This was followed by a 289 neuron CMOS chip.

### 3.3. AT&T Associative Memory Chip

AT&T are investigating CMOS associative memory chips that contain over 50 artificial neurons on a single chip using a combination of analog and digital VLSI technologies, together with a special microfabrication process. The chips are being used in pattern recognition where they perform feature extraction.

One associative memory chip implements a connectionist model of a neural network, and consists of 54 amplifiers plus a programmable coupling network where each amplifier can be connected to every other amplifier. Figure 10 shows a schematic of the implemented circuit. It consists of an array of 54 amplifiers with their inputs and outputs interconnected through a matrix of resistive coupling elements. All of these elements are programmable i.e. a resistive connection can be turned on or off.

The connections between the individual "neurons" are provided by amorphous-silicon resistors which are placed on the CMOS chip in the last stage of fabrication using
electron-beam direct-writing. The associative memory chip was fabricated in 2.5um CMOS and contains roughly 75,000 transistors in an area 6.7mm x 6.7mm. Ninety percent of the chip area is used for the coupling network. Extensive tests were made with 10 stable states of 40 bits length programmed into the associative memory circuit. The time it takes the circuit to converge to a stable state is between 50 and 600 ns.

4. GENERAL-PURPOSE NEUROCOMPUTERS

Neurocomputing is a fundamentally different domain of computation, to traditional computing. Neurocomputing performs "pattern processing", while traditional computers perform "symbol processing" specified by an explicit series of instructions. However, traditional computers are extremely flexible for symbol processing. What now is required is a complementary general-purpose neurocomputer able to support a spectrum of neural network models.

For a general-purpose Neurocomputer; a number of properties are identifiable:

- **Modular Processing Element** - PEs should be modular and hence replicatable, thereby each PE should be a self-contained unit comprising processor, communications and memory.

- **Primitive Processing Element** - to make large neurocomputers (with millions of PEs) feasible, a PE must be primitive, allow a number to be packed on a single VLSI chip or wafer.

- **Regular Communications** - to allow neurocomputers to be extensible, regular communications structures are required, especially to overcome connectivity limitations of VLSI.

- **Asynchronous Operation** - the potential to match the heterogeneous richness of the brain, neurocomputers may need to become multi-instruction-multi-data stream (MIMD) devices.

- **Programmability** - for a neurocomputer to be general-purpose, and hence support a wide range of neural network models, the PEs must be programmable, both in terms of interconnections and the function supported by a PE.

- **Stability** - any asynchronous parallel system requires the processing and communications to provide inherent stability in all programmed situations.
• **Virtual Processing Elements** - for a neurocomputer to execute potentially any massively parallel neural network, the concept of virtual processing elements that can be "paged" onto the neurocomputer from a backing store seems inevitable.

Below we examine some of the general-purpose Neurocomputers that have been developed.

### 4.1. Overview

Neurocomputer development is a subject still in its infancy, hence the number of complete working Neurocomputers is limited [7,8,30,34,36]. Below we review the major developments in the USA, Europe and Japan.

In the USA neurocomputing products are being marketed by such corporations as TRW, Hecht-Nielsen Neurocomputer (San Diego), Nestor Inc. (Rhode Island), Verac Inc. (San Diego), AIWARE Inc. (Cleveland), Neural Systems Inc. (Vancouver), NCI (New Jersey), Neuraltech Inc. (Portola Valley), Neuronics Inc. (Chicago) and SAIC (Tucson). The pioneer of Neurocomputer design is Hecht-Nielsen, who has produced most of the commercially available general-purpose neurocomputers in which an arbitrary interconnectivity of the PEs can be defined. At TRW, Hecht-Nielsen produced the Mark III and Mark IV machines. The Mark IV has 200,000 processing elements, each capable of 25 interconnections. Subsequently, Hecht-Nielsen's own company HNC has developed the ANZA system [22], a coprocessor board which interfaces to an IBM PC-AT. ANZA has 30,000 processing elements and allows a total of 300,000 interconnections between all the PEs. Lastly, Nester Inc. produces neurocomputer systems for handwritten-character-recognition based on the work of Cooper from Brown University.

In Europe, Neurocomputers have been produced by Aleksander of Imperial College London, by Kohonen and by Garth of TI. Aleksander has developed a series of systems called Wisard. Wisard II is organised as a hierarchical network of PEs, with each PE being constructed from commercial RAM. The RAMs address inputs are used to detect binary patterns, with the input field for an image (comprising 512 x 512 binary pixels) being connected to the first layer of PEs. Kohonen, at Helsinki University, a pioneer in associative "memory neural" networks, has experimented with several "neural network" distributed memories. He has recently completed a commercial-level neurocomputer based on signal processor modules and working memories to define a set of virtual processing elements. This neurocomputer, optimised for speech recognition, allows 1000 virtual processing elements with 60 interconnections. It can perform a complete spectral analysis and classification in phonemes every 10 ms. Lastly, Garth of TI, working with Cambridge University, has developed NETSIM [18], a 3-D array of processing elements, each based on specially designed chips plus a 80188 microprocessor.

In Japan [27], Nakano of Tokyo University has completed a number of neurocomputers, some dating from 1970. The Association, his best known neural network system, is a hardware, correlation-matrix type of associative memory. A number of companies such as Fujitsu are also working on neurocomputers.
The properties of the above Neurocomputers are summarised in Figure 11.

<table>
<thead>
<tr>
<th>Neurocomputer</th>
<th>virtual PEs</th>
<th>interconnects</th>
<th>updates/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNC ANZA</td>
<td>30K</td>
<td>300K</td>
<td>25K</td>
</tr>
<tr>
<td>TRW MARK III</td>
<td>65K</td>
<td>1M</td>
<td>450K</td>
</tr>
<tr>
<td>TRW MARK IV</td>
<td>256K</td>
<td>5.5M</td>
<td>5M</td>
</tr>
<tr>
<td>IBM NEP</td>
<td>1M</td>
<td>4M</td>
<td>800K</td>
</tr>
<tr>
<td>NETSIM</td>
<td>256x27K</td>
<td>64Kx27K</td>
<td>4M</td>
</tr>
</tbody>
</table>

Figure 11: Comparison of Neurocomputers

4.2. HNC ANZA

The ANZA Neurocomputer [22], developed and marketed by Hecht-Nielsen Neurocomputer Corporation, is designed to support any neural network algorithm. The ANZA system comprises: the ANZA neurocomputer co-processor board for a PC AT, the User Interface Subroutine Library, and basic netware packages for the common neural network algorithms.

The ANZA co-processor board plugs into the backplane of a PC AT. The board is based on a Motorola M68020 plus a M68881 floating point co-processor, with 4M bytes of dynamic RAM to store the network. ANZA is capable of implementing 30,000 PEs with 480,000 interconnections. These interconnections are updated at 25,000 interconnections per second during learning and 45,000 in feed forward mode.

The User Interface Subroutine Library (UISL) is a collection of routines providing access to the ANZA system functions. Examples include: load network, set learning etc. Lastly, the Basic Netware Package contains five of the classic neural network algorithms in a parameterised specification that can be configured for a specific user application. These algorithms are: Backpropagation, Spaciotemporal (Formal Avalanche), Neocognition, Hopfield (plus Bidirectional Associative Memory) and Counter-Propagation networks. In these networks the interconnection geometry and the transfer equations are already specified. However, the number of PEs, their initial state and weight values, learning rates and time constants, are all user selectable.

4.3. TRW Mark III & IV

The TRW Neurocomputer family consists of the Mark II simulator, the Mark III neurocomputer workstation and the Mark IV high-speed neurocomputer [31]. All share the common Artificial Neural System Environment (ANSE) user environment.

The Mark III neurocomputer consists of up to 15 physical processors, each built from a Motorola M68020 microprocessor and a M68881 floating point co-processor, all connected to a common VME bus (see Figure 12). A neural network to be processed is distributed across the local memories of the PEs. Currently, the Mark III supports 65,000 virtual processing elements with over 1,000,000 trainable interconnections, and can process 450,000 interconnections per second.

The Mark IV neurocomputer is a single high-speed, pipelined processor using virtual PEs and interconnection structure. Here the bulk of the hardware is devoted to forming the interconnections. The Mark IV supports 236,000 virtual processing elements with over 5.5M trainable interconnections, and is capable of processing 5M interconnections per second, including the pre-weighting function and learning law.
4.4. IBM NEP

IBM has developed a complete experimental neural network programming environment, called Computation Network Environment (CONE) [21]. CONE comprises: a Network Emulation Processor (NEP) - a cascordable parallel co-processor for the PC; a Network Interactive eXecution Program (IXP); and a high level Generalised Network Language (GNL).

The major functional blocks of a NEP [12] are shown in Figure 13. They consist of 6 major units: a 5MIPS T1320 signal processor, a 64K word x 16-bit SRAM data memory, a 4K word x 16-bit SRAM program memory, an 80M bytes/sec local I/O interface, a global interface to the PC host and a 100M bytes/sec inter-NEP NEPBUS interface.

Figure 13: IBM Network Emulation Processor

Up to 256 NEPs can be cascaded in a uni-directional interprocessor communications network (NEPBUS), supporting in total 1M virtual PEs and 4M interconnections. To preserve interprocessor communication bandwidth, each NEP contains a high speed "local I/O" unit for the attachment of real-time I/O devices. Both the NEPBUS and the local I/O interface are FIFO buffered, allowing a group of NEPs to asynchronously update the state of their respective portions of a large neural network.
Each NEP can simulate about 4,000 virtual PEs and 16,000 interconnections, with 30-50 complete network updates per second. The number of PEs emulated (by each NEP) can be increased by decreasing the total number of interconnections. In addition, the length of a network update cycle can be reduced by dividing a network across more of the NEPs, with a speed increase proportional to the number of NEPs.

4.5. IC WISARD

The WISARD systems [3] are a series of neurocomputers specifically for image processing, developed by Aleksander and commercialised as the WISARD/CRS1000 by Computer Recognition Systems Ltd.

Design of a WISARD system centres on an array of RAM cells used as a set of Discriminators for the image to be processed. These Discriminators operate in an analogous way to a Hologram. The conceptual structure of WISARD is shown in Figure 14. Consider the processing of a 512x512 bit binary image. From this binary image groups of n bits are extracted to form n-tuples, using a random but fixed mapping. In this case the image contains $2^{18}$ bits, so for $n=8$, $2^{15}$ n-tuples are taken. Each n-tuple is then used to address a specific RAM cell in a Discriminator. Conceptually, in this example, $2^{15}$ RAM cells, each of 256 bits are needed. However, for efficiency these cells can be grouped into Discriminators. By using RAM arrays organised as k-bit words, k Discriminators may be provided simultaneously.

![Figure 14: Conceptual Structure of WISARD](image)

Initially all cells in these Discriminators are set to zero. During training, a Discriminator is selected and ones are entered into all cell addressed by the image. If this image is presented again, the effect is for the Discriminator to produce a logical one on all its outputs. A partial image produces a reduced set of ones.

CRS has developed the WISARD by adapting its CRS1000 automatic inspection system. It operates as follows. Initially a video image is captured, and WISARD fetches n-tuples from the image store, thresholding the data with a programmable comparator. The address in Discriminator memory is made up from an auto-incrementing counter and the 5-tuple (say ABCDE) read from the image memory:
In WISARD up to 16 Discriminators are directly supported by having a 16-bit wide memory Discriminator memory. Tuples ranging from 4 to 8-tuples are sufficient for most industrial applications.

4.6. NETSIM

The NETSIM [18] Neurocomputer has been developed through a collaboration of Texas Instruments (UK) and Cambridge University.

A NETSIM system, as illustrated by Figure 15 consists of a collection of neural network simulator cards physically connected in a 3-dimensional array, with a PC host acting as a front-end. Each NETSIM card is an autonomous processing element, comprising: an industry-standard 80188 microprocessor; two custom chips, a solution engine and a communications processor; and three memories for synapses, program and BIOS.

![3-D ARRAY OF NETSIM CARDS](image)

Figure 15: NETSIM Neurocomputer System

The Solution enquire, CF30111, operates as a back-end vector processor for the microprocessor. Assuming a network "solution" is given by:

\[ O_j = f \left( \sum I_i T_{i,j} \right) \]

The Solution enquire computes the sum of products between the input vector I and the relevant synapse vector T in its memory, and returns the result as a 16 bit integer to the microprocessor. The microprocessor then computes the non-linear function \( f() \) to produce the output of the neuron. The output is then passed to the communications processor for transmission to the network to which the neuron is logically connected.

The Solution chip performs four instructions:

- **Dummy cycle** - chip management e.g. clear register
- **Repeat-multiply-sum** - solve the network by multiplying an input vector by the corresponding synapse and adding it to the sum register
- **Read-write** - move data within memory, allowing access to the synapse and input memory space from the microprocessor

- **Repeat-multiply-sum-write** - update synapses by multiplying two 8 bit vectors (e.g., Input x Delta) with prescale, adding the product to a 16-bit term (T_{ij}) and storing it as a new term (T'_{ij}).

Organisation of the Communications chip, centres on a 64-bit message register, where the first two bytes represent the messages destination address (relative to the sending mode) and the remaining bytes are for data. The addressing scheme allows messages to be transmitted to +/-15 NETSIM cards in each of the three dimensions.

In conclusion, the NETSIM Neurocomputer has been shown to support the majority of common neural network algorithms including the Hopfield model and the Backpropagation model. Each NETSIM card is capable of solving rectangular networks at a rate of 4 million synapses per second and with learning at a rate of 1.3 million synaptic updates per second.

4.7. UCL Neuro-Chip

At University College London we have designed and are currently implementing in CMOS, a primitive processing element for building a parallel MIMD Neurocomputer, configured from an array of these elements [38]. The goal of the Neurocomputer is to support a range of Connectionist algorithms [4,15], spanning both neural network models and semantic network languages.

Each processing element, as shown by Figure 16, comprises three units: communications, processor and local memory. The communications units, when interconnected by their bi-directional, point-to-point connections, support a logical bus structure for routing message packets. Each processing element has a neuron name, used for message routing.

Message:

```
16 bit  16 bit  16 bit
neuron  dendrite  value
```

Neuron:

```
In
Communication Name Out
Out
```

IP: Processor
AX: Primitive ALU
Memory
4K x 16 bit words

Figure 16: UCL MIMD "Neural" Processing Element

The processor consists of a primitive ALU, supporting ADD, SUB, AND, XOR etc.; two visible registers, an instruction pointer IP and an accumulator AX; and 16 instructions. All data, addresses and instructions are 16 bits. (Note a processor with only 2 instructions could have been built, but this would have increased program sizes and hence the local memory required.)
The memory is 4Kx16-bit words, although the instruction set allows a larger address space.

The Neurocomputer is configured by loading a simple program into each element; this code can be identical or different for each element. During operation, messages are sent from element to element. Each message (see Figure 16) consists of: the neuron name (defining the destination processing element), the dendrite (defining the input link) and the value. When a message arrives at an element, an interrupt is generated and the message is processed by the neuron-like element.

This investigation of MIMD Neurocomputers is still at an early stage, and we expect to design and fabricate a series of progressively simpler neural processing elements during the course of the project.

5. NEW TECHNOLOGIES

Advances in technology have always constituted a major driving force for computer development. Three technologies that could have a big impact on future computers are: in the short term, Gallium Arsenide (GaAs); in the medium term, Optical devices; and in the long term, Molecular devices.

5.1. Overview

Research into new technologies such as Optical devices, has expanded rapidly in recent years. In addition, many researchers into these new technologies look to Neurocomputing to provide parallel architecture to utilise these novel devices [1,23].

GaAs Technology [35] has made rapid progress in recent years particularly in the area of digital chip complexity. When comparing GaAs with silicon, its two main advantages are higher switching speed and greater resistance to adverse environmental conditions. However, GaAs is inferior to silicon in terms of cost (of material and lower yield) and transistor count (related to yield and power consumption). For Neurocomputers, packing density (i.e. miniaturisation) of PEs would seem to be more important than switching speed. Thus currently, GaAs does not seem to provide any major benefits compared to Silicon for Neurocomputers.

Optical techniques for information processing have made rapid advances in recent years. Within this area, the term Optical computing is defined [5,9] as: the use of optical systems to perform computations on one-dimensional or multi-dimensional data that are generally not images. The goal of this work is to build an Optical binary digital computer which uses photons as the primary information carrying medium rather than electrons. The potential advantages of optical computers include: (i) high space-bandwidth and time-bandwidth products, (ii) they are inherently two dimensional and parallel, (iii) optical signals can propagate through each other in separate channels with essentially no interaction, (iv) optical signals can interact on a subpicosecond timescale and (v) optical devices can, theoretically, be made orders of magnitude smaller than silicon devices. Thus the potential of Optical parallel computers for Neurocomputer is clear.

Finally, in the longer term Molecular computers promise an exciting research area. Although no molecular computing device seems so far to have been constructed [11], the possibility of organic switching devices and conducting polymers may come about from current developments in polymer chemistry, biotechnology, the physics of computation and computer science. Although there is no clear consensus as to the viability of molecular computing devices, the potential for collaboration with Neurocomputer research is obvious.
Below we examine research into optical and molecular devices, and their possible use in parallel architectures.

5.2. Optical

Using optical computing devices [5,37], it is theoretically possible to fabricate integrated circuits that are smaller, faster and with greatly increased density, than electronic technology.

Optical computing research is being pursued throughout the world. Major national projects include in Japan, the $70 million, six-year Optoelectronics Project; in Europe, the European Community's Joint Optical Bistability Project; and in the USA, the Optical Circuity Cooperative centered on the University of Arizona in Tucson. The largest single company commitment to Optical computing is that of AT&T Bell Laboratories.

Optical computing research divides into:

- *Optical Digital computing* - an optical supercomputer performing binary digital computation which utilise bistable or nonlinear optical devices to model electronic transistors.

- *Optical Analog computing* - an optical system performing pattern recognition computation which utilise lens for Fourier transform and convolution operations.

To build Optical digital computers the prerequisite is a three-port optical transistor, exhibiting optical bistability. Optical bistability is analogous to light sensitive sunglasses: when you look at the sun they go black (cf. off) and when you turn away they go light (cf. on). Candidate three-port optical transistors centre on two technologies, namely (i) a nonlinear Fabry-Perot interferometer and (ii) multiple quantum-well material.

Turning now to Optical analog computers, here the base technology is the spatial light modulator. A spatial light modulator, in general, modulates the light output as a function of the light intensity input. The device consists of the spatial light modulator, the detector, and three beams: write, readout and output. The amplitude (or phase) of the "write" beam is modulated as a function of the intensity of a controlling "write" beam, and the reflected product of this two-dimensional information pattern is the "output" beam.

The architecture of Optical computers relate closely to the properties of optical technology. Optical computers are naturally parallel and can support global communications from arrays of transmitters to arrays of receivers at the speed of light. Parallelism in Optical computing means the ability to perform a large number of operations simultaneously but independently, such as switching all the optical logic gates in an entire two-dimensional array.

Consider the three major functional units of a computer, namely memory, processor and input/output. In a classic von Neumann electronic computer the processor must access the memory (or input/output) sequentially. This form of execution is "single-instruction-single-data stream" (SISD). With an Optical computer, in contrast, the three units can access each other simultaneously in parallel. This form of parallelism is "single-instruction-multiple-data stream" (SIMD), as supported by traditional array processors. As shown in Figure 17, an Optical computer should be implementable as a large optical gate array with the three units being indistinguishable, and global communications being provided by a separate unit such as a computer generated hologram.
5.3. Molecular

Molecular computers are intended to be information processing systems made of proteins, and other large molecules, where these molecules sense, transform and output signals.

Molecular computing has its origins in the early 1970s when biological information processing models were first developed. Since that time, rapid progress has been made in the underlying biotechnology infrastructure required to develop a Molecular computer, namely biosensors, protein engineering, recombinant DNA technology, polymer chemistry, and artificial membranes [11]. This research has culminated in major research programmes, such as the Japanese Government’s eight year, $65 million, project under the auspices of the Research and Development Association for Future Electronic Devices.

The building blocks of a Molecular computer are proteins and enzymes. A protein is a large molecule comprising smaller molecules called amino acids organised as a linear chain. A chain might typically contain 300 amino acids, chosen from among 20 commonly occurring types. An enzyme is a protein module that supports the pattern recognition. An enzyme is responsible for recognising a "messenger" module (referred to as the substrate) and causing it to change to a "product" module. Each enzyme recognises a specific type of messenger molecule by its geometric shape and transforms it (i.e. switches its state) by making or breaking a precisely selected chemical bond.

The basis of this pattern recognition is the folding of protein chains. A protein assumes a shape when numerous weak interactions amongst the amino acids in the linear chain cause it to fold into an elaborate three-dimensional shape. Clearly, the number of different protein shapes is potentially enormous. In addition, the recognition process, involving the enzyme matching with the messenger molecule is itself sensitive to interaction with other molecules and local physicochemical conditions. This interaction is important, particularly for Molecular computing, because an enzyme can also be switched into a different shape, thus allowing for memory and control at the molecular level.

A Molecular architecture might comprise three layers of molecules [11] performing input, processing and output. This is illustrated by Figure 18, based on [11].

Input (i.e. receptor) molecules in layer one transform the input signals into "messenger" (i.e. the substrate) molecules released inside the tactilizing medium. These sensory inputs might be light, temperature or pressure. Processing molecules (i.e. tactil-
izing enzymes) interact with the messenger molecules, transforming them and causing a reaction-diffusion pattern of activity. Lastly, the output molecules (i.e. readout enzymes) read the local messenger molecules that result from the reaction, and generate the output signals from the computer.

6. FUTURE TRENDS

In the search for the "correct" parallel architecture for Neurocomputers the desire for versatility (i.e. programmability) must be balanced against both the hardware complexity and the computational power. In Figure 4 these trade-offs are presented pictorially. For Neurocomputers the potential complexity of PEs range from RAM cells to microcomputers like INMOS' Transputer. The near-neighbours of Neurocomputers are the special-purpose hardware nets and Cellular Arrays.

To date neural network models and applications have typically been developed through simulations on conventional computers, such as the DEC VAX. Due to severe performance constraints, these software simulations are being transferred to parallel computers [17], such as the Intel iPSC Hypercube and Inmos Transputer-based systems.

Stimulated by these neural network software advances, have been the recent developments of neurocomputer hardware components. These developments, as we have seen above, subdivide into: special-purpose, analog implementations of specific models, typically the Hopfield model; and more general-purpose Neurocomputers, such as the HNC ANZA and the TI NETSIM.

With regard to the future, Neurocomputers are believed to represent a fundamentally new Pattern Processing domain of computation, complementary to the Symbol Processing domain of traditional computers. This, we believe, recommends the development of a general-purpose parallel architecture for Neurocomputers, whether analog or digital.

In conclusion, over the next, say, twenty years Neurocomputers can be expected to evolve through the following stages:

- Design of novel hardware components
- Production of Electrical Neurocomputers
- Design of components for Optical Neurocomputers
- Production of Hybrid Electro-optical Neurocomputers
- Design of components for Molecular Neurocomputers
- Production of Optical Neurocomputers

Readers wishing to examine further the developments in Neurocomputer architectures and hardware components should look at two extensive industrial surveys [30,34], as well as a number of good overview articles appearing in the general press [7,8,32]

ACKNOWLEDGEMENTS

In preparing this survey paper I have been greatly assisted by my colleagues in the Neurocomputing Research Group at University College London. I would particularly like to thank Matthew Lee, Marco Pacheco, Siri Bavan, Marley Vellasoo and Steve Britton.

REFERENCES


1. Introduction

In programming a scientific or an engineering problem for a conventional uni-processor computer, both the hardware and the programming language require the problem to be cast into sequential form for its solution. In a real sense then, programming such problems on multi-processor computers liberates the programmer from this sequential straight-jacket and can allow the natural parallelism of the particular problem to be straightforwardly exploited. Indeed, most real-life scientific and engineering problems naturally decompose into many subtasks that can be performed concurrently, but whether or not such a decomposition will run efficiently on a particular piece of parallel hardware is a much more open question. Moreover, although the problem may have a natural parallelism, the programmer needs an appropriate programming language to allow such concurrency to be exploited easily and safely. Transputer-based machines, with their low-latency communications and their fast process switching, together with the occam programming language providing a mathematically sound parallel system language, can go a long way towards making MIMD concurrency a commercially viable reality. Before discussing the use of such transputer-based machines in any detail, it is worthwhile restating the reasons for pursuing parallelism in scientific and engineering computing.

Setting aside for the moment very real questions about the provision of high-level programming languages and intelligent compilers, it is clear that parallel hardware can offer a dramatic reduction in cost per Megaflops. In what follows, we shall often use Megaflops - millions of floating-point operations per second - as a standard computer performance 'unit', although it must always be borne in mind that manufacturers' quoted Megaflops ratings may often quite fairly be described as "the performance that cannot be exceeded" on their machine, rather than an "average" rate on a "typical" problem. Nevertheless, if Megaflops ratings are contentious for vector supercomputers, for the more exotic parallel architectures we are considering, they are even more so. Many other parameters, such as interprocessor bandwidth and memory access time, are relevant, and furthermore, the performance will be very dependent on the particular problem, the particular algorithm and the connectivity of the multi-processor computer.

The potential benefits of a parallel approach to problems are not only the reduction in cost per Megaflops, but also the fact that only with massively parallel systems will we be able to achieve a total computational throughput far in excess of that achievable using conventional vector supercomputers. What sort of problems require or can benefit from this scale of computing
As an example of the non-deterministic nature of multiprocessor programs consider the following situation. We have the simple 3-processor system indicated in Fig 2.5. Processor 3 is programmed to receive inputs from processors 1 and 2 and assign the first input to a variable called "a" and the second to variable "b". Processor 1 sends the value "100", say, and processor 2 the value "-1". Since each processor is computing independently on different data, it may not be possible to determine in advance which result will arrive first and thus which of the two possibilities "a=100, b=-1" and "a=-1, b=100" is selected. Such non-determinacy is an inherent property of multiprocessor machines of this type. One needs to be aware of such potential problems and, if necessary, program around them.

The most common problem, however, encountered by parallel programmers is undoubtedly that of deadlock. This is a situation in which each processor ends up waiting on an input from another processor so that the whole system hangs up. Techniques for deadlock avoidance and correct termination of parallel programs soon become a standard part of the parallel programmers' armoury. An oft-quoted example illustrating deadlock is that of Dijkstra's Dining Philosophers. The essence of this example is as follows. A famous Oxbridge college employs five fellows who are required only to philosophize. However, everyone, even philosophers, need to eat and so the college maintains a dining room with a circular table, five plates, five forks and a bowl of spaghetti in the middle of the table which is kept perpetually filled by the kitchen staff. When a philosopher feels hungry, he (or she) takes a seat at the table and, because of a design flaw in the system, has to use two forks to lift spaghetti from the bowl in the centre to his plate. He then puts one fork down and eats with the other. This example illustrates both non-determinism and deadlock. If all five philosophers sit down together and all pick up one fork there is no fork free for any one of them to lift spaghetti from the central resource to fill his local resource (his plate). Unless they can agree that one of them should put down a fork, thus allowing another to pick up spaghetti and eat, they can all starve. There are many possible ways to ensure that this deadlocked situation cannot occur; for example, by the introduction of a butler who prevents more than four philosophers from sitting down at once. (One also has to ensure that the butler is "fair" and does not victimize one particular unfortunate philosopher whom he never allows to sit down and eat!) It should be noticed that this example contains the possibility of deadlock but is non-deterministic in that, over any given period of time, deadlock is not guaranteed to occur.

Although the practical parallel programmer soon develops an awareness of these problems and develops methods to avoid the more obvious pitfalls[2,3] one of the significant advantages of multi-transputer systems lies in the clear formal semantics of the occam programming language[4,5,6] Occam, based as it is on Hoare's concurrency model of Communicating Sequential Processes [7] allows the possibility of mathematical reasoning about the behaviour of concurrent programs. As programs and systems become more complex, and, in certain applications, reliability issues more critical, this feature of occam and transputer arrays is likely to become of increasing importance.
3. Transputers and Occam

On a single VLSI chip the Inmos "T800" transputer provides processing power, memory and communication hardware. The T800 has two processors, one a 32 bit, 10 Mip CPU and the other a floating-point co-processor capable of 1.5 Megaflops performance. The on-chip memory consists of 4Kbytes of fast, 50ns static RAM, and the communication hardware comprises four fast 20 Mbit/sec serial links. Both processors and all four links (each in two directions) can operate concurrently. The transputer hardware makes it easy to construct large and powerful MIMD (Multiple Instruction Multiple Data) arrays of transputers; just two wires per link are needed to provide bidirectional, point-to-point communication between transputers and no additional buffering is required.

In conjunction with the development of the transputer family of microprocessors, INMOS have also developed the "occam" programming language which enables an application to be described as a collection of processes which operate concurrently and communicate through "channels". An occam process describes the behaviour of one component of the implementation and each channel provides a one-way connection between these components. If the two processes are on different computers, the transfer of a value from one end of the channel to the other is only allowed when both processors are ready. The occam protocol thus enforces synchronization between the communicating computers. We shall have more to say about the occam language in the next section. Here we wish to concentrate on aspects of the transputer processor design which resulted from the need to implement the occam model of concurrent processes and process interaction in a straightforward and efficient way.

Instead of the relatively large number of registers and the correspondingly complex instruction set of conventional microprocessors, the transputer exploits the availability of fast-on-chip memory by having only a small number of registers and a reduced instruction set. For sequential processing only six registers are used:

- The workspace pointer which points to the area of store where local variables are kept.
- The instruction pointer which points to the next instruction to be executed.
- The operand register which is used in the formation of instruction operands.
- Three registers which form an evaluation stack and are the sources and destinations for most arithmetic and logical operations. Expressions are evaluated on this stack and instructions refer to this implicitly. The choice of a three register stack was arrived at after gathering statistics from a large number of programs on how to achieve an effective balance between code compactness and implementation complexity. The compiler ensures that no more than three values are loaded on to the stack.
The instruction set was designed for simple and efficient compilation and contains a relatively small number of instructions, all with the same format, and chosen to give a compact representation of the operations most frequently occurring in programs. Each instruction consists of a single byte divided into two 4 bit parts. The four most significant bits are a function code and the remaining four bits are a data value. This representation provides for 16 functions, each with a data value ranging from 0 to 15. Thirteen of these are used to encode the most important functions performed by any computer. Examples of single byte instruction are: load/add constant, load/store local, jump/conditional jump. Two more of the function codes allow the operand of any instruction to be extended in length: prefix/negative prefix. The remaining function code, 'operate', causes its operand to be interpreted as an operation on the values held in the evaluation stack. Thus up to 16 operations can be encoded in a single byte: the prefix instructions can also be used to extend the operand of an 'operate' instruction. The encoding of these 'indirect' functions is chosen so that the most frequently occurring operations are represented without the use of such a prefix instruction. These include arithmetic, logical and comparison operations such as 'add', 'exclusive or', and 'greater than'. Measurements show that about 80% of executed instructions are encoded in a single byte. Many of these instructions, such as 'load constant' and 'add' require only one processor cycle, 50 ns on currently available transputers.

The processor also provides efficient support for the occam model of concurrency and communication. There is a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing processor time. In order to make the run-time overhead for concurrent processes very small, the occam compiler can establish the amount of space needed for execution of each component at compile-time. The processor does not therefore need to support dynamic allocation of storage. Moreover, process switch times are also very small as very little state need to be saved: it is not necessary to save the evaluation stack on rescheduling. The scheduler operates in such a way that inactive processes do not consume processor time. The processor provides a number of special operations to support the process model - e.g. start process, end process - and also a number of operations to support message passing - e.g. input message, output message.

The key question for a user, however, is whether or not such powerful distributed memory arrays of processors can be easily programmed. In fact, concurrently with the design of the transputer, Inmos also developed a programming language called occam[4]. This language embodies Hoare's communicating process model of concurrency[7] and incorporates communication primitives and concurrency ab initio. Moreover, the features present in the occam language represent the result of an elegant engineering compromise between the desirability of a given language construct and its ease of implementation in silicon. The transputer is therefore engineered not only to execute the occam language primitives efficiently but also to support both simulated concurrency on a single processor as well as a truly distributed implementation on a network of transputers.
An occam program consists of "communicating sequential processes". Processes are themselves sequential but can be run in parallel with other processes. Communication between these concurrently operating processes is achieved by point-to-point 'channels'. There are three primitive processes in occam:

\[
\begin{align*}
&v := e \text{ assign expression } e \text{ to variable } v \\
&c! e \text{ output expression } e \text{ to channel } c \\
&c? v \text{ input variable } v \text{ from channel } c
\end{align*}
\]

The novel feature of occam is that the programmer can specify whether processes are to be executed sequentially or in parallel. This is done with the two declarations.

\[
\begin{align*}
&\text{SEQ} \quad \text{sequential execution} \\
&\text{PAR} \quad \text{parallel execution}
\end{align*}
\]

With the SEQ or PAR constructs, conventional sequential programs can be constructed in the usual way using variables, assignments, mathematical and logical expressions, and conventional constructs such as IF, WHILE and FOR.

The conventional IF construct makes a choice according to the state of some variables: the alternative construct 'ALT' in occam makes a choice according to the state of channels. At its most basic an ALT watches all available input channels and executes the first process that becomes ready. It is here that the non-deterministic nature of multiprocessor programs arises. If two inputs arrive simultaneously, the machine will take only one of them, and which one is not specified by the program. One of the possible advantages of occam, however, over more complex concurrent languages, such as Ada, is its very clean formal semantics. These allow the possibility of program proving and program transformation, and in the future, could lead to the generation of useful software tools for concurrent occam programs.

Early versions of occam did not, for example, support floating point variables but the language has now been extended, and the specification of "occam 2" has just been 'frozen' by INMOS[4]. Details are probably best obtained direct from INMOS although several books are now available. Here we shall only make some simple points. An example program is also given as an appendix.

In occam, an application program is decomposed into a collection of sub-programs - "processes" - that can execute either sequentially or in parallel. For example:

\[
\begin{align*}
&\text{SEQ} \\
&P1 \\
&P2
\end{align*}
\]

means execute process P2 after process P1 is finished. By contrast, the program fragment

\[
\begin{align*}
&\text{PAR} \\
&P1 \\
&P2
\end{align*}
\]

instructs the program to execute P1 and P2 concurrently.
The occam process model is illustrated in Figure 3.2. The three sequential processes P1, P2 and P3 can all execute in parallel and communicate with each other via one-way communication "channels". Notice that this model of concurrency is very different from that embodied, for example, in the Ada language or in shared memory multiprocessor machines. Here, there is no shared memory and variables can only be passed between concurrently executing processes via channels. This has the advantage of avoiding contention problems and ensuring a secure and side-effect-free multiprocessor "system" language.

In the occam model of concurrency, parallel processes exchanging information are obliged to engage simultaneously in the act of communication, regardless of which process is sending or receiving. Synchronization between the two processes is thus enforced with communication: the data transfer from one end of a channel to the other can only happen when both processes are ready. If one process sends data to another process which has not yet reached the communication point in its code, the transputer implementation of occam automatically suspends the sending process until the receiving process signals that it is ready to receive data. Similarly, if one process reaches a point at which some input is required from a second process which is not yet ready to send, then the receiving process is suspended till the data is received.

In contrast to approaches to parallel programming in which all the parallelism is left implicit for the compiler to extract (if it can), occam requires the programmer to make the parallelism entirely explicit. Thus, for example, the three processes of Figure 3.2 may be run on one transputer or divided between two or three transputers as indicated in Figure 3.3. The choice between implementing the multi-process code on two or three transputers may be dictated by issues such as load balancing, communication bandwidth or even simple economics!

4. Practical Methodologies

In developing code for execution on transputer arrays, one can identify some basic principles. For each application, it is clearly important to identify all the opportunities for parallelism at the design stage of the program. The type of parallelism selected for implementation will then, subject to the constraint of no more than four links per transputer, dictate the configuration network for the transputer links. A feature of the occam language is the great freedom it allows the programmer in utilising different forms of concurrency. In the applications we have studied, we have found it useful to distinguish between three common broad classes of parallelism. We refer to these three types of parallelism as "Processor Farm", "Geometric" and "Algorithmic". All three may be implemented on reconfigurable transputer architectures. We shall briefly elaborate on each type before proceeding to the discussion of the individual applications in the next section.

(i) Processor Farm Parallelism

Many scientific problems require repeated execution of the same program, or some subsection of a program, with different
initial data (random number seeds, for example). Later runs of
the program do not require any knowledge of previous runs, so
many runs could be done simultaneously. On most computers this
option is not available, resulting, typically, in the submission
of many different jobs consisting of the same program but
accessing different data, or running with different parameters.
By contrast, this type of application can be run very efficiently
on a multi-processor machine. Little or no communication is
required between processors, except that, after execution, the
results from each of the processors need to be collated and,
perhaps, some kind of statistical analysis performed.

A similar situation occurs when a 'controller' issues work-
packets to a network of processors, without caring which
processor accepts it. The only real difference is one of scale.
This "farm" structure will automatically balance the load between
the workers, because a worker which accepts a 'difficult' packet
will not accept another until it has finished, whilst a worker
which had an 'easy' packet can take another relatively soon.

Typical architectures for these types of application are
thus "farms" of processors reporting back to, and receiving
instructions from, a controller. The work can be distributed
down a linear chain (Figure 4.1) with a simple control structure,
or on a ternary tree (Figure 4.2) with a more complex control
structure but faster broadcasts. Each processor runs the same
program (with data dependent branches) and has a complete, but
different, set of data from its workpacket. Large amounts of
storage may therefore be required on each element. Because of
the limited communication requirements this method can be
efficient, but because of the memory requirements it is not
necessarily cost-effective.

(ii) Geometric Parallelism

Many physical problems have an underlying regular
gEometrical structure, with spatially limited interactions (e.g.
problems in field theory or hydrodynamics). This homogeneity
allows the data to be distributed uniformly across the processor
array, with each processor being responsible for a defined
spatial area. This is illustrated in Figure 4.3.

Processors communicate with neighbouring processors and the
communication load will be proportional to the size of the
boundary of the subdomain, while the calculational load will be
proportional to the volume of the subdomain. This type of
parallelism is sometimes referred to as "domain decomposition" or
"data parallelism". It is this type of decomposition which Fox
and co-workers used to get the excellent (over 80%) efficiencies
on the "Cosmic Cube" machines[8]. The original prototype CalTech
machines were able to show the exciting possibilities of this
type of architecture but were limited by the lack of a suitably
powerful VLSI chip to handle the communication and processing.
The transputer provides us with not only a fast 32 bit processor
and floating-point co-processor but also has the singular
ability to overlap communication and calculation. Thus large
transputer arrays are easily able to match and better these
Cosmic Cube efficiencies.
(iii) Algorithmic Parallelism

This is a more fine-grained parallelism in which features of the algorithm that are capable of concurrent operation are identified and each processor executes a small part of the total algorithm. Clearly, the resulting structure will be specific to the particular algorithm used in the application. This type of parallelism can be expressed naturally in occam on transputer networks.

A common feature of this approach is the construction of a number of "pipes" of processors, similar to those found in pipelined vector supercomputers. Here, however, the pipes may be more general and capable of splitting and merging in much more flexible way and operate at a different level of granularity.

In such a decomposition of the problem, the data now flows between the processing elements, and is sometimes referred to as "Data Flow" parallelism (not to be confused with the machine of the same name). The communication load on each processor is severely increased in this scheme. Indeed, without care, communication bandwidth problems can become dominant and severely degrade the performance. In addition, an elaborate communication and control structure is needed. An advantage of this type of decomposition, however, is that little data space is required per processor, and in many of the problems that we have investigated all, or almost all, of the elements of the network need no memory other than the internal memory (4K at present) of each transputer. We have found that efficiencies of 50% are typical (without much effort) but that detailed analysis and load-balancing can usually improve the efficiency significantly.

(iv) Hybrid Methods

The two distribution techniques described above distribute either the data or the code but not both. In the former case the communication overheads grow as the individual data areas shrink, eventually becoming dominant. In the latter case the communication load remains fixed as the code is further subdivided, but the computational work supported by that communication is reduced and again communication will eventually dominate. In both cases there is a limit to the number of processors that can be utilised efficiently. The solution to this dilemma is to incorporate both features into the distribution scheme. This will allow many more processors to be used before the communications limit is reached. Since we have a hierarchical process model this is easily done: a typical structure (Figure 4.4) might have a farms of random number generators feeding an array of algorithmic pipes and this entire structure is then replicated in a regular geometric lattice.

In the next section we shall discuss issues of efficiency and performance and analyse some examples of each type of parallelism.

5. Performance and Efficiency

In order to program multiprocessor machines efficiently it is necessary to have some knowledge of both the underlying
architecture and the basic hardware parameters. In this
discussion we shall restrict ourselves to distributed memory MIMD
computers although some of the issues raised have more general
validity. For such an N-processor machine it is convenient to
define the speedup or efficiency as follows

\[ E = \frac{\text{time to compute problem on one node}}{N \times \text{time to compute problem on N nodes}} \]

For conventional multiprocessor machines such as CalTech's
"Cosmic Cube" and the other hypercube machines, this efficiency
formula can be rewritten in the form [8]

\[ E = \frac{T_{calc}}{T_{calc} + T_{comm}} \]

where \( T_{calc} \) = total calculation time

and \( T_{comm} \) = total control and communication time

The inefficiency is thus seen to be introduced by the
additional control and communication involved in distributing the
problem over the N-processing nodes.

Transputer arrays fall into this category of machine with
the important proviso that the transputer hardware allows
communication to take place concurrently with computation. Thus,
with only a relatively small increase in code complexity, part of
the "wasted" communication time, in which a conventional
processor would normally not be able to get on with useful
computation, can be overlapped with useful computation. Thus,
for transputer-based multiprocessor machines we may write

\[ T_{comm} = T_{setup} + T_{overlap} \]

where \( T_{setup} \) comprises non-overlappable channel set-up and other
overheads, while \( T_{overlap} \) consists of communication time that can
be overlapped with calculation. Thus, for transputer arrays, we
expect to be able to achieve higher efficiencies since now

\[ E = \frac{T_{calc}}{T_{setup} + \max (T_{calc}, T_{overlap})} \]

In order to examine the validity of this (somewhat
simplified) analysis let us look at an explicit example. We
shall also use this and other examples to illustrate the
application of the three programming paradigms of geometric,
algorithmic and processor farm parallelism.

We begin by considering a typical grid problem - Laplace's
equation in two dimensions. Our analysis will focus on
distribution techniques using a simple relaxation method rather
than on a search for the most efficient parallel algorithm. We
wish to solve the 2-dimensional Laplace equation

\[ \nabla^2 \phi = \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \phi(x, y) = 0 \]
in a square region with fixed non-zero boundary potentials. The Laplacian may be approximated using finite differences on a uniform grid, leading to the Gauss-Jacobi relaxation algorithm. In an obvious notation the updated field at the grid point \((n, m)\) is given by

\[
\phi (n, m) = \frac{1}{4} \left( \phi (n+1, m) + \phi (n-1, m) + \phi (n, m+1) + \phi (n, m-1) \right)
\]

corresponding to the update "stencil" shown in Figure 5.1. We now consider both geometric and algorithmic implementations of this algorithm.

(i) Geometric parallelism

Each of the \(N\) processors is assigned a sub-region containing \(n\) gridpoints so that the whole domain consists of \(N \times n\) points. As is evident from Fig.4.3 the total calculation in each sub-region is proportional to \(n\), the area of the region, while the communication with neighbouring processors - to obtain the necessary data to update the edge points - is proportional to \(n\). Thus we expect

\[
T_{calc} \sim n
\]

\[
T_{comm} \sim \sqrt{n}
\]

and, for large enough \(n\), high efficiency is assured since the \(n\) dependence has the form

\[
E \approx 1 - \frac{A}{\sqrt{n}}
\]

where the coefficient \(A\) is specific to the particular multiprocessor hardware. As shown by Fox and others, these arguments generalize to higher dimensions and to a surprisingly wide range of problems[8]. We are concerned here with a multi-transputer implementation of this 'domain decomposition' technique.

Consider a \(4 \times 4\) array of transputers connected as a regular 2-dimensional grid. To map \(130 \times 130\) grid with fixed boundary conditions on to this array, each processor is assigned a \(32 \times 32\) subregion. To examine the effect of overlapping communication with calculation we mimic the effect of slowing down the communication speed by communicating the data \(M\) times. Writing

\[
E = \frac{T_{calc}}{D(M)}
\]

we have, for the non-overlapped case

\[
D_1(M) = M T_{comm} + T_{calc}
\]

while, for the overlapped situation

\[
D_2(M) = M T_{setup} + \text{Max} (M T_{overlap}, T_{calc})
\]
The results are shown in Fig.5.2 and 5.3. For the non-overlapped case we see the expected linear dependence on M with slope $T_{\text{comm}}$ and intercept $T_{\text{set}}$. The actual efficiency (for the case when $M = 1$) is 92%. With overlapped communications, we see that until M is around 120 the $T_{\text{overlap}}$ term is entirely masked and the slope proportional only to $T_{\text{set}}$. After the turnover region the slope reverts to $T_{\text{comm}}$ as for the non-overlapped case. The efficiency ($M=1$) has now, as expected, increased, almost to 99%.

Whilst it is gratifying to be able to achieve such high efficiencies and validate the simple analysis outlined above, two words of caution are in order. Firstly, it may well be that for purely pragmatic reasons, such as simplicity of code and so on, it is better not to worry about extracting every last percent of speedup but rather code the problem more simply and throw more transputers at it! Secondly, if the program is written to communicate in all 8 directions simultaneously and another process also needs to access external memory, then external memory bandwidth may become a significant limiting factor.

(ii) Algorithmic parallelism

In this approach to the problem, the program must be split up into roughly equal "size" pieces. In this case there is a very simple basic algorithm which may be written as

\[
\text{Update} = (\text{Left} + \text{Right} + \text{Up} + \text{Down})/4.
\]

Dividing this up as shown in Fig.5.4 into operations on vectors, two transputers perform additions on vectors and a third an addition and a multiplication. All these transputers need little or no external memory: a fourth transputer with substantial external memory keeps the old and updated versions of the entire array. Using 4 T414 transputers and a 40 x 40 array an efficiency of about 50% was obtained[9]. Given the relatively poor load balance between the transputers this lower efficiency is to be expected. Algorithmic networks for more complex algorithms can be very complicated. Fig.4.5 shows one such network constructed by Bryan Carpenter at Southampton for a Monte Carlo simulation of a statistical mechanical spin system[10]. Such networks need not only care with load balancing but also with deadlock and termination. Efficiencies between 50 and 60% are typical.

(iii) Hybrid Parallelism

For certain applications a combination of geometric and algorithmic parallelism can make optimal use of the processing power available. This hybrid technique has been successfully used by Bryan Carpenter to code 1260 16-bit T212 transputers to solve the three-dimensional Ising ferromagnet[11]. To our knowledge, this is the largest MIMD machine ever programmed and one that can honestly be described as a 10 Gip machine - albeit with RISC instructions. The Ising model simulation can be programmed using several different algorithms so in Table 1 we compare the performance of this "B001260" machine against several other computers using the same "Metropolis algorithm". As can be seen, this transputer array, assembled out of standard components
over a few days, is faster than a special purpose machine built at Santa Barbara to solve just this one problem! Moreover, the B001260 achieves almost a third the performance of a Cyber supercomputer for a small fraction of the hardware cost. In fact, as the last column in Table 1 shows, the world best performance for the Ising problem using this algorithm is probably held by the SIMD ICL DAP machine: the binary nature of the Ising model is particularly well-suited to the single-bit processing elements of the DAP. For a more floating-point intensive problem, the DAP does not compare so well. Moreover, if the 1260 transputers had more than just the 2 Kbytes of on-chip memory and were more reconfigurable, it is probable that at least an order of magnitude improvement in performance could be achieved[11].

<table>
<thead>
<tr>
<th></th>
<th>Santa Barbara</th>
<th>B1260</th>
<th>2-pipe CYBER</th>
<th>ICL DAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed M update /s</td>
<td>25</td>
<td>27</td>
<td>93</td>
<td>218</td>
</tr>
</tbody>
</table>

Table 1. Comparative Performances for the Metropolis algorithm for 3 dimensional Ising system.

(iv) Processor Farm Parallelism

There are two basic types of processor farm - one in which the same entire code is held in each processor, which can then operate on entirely independent sets of data, and one in which independent pieces of "work" are sent to each processor by a farm controller. They can use various topologies such as the linear chain or ternary tree shown in Figs 4.1 and 4.2.

The mathematical analysis of all such farm types is very similar[12]. For example, if one wants to maximise the rate, $S_N$, at which results are obtained from the end of an $N$ processor chain, one finds, for the case that

$$T_{calc} > T_{setup}$$

there is a critical value $N_c$ for the largest useful chain. Pritchard[12] quotes the result

$$N < N_c : S_N = \frac{1}{2 \cdot T_{setup}} \left( \frac{T_{calc} - T_{setup}}{T_{calc} + T_{setup}} \right)$$

$$N > N_c : S_N = \frac{1}{T_{comm} + T_{setup}}$$

Processor farms also give the opportunity for porting certain types of existing Fortran C or Pascal programs with a minimum of effort. For example, we have ported a small 3000 line Fortran 77 program for Monte Carlo simulation of events generated
in electron-positron annihilations to run on a transputer farm. This program has been implemented on a Meiko system consisting of up to 30 transputers running in an occam farming harness[13]. This application has very limited communication requirements and a linear speedup was observed. Comparative figures for this application on a VAX 750, T414 and T800 transputer are shown in Table 2[14].

<table>
<thead>
<tr>
<th></th>
<th>VAX 750</th>
<th>T414</th>
<th>T800</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds/event</td>
<td>0.18</td>
<td>0.61/N</td>
<td>0.07/N</td>
</tr>
</tbody>
</table>

Table 2. Comparative performances for LUND Monte Carlo FORTRAN program. N is the number of transputers in the farm which varied from 2 to 30.

One final topic in this section concerns the question of reconfigurability of transputer networks. Until recently, our transputer systems at Southampton required to be hand-connected to the desired topology. This is not a trivial issue since we have observed definite performance gains in using optimized specially configured networks rather than forcing the code to fit a specific hard-wired network of transputers. However, whilst hand-connecting the required wiring diagram for a 32 processor system is merely rather tedious, connecting the processors in a thousand transputer system is extremely laborious and error-prone. With the present transputer hardware, without some specific hardware to route messages over a network, electronically reconfigurable transputer networks via a software controlled switch are highly desirable. Such a switch also allows, in principle, the possibility of dynamically reconfiguring the array during execution of the program to achieve dynamic load balancing. Problems such as shock wave formation, stress fractures and meteorological simulations may benefit from this type of flexibility.

In designing a switch for transputer networks we wish not only to have a "universal" static switch - universal in the sense that any valid transputer graph (i.e. one requiring no more than 4 links per transputer) can be realized - but also to have simple and fast algorithms to translate the desired network topology into the appropriate switch setting. A beautiful analysis of such problems in terms of Hamiltonian and Eulerian cycles has been given by Nicole, Lloyd and Ward[15], and the reader is referred to that paper for details. This analysis forms the basis for the link switch of the RTP supernode machine, designed and constructed by ESPRIT project 1085, and now being marketed by Telmat in France and Thorn-EMI in the UK.

6. Detailed Case Studies

In this section we describe the results of an investigation in methods of implementing several different types of problem on transputer arrays. Most of these studies were performed on arrays of up to 32 T414 transputers and the programs would run unchanged - but faster - on arrays of T800s. The chosen problems are the following.
(i) A "supercomputer" problem.
We report on work in implementing a problem from particle physics, namely, the solution of the putative non-linear, relativistic quantum field theory of the strong nuclear force, Quantum ChromoDynamics (usually referred to as QCD).

(ii) Dusty-deck FORTRAN.
There is a large amount of pre-existing code which it will not be possible to completely recode. Typically, in the science and engineering community, these are large (100,000 or more lines of code) FORTRAN programs. We report on how a subset of such programs can be easily ported to run in a standard OCCAM harness on multiple transputer systems.

We discuss each of these in turn.

(i) Quantum ChromoDynamics

This is not the place to describe in detail the intricacies of QCD itself. Instead, we shall discuss and compare various strategies for implementing numerical simulations of the "pure gauge sector" of QCD. This means that we shall only be concerned with simulating the interactions of the gluon vector fields - simulations including the quark matter fields are another story.

Lattice gauge theories are defined in four-dimensional space-time. In simulations, continuous space and time are approximated by a discrete four-dimensional lattice of points. At each lattice site there are complex unitary matrices associated with the "links" in each of the four directions, corresponding to a discretized version of the gauge field degrees of freedom. In a full simulation of QCD the gauge group is SU(3) - the special unitary group in 3-dimensions - whose group elements are 3 x 3 unitary matrices. We give results here for the gauge group SU(2) so that each lattice site has four 2 x 2 complex unitary matrices associated with it.

The heart of the gauge-field numerical simulation is a Monte Carlo procedure for updating the gauge field degrees of freedom at each lattice site. The lattice version of the gauge field theory is akin to a statistical mechanical system. The evaluation of the "energy" of the "old" and "new" configurations to perform a Metropolis-like test, involves much matrix multiplication of gauge field matrices both from the site in question and from neighbouring sites. More details of the mathematics of gauge field theories and of techniques for lattice simulations may be found in the book by Mike Creutz[16], who, along with Nobel prize winner Ken Wilson, was responsible for the current research activity.

Geometrically Distributed Program

For a lattice size of 16 x 8 x 8 x 8 the storage requirements for a single configuration are 524 kbytes (assuming REAL 32 arithmetic). For a geometric
decomposition this four dimensional lattice must be mapped onto a network of transputers: with only four links per transputer, a four-dimensional lattice is impossible. Three types of network have proved useful for an implementation on 16 "worker" transputers. In each case, each individual processor in the network is responsible for some small four-dimensional block of the whole lattice. The constraint that each processor has only four links to its neighbours limits one to the following possibilities:

- 4-dimensional binary hypercube. This network is restricted to precisely 16 worker processors and cannot accommodate more processors. The attractive properties of hypercubes are well known but this restriction on the number of transputers is an obvious disadvantage for larger simulations. A second disadvantage is that in some of our calculations it was desirable to single out one particular direction of the lattice, and arrange that the lattice could be traversed in this direction without having to cross a boundary between zones held on different processors. In those cases, one of the networks below was used.

- Periodic square lattice. The lattice can be partitioned on to a N by M array of processors with (say) the 0-dimension split into M-slices and the 1-direction into N-slices, with the 2- and 3-directions not split at all. For the example of a 16 x 8 x 8 x 8 lattice with M = N = 4, each processor holds a 4 x 2 x 8 x 8 sub-block of the lattice.

- Repeated binary square. The basic unit of this network is a 2 x 2 square, connected together as a ring of 4. This leaves 2 links free on each processor to connect to adjacent squares. N such squares may then be stacked "vertically" (with cyclic connectivity between top and bottom of the stack). This yields a 3-dimensional array of size N x 2 x 2 onto which the 4-dimensional lattice may be mapped. For example, a 16 x 8 x 8 x 8 lattice could be mapped onto an N = 4 net with a 4 x 4 x 4 x 8 sub-block on each processor.

Note that for an implementation on 16 processors, these three networks are actually isomorphic! The second and third networks, however, may be extended to more than 16 processors. In our 16 transputer implementation, as far as is possible the program is interchangeable between the different networks described above - changing the network merely involves changing some compile-time constants.

Algorithmic decomposition

In contrast to the geometric decomposition described above, for which each of the 16 transputers had a significant amount of external memory (256 kbytes), this algorithmic decomposition was written to test the
feasibility of distributing a useful algorithm over a large number of transputers with no external memory. This test program covers only the update phase of the SU(2) simulation: we do not consider measurement of correlation functions within the updated configurations. There is one further significant restriction: the program is implemented on a fixed 4 x 8 2-dimensional periodic lattice of transputers with no reconfigurability.

The total configuration is held on a local host processor and fed through the rest of the array in manageable quantities. The updating of an individual site variable requires knowledge of the variables located at several nearby sites. Thus, in this systolic decomposition, the data store processor has to transmit 19 2 x 2 complex matrices - 76 real numbers or 304 bytes of data (assuming 32 bit arithmetic) - to the remainder of the array for each update. This amount of data transfer is tolerable because of the large amount of real arithmetic (several hundred floating point operations) involved in each update, and because the transputer can do communication in parallel with other CPU activity.

A certain amount of compromise is involved in fitting the problem on to the fixed 4 x 8 array. Some of the transputers are occupied merely with routing data and are under-employed, and in several other respects, the decomposition described above is not ideal for balancing computational load between processors. Much more freedom in attaining optimal load balance would be possible with a switched reconfigurable array.

Performance

The geometric program running on 16 T414 transputers with 256 kbytes of external memory and the algorithmic decomposition for the no-external memory 32 T414 transputer array were both timed and compared with equivalent sequential occam programs run on a single transputer. The benchmarks were run with lattices of various sizes but the update times can be scaled to give the update time of a single gauge link variable.

The performance is summarized in table 3. The geometric decomposition achieves an efficiency of 96% on 16 transputers whereas the algorithmic implementation on 32 has an efficiency of 65% or a speedup of over 20. Thus, although the latter implementation is less efficient, with more transputers employed than in the geometric case, the absolute speed is faster. A rough count of the number of floating-point operations involved in an update (about 400-500) yields a floating point performance of around 1/30 Mflop for T414s with software floating-point. Much other activity besides floating-point arithmetic is also going on in the program so this should only be taken as indicative. For T800 implementations of lattice theories, sustained floating-point performance of around 1 Mflop have been achieved[17].
Update timings (ms)

<table>
<thead>
<tr>
<th></th>
<th>Single T414</th>
<th>Array of T414</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric</td>
<td>15.6</td>
<td>1.01</td>
<td>16</td>
</tr>
<tr>
<td>Algorithmic</td>
<td>13.5</td>
<td>0.65</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 3. Performance summary of QCD code.

(ii) **FORTRAN Farms**

A FORTRAN farm harness has the following general features. The sequential version of the FORTRAN code should have the following general template:

**Sequential algorithm template**

Read initial data (from file/keyboard)
Open input and output files
LOOP **INDEPENDENTLY** OVER INPUT DATA PACKETS
    Read an input data packet (from file)
    Perform task on (part of) the input data
    Write output from this task (to file)
Close input and output files

Here the term 'packet' refers to that subset of the data required for one iteration of the ensuing loop.

The important feature which makes it possible to farm such a problem is that the iterations in the loop over input packets are independent, so that in principle one could perform them in any order, or indeed simultaneously. The workers in the farm perform the work of the interior of the loop, with read and write stages replaced by packet communications, whilst the farmer processor handles assembly and despatch of the packets and receives the returning results.

A sensible partitioning of the problem into farmer and worker tasks is therefore:

**Worker**

SEND input data request
LOOP UNTIL end-of-data received
    RECEIVE input from the farmer
    IF (input packet)
        Perform calculation on the input data
        SEND output data packet to farmer
    IF (end-of-data signal)
        Exit loop
Farmer

Open input and output files
LOOP UNTIL Terminate signal received
  RECEIVE input from farm
  IF (input request)
    IF (more data available)
      read data into input packet from host file
      SEND input data packet to network
    ELSE
      SEND end-of-data signal to network
  IF (output data)
    Write data from output packet to host file
  IF (Terminate signal)
    Close files
  Exit loop

It is clearly necessary to run extra processes to handle the problem of connecting up SEND steps on one processor with RECEIVE steps on some other processor. This is the job handled by the farm harness.

The important feature of this structure (in both worker and farmer) is that, apart from some setting up actions before entering the loop, each FORTRAN process is a slave to its input which is sent to it by some other processor. (In this formulation the farm is 'kick-started' by the worker processes which send off input requests before entering the main loop). The routing strategy required for this system is very simple since farmer-worker communications and the worker-farmer calculations can be handled separately. For a linear farming network the resulting process structure is shown in Figure 6.1.

As shown in the figure, any messengers leaving the farmer (upper channels) must pass through one of the workers before it can be returned to the farmer. In order to prevent deadlock, the farmer will only send a message if it knows that there is a worker ready to receive it. Thus the communication processes can always output to a worker main-body process and will never become clogged up with messages waiting for the destination worker to finish some other work.

In order to achieve this self-regulation it is necessary that the farmer process only composes and sends input packets when either a signal requesting a packet is received from a worker or an output packet is received from a worker. Since the workers will not be sent input packets until a request reaches the farmer, buffering for a spare packet is provided at each worker. Thus an extra input-request must be sent by the worker to ensure that this buffer is filled.

In the farm harness developed by Mike Surridge at Southampton [18] each message is passed to the sending FORTRAN subroutine from the user-written FORTRAN process together with a 3-word header array and a further integer giving the length of the message. The first word
contains a message-type code, signifying input or output requests etc. The second word of the header gives a processor ID value indicating the source or destination of the message. The third word is used for file-protocol communications (produced by the FORTRAN run-time library routines) and is redundant in the messages passed directly to and from user-written processes.

In addition to data packet passing, it is useful for the harness to permit any FORTRAN process in the network to send commands back and forth to the Alien-File-Server. However, since the FORTRAN process is forced to halt while conducting any exchange of this type, overuse of this feature will severely impact on the efficiency of the farm and is intended mainly to allow diagnostic output to be obtained during debugging of the user code. Some other features for monitoring and timing processes are also desirable in the farm harness.

Two straightforward tests of harness performance have been made, one using a simple bench-test program and one using an existing sequential FORTRAN program from an industrial application. The bench test showed that the harness can be very efficient and an efficiency of 99% was obtained for a 4 worker farm falling to 96% for 16 workers. It is interesting to note that there is an optimum number of packets into which a given amount of work should be divided to achieve maximum efficiency. The general form of the results of a numerical experiment is indicated in Figure 6.2. The shortest time is obtained with between 5 and 10 packets per worker which is the result of a trade-off between lower communications overheads with less packets and better load-balancing with more packets.

The second test of the farm harness was an actual user code for the evaluation of electric fields using a Monte Carlo random-walk method described by J.H. Pickles [19]. The user code for this experiment was provided by the CEGB. The farm program was produced by writing a shell corresponding to the templates above and inserting calls to subroutines produced by snipping code from the original sequential program. About 65% of the pair of programs produced (for workers and farmer) was taken directly from the original source code. In performance trials, the farm showed a speed-up almost linear with the number of workers (up to 16).

The High Energy Physics community is a major consumer of computing power and their projected requirements for future experiment at the new accelerators, LEP in Geneva and HERA in Hamburg, currently being constructed are rather alarming. They have recently begun to examine the potential for vectorizing their code but this requires considerable programming effort and has so far only yielded comparatively modest performance improvements[20].

The development of increasingly complex detectors for high energy particle physics experiments, culminating in
those for LEP and HERA, has brought with it the necessity to process ever larger volumes of data. For instance, the data acquisition system for the ALEPH experiment at LEP is expected to read out 100-200 kbyte events at a rate of around 1Hz, each of which will require about 30s (IBM 168 seconds, roughly equivalent to VAX 8600 seconds) to process. Generating sufficient Monte Carlo events to compare with this experimental data is an even more daunting task. The ALEPH detector simulation program, GALEPH, can take up to 300s to perform a full event simulation.

Faced with such severe demands for processing power, physicists have been forced to consider more novel, and cost-effective, architectures than the traditional mini and mainframe computers. One architecture which is particularly well suited to event simulation and analysis is that of the processor farm.

In this context, the processor farm consists of a set of processors running identical code, completely independently, but each operating on different data. Typically, there is a single stream of input data 'packets', which are distributed to any processor which is ready for more data. Output packets are merged into a single output stream on a first come first served basis (figure 1). This scheme is clearly ideal for the processing of high energy physics events, since they are all independent of one another. It should be noted that the order of input packets is not in general preserved in this simple scheme, although it is possible for that to be arranged, if desired, at the cost of a small loss in performance.

The INMOS transputer represents a natural building block for various multiprocessor systems, of which the processor farm is one instance, because of the presence of on-chip communication links. A practical realisation of a transputer based processor farm is shown in figure 2. The farm elements (workers) are arranged in a linear chain, connected at one end to a 'farmer' processor, which interfaces to a host system, such as a micro VAX, which provides access to peripherals. Each farm element runs two processes concurrently, one of which performs the task which has been farmed out, while the other is responsible for routing input packets to the first available worker, and output packets back to the host. On the face of it, it might be thought that, say, a tree would perform better than a linear chain, but apart from having less latency when starting and stopping, this is not the case. It has been shown[12] that, in the steady state, a linear chain is optimal in that the rate at which packets are processed grows essentially linearly with the number of processors until the link bandwidth of the farmer is saturated. In the case of the T800 transputer, this is at around 2 Mbytes/s, which is comparable to the bandwidth of a good tape drive.

The farmer and routing processes together constitute what is known as a 'harness'. The application programmer need
not know about its workings - it is merely linked with the application program. Farm harnesses for Meiko transputer systems have been available for some time, and a harness which is compatible with the latest INMOS software has been developed at Southampton[18], funded by the SERC/DTI transputer initiative.

GALEPH is the ALEPH experiment’s detector simulation program, and is coded in Fortran 77. Figure 6.3 represents a simplified picture of the flow of data between program modules, and their relation to the main library packages (shaded grey).

GEANT 3 builds the detector geometry from information stored in a detector description file. Particle four-vectors from an event generator, such as the Lund Monte Carlo, are tracked, and the subdetector responses and trigger are simulated. Events may optionally be output to a mass storage device, using EP10. Memory management is performed by the ZEBRA package, within GEANT3, and by the BOS package within the general framework of GALEPH.

The implementation of GALEPH required the porting of elements of the CERN program library (CERNLIB) and, in fact, this constituted the major part of the work. The library consists of a nucleus of basic utility and numerical routines, known collectively as KERNLIB, together with other, more specialised, routines and packages.

Two kinds of problem were encountered in porting the code - non-standard language features and machine dependencies. The bulk of the machine dependency was contained in KERNLIB. In particular, bit, byte and character handling routines had to be written. This task was made relatively simple by the support in Meiko Fortran for non-standard bit manipulation routines, although bit formats differed in some cases. Furthermore, many CERNLIB routines make use of the LCCF function in KERNLIB, which returns the address of a variable. An equivalent function (IADDRESS) was kindly provided by Meiko[21] specifically for this purpose. Most of the remaining work involved modification to code relying on implementation specific use of character variables. The syntax of hexadecimal constants, which do not form part of ANSI standard Fortran, also required modification.

The demonstration hardware used consisted of a Meiko M10 crate, containing an MK014 local host board and an MK021 mass store board. The MK021 was populated with 8 Mbytes of memory and a 20 MHz T414 transputer. The host system was an IBM PC clone, containing an MK026 link adapter card. The size of the program meant that an 8Mbyte board was needed for the linking stage (the executable image is some 3.5 Mbytes). We note in passing that the current version of the linker is one-pass, requiring either careful ordering of routines presented to it, or multiple inclusions of the same code.
We have successfully ported KERNLIB, HBOOK, ZEBRA and GEANT3 from the CERN program library. The BOS package has also been implemented, although references to EPI0 are currently dummyed. These libraries were used to run the GALEPH program on the MK021 board. The same program was also run on a VAX 8200 for comparison. (The VAX 8600 is approximately 4 times faster than the 8200). The timings obtained are shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>T414</th>
<th>VAX 8200</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds/event</td>
<td>209.0</td>
<td>29.0</td>
</tr>
</tbody>
</table>

Table 4. Performance summary for GALEPH FORTRAN farm: N is the number of transputers.

Lack of time precluded running the program on a T800 transputer. However, the number of floating point operations in the code leads us to expect a speedup by a factor of between 6 and 10. This may be compared with a value of 8.7 previously obtained for the Lund Monte Carlo generator[13], run independently from GALEPH.

In summary, therefore, we have run a large high energy physics Fortran program on a transputer system. In doing so, we have implemented several extensively used libraries. Although the program has not yet been run on more than one transputer, this step is a straightforward extension, as our previous experience with transputer based processor Farms[13] has shown. Benchmark timings show that the T800 transputer has a processing power which is comparable to that of a VAX 8200, indicating that a T800 based processor farm would represent an extremely cost-effective source of compute power for high energy physics simulation. This program of work was carried out by Andrew Carter and Ian Glendinning with assistance from Jim Cownie of Meiko and Mick Storr of CERN. The above account is taken from Ref. 21.

7. Conclusions

What should be clear from the foregoing sections is that transputer arrays already constitute a practical and cost-effective solution to a wide range of problems. Indeed, for users willing to rethink and recode their problems, an array of T800 transputers gives access to a new scale of local computing power. Instead of having to bid for supercomputer time and access a national facility over a network, the user can obtain more computational throughput from a modest sized transputer array consisting of 100 or 200 T800s which can be running his (or her) problem 24 hours a day, 7 days a week. Thus for universities, research institutions and special-purpose applications transputer systems offer almost unrivalled cheap, expandable computing power.
Appendix

Consider the example finding the square root of a number using Newton’s method. To find the square root of Q we consider the function:

\[ f(x) = x^2 - Q \]

and use Newton’s method to improve on an estimate of the position of the zeros. We obtain after \( n \) iterations

\[ x_{n+1} = (x_n + Q/x_n). \]

In occam we could code this sequentially as follows:

```occam
REAL32 x, Estimate
SEQ
  Sq.root?x -- input initial value
  Estimate: = x/2 -- form initial estimate
SEQ i = 0 FOR n
  Estimate:= (Estimate + (x/Estimate))/2
SEQ Sq.root.result! Estimate -- output result
```

However, in occam we also have the option to perform iterations in parallel and form a pipeline for processing many square roots. This could be programmed as follows:

```occam
CHAN values [n + 1]:
PAR
  REAL32 x:
SEQ
  Sq.root?x -- input initial value
  values [0]!x -- output to 1st stage of pipe
  values [0]!x/2
SEQ i = 0 FOR n
  REAL32 x, Estimate
SEQ
  values [i ]?x
  values [i ]? Estimate
  values [i + 1]!x
  values [i + 1]! (Estimate + (x/Estimate))/2
SEQ
REAL32 root, any:
SEQ
  Values [n]? any
  Values [n]? root -- receive final estimate
  Sq.root.result!root -- output result
```

The above code will execute on a single transputer but is now clearly in a form enabling it to be run on a number of transputers. This example is, in fact, somewhat artificial since the ratio of computation to communications overheads is extremely low in this algorithm. Nevertheless, the fact that inter-transputer communication is independent of the processor operation shows there is potential for a significant speed-up in performance and in fact, on more realistic examples, the relevant algorithm may usually be distributed to yield a speed-up proportional to the number of transputers.

151
References


14. J. Cowrie (Meiko Ltd) private communication.


17. J. Hoek, "SU(3) Lattice Gauge Theory Simulation on T800 Transputer Arrays" to be published in the proceedings of the International Conference on the Impact of Digital Microelectronics and Microprocessors on Particle Physics, Trieste 91988), and B. Carpenter, private communication.


22. L. Valiant, "General Purpose Parallel Architectures", to be published in the Handbook of Theoretical Computer Science.

2.1 Fox’s wall.

2.2 Pipelined solution to Fox’s wall.

2.3 Geometric solution to Fox’s wall.

2.4 Farm solution to Fox’s wall.
2.5 Nondeterminacy example for multi-processor system.

3.1 The T800 transputer: layout of main features.

3.2 The occam process model: sequential processes.

P1, P2 and P3 and point to point communication channels.
3.3 Simulated concurrency or true concurrency: possible distributions of processes P1, P2 and P3 on (a) one, (b) two and (c) three transputers.
4.1 Farming network: linear chain.

4.2 Farming network: ternary tree.

4.3 Geometric parallelism: 12 x 12 data array distributed over 2 x 2 processor array. Laplace update stencil highlighted.
4.4 Hybrid structure.

\[ \bullet \bullet \bullet = \frac{1}{4} \bullet \bullet \bullet \]

5.1 Update stencil for Laplace's equation.
5.2 Laplace results: non-overlapped communications.

5.3 Laplace results: overlapped communications.
5.4 Algorithmic network for Laplace problem.

6.1 Worker-farmer process structure for a linear farm network.
6.2 Time \( T \) to complete a fixed amount of work versus number of packets \( N \).

6.3 A simplified picture of the flow of data between program modules in GALEPH, and their relation to the main library packages (shaded grey).
HIGH-SPEED NETWORKS

A. Danthine

Institut d'Electricité Montefiore, Université de Liège, Liège, Belgium

Introduction

A few years ago, Bob Metcalfe has tried to characterize the evolution of the computer systems by the following statements:
- the Sixties have been based on a computer in each company;
- the Seventies have seen a computer in each building of a company;
- the Eighties will be remembered by a computer in each office of a company.

It is during the Seventies that the networks have been perceived as the only way to interconnect the mainframes located in each building of a company and to allow the remote access to a computer from a terminal located either in the same building or in a remote location.

It is also during this decade that the communication domains have been clearly identified
- communication inside the computer system with distances between 1 and 100 meters;
- wide area network (WAN) with distances from 10 to more than 10,000 km;
- local area network (LAN) allowing communication inside a building or an industrial or an academic site, i.e. between 100 m up to 10 km.

But at the end of the Seventies, two technologies were available to cover the three domains:
- the system bus with its parallel transmission was providing between 10 and 200 Mbps;
- the WANs were relying on the PTT switched and leased lines with the packet switching networks at the end of the decade. The range of transmission rate was then between 300 bps and 56 or 64 kbps.

From the figure 1, it is clear that a technology was missing and the local area was forced to use, in the Seventies, the technology associated with the WANs.

![Diagram](image)

**Fig 1. Situation at the end of the Seventies**
At the beginning of the Eighties, with the success of the PC, the trend was for a stand-alone computer in each office of a company but it was soon recognized that a computer system of an enterprise can not be based on a set of isolated computers. Distributed system does not mean atomised system. The need for interconnection was then recognized but unfortunately, the technology of the WANs was not adequate to interconnect all the equipments of the local area.

Then Ethernet came..., and soon behind, IEEE 802.3, 802.4 and 802.5, not to mention ARCNET, the Cambridge Ring, STARLAN, Apple Talk, ...

With transmission rate between 100 kbps and 10 Mbps, we have the missing block and the figure 2 is reflecting the situation around 1985.

![Diagram showing transmission rate vs distance in km]

**Fig. 2. Situation in 1985**

This new environment is favorable to the goals of the end of the Eighties: "a networked workstation in each office of a company".

Today, in 1989, the overall trend is for an increase of the transmission rate in all the three domains:
- inside computer we have architectures based on fast channels and supercomputers are offering up to 800 Mbps;
- for the WANs, the 64 kbps of the ISDN look already oldfashioned with respect to the announced 2 Mbps of the B-ISDN;
- for the LANs, with FDDI and several Esprit Projects such as BWN and LION, we are today between 100 and 140 Mbps. These LANs are also envisaged as MANs (Metropolitan Area Network), the distance covered going now up to 50 km.

It seems therefore that the future is rosy as we will have faster and faster networks. But recently, there was some questioning about the need of such an evolution: "we do not need high speed local area network because our workstations operate the same way on a network at 1 Mbps than on a network at 10 Mbps".

We will first try to understand why it is so today and later on we will discuss about tomorrow.
About Rates in LANs

The network transmission rate is the one which is the most often mentionned, 10 Mbps for IEEE 802.3 or 100 Mbps for FDDI.

In many cases, the network signaling rate is higher: 20 Mbps for IEEE 802.3, 125 Mbps for FDDI, but of interest only for the engineering team having to deal with the hardware and the physics of the medium.

However, the network transmission rate is not of direct interest for the user due to the overhead associated with a network protocol. The service provided by the network may be better evaluated by the network data rate which may be defined by the rate at which the network can transfer the data submitted at the MAC access point.

![Network Data Rate for IEEE 802.3](image)

Fig. 3. *Network Data Rate for IEEE 802.3*

For Ethernet, assuming no collision, the network data rate is function of the packet data field length and is reproduced at figure 3. It is clear that it is not interesting to use Ethernet for packet with less than 100 data bytes and this explains the poor performances of Ethernet for terminal access to character-oriented time-sharing systems.

![Network Data Rate for the Cambridge Ring](image)

Fig. 4. *Network Data Rate for the Cambridge Ring*
For the Cambridge Ring, which has also a network transmission rate of 10 Mbps, the network data rate is function of the ring length (fig. 4) and much smaller due to the ratio of the overhead to the user data.

The network data rate is important for the provider of service. But it is not of direct interest for the user which is interested in the service access rate as seen by an individual user.

This individual service access rate is the MAC access rate, i.e. the maximum access rate available to a user, assuming no interference with other service users, by taking into account the constraints of the MAC protocol. For Ethernet, with the assumption of no interference with other users, the MAC access rate is equal to the network data rate, but for the Cambridge Ring, the MAC access rate, reproduced at figure 5, is drastically reduced with respect to the network data rate reproduced at the figure 4.

![MAC access rate (Mbps)](image)

**Fig. 5. MAC Access Rate for the Cambridge Ring**

It is clear that the ratio (MAC Access Rate) to (Network Transmission Rate) is one of the basic characteristic of a network. A small value of this ratio does not necessarily mean a bad network but it reflects a certain type of sharing which may or may not be suitable for a given application.

To conclude this section, it is interesting to notice that the value of this ratio may depend upon some parameters of the network or of the user. The figure 6 gives the evolution of this ratio for the IEEE 802.5 as a function of the ring length and with the data field length of a packet as parameter.

In the figure 6 the length of the network is expressed in bits. The length of the network in kilometer depends upon the network transmission rate and is also reported on the same figure for three values: 4, 16 and 100 Mbps. An increase of the network transmission raise the issue of the way the network behaves with small frames.
MAC access rate / network transmission rate

- 1500 bytes
- 500 bytes
- 100 bytes
- 10 bytes

Ring length: 0, 500, 1000, 1500, 2000 bits

If network transmission rate = 4 Mbps

- 0 km: 22.5
- 500 km: 47.5
- 1000 km: 72.5
- 1500 km: 97.5

If network transmission rate = 16 Mbps

- 0 km: 5.5
- 500 km: 12
- 1000 km: 18
- 1500 km: 25

If network transmission rate = 100 Mbps

- 0 km: 0.9
- 500 km: 1.9
- 1000 km: 2.9
- 1500 km: 3.9

Fig. 6. Ratio of MAC Access Rate to Network Transmission Rate for the IEEE 802.5

Measurements on Systems

The previous discussion has pointed out that there is an upperbound limit to the MAC access rate and this limit is not always close to the network transmission rate.

However this MAC Access Rate is a theoretical limit and the user is interested at the performance of its workstation and of the associated servers.

The direct interest of the users lies in the performance seen from the application layer, involving not only the lower layers but also the protocol suite. However we will not discuss here this aspect but concentrate on the performance achieved at layer two.

We will assume that the access of the station to the network is achieved through a hardware board (CC : Communication Controller) implementing at least the real time part of the MAC protocol. This board is plugged into the system bus and driven by a device driver which may be considered as the lower level service interface seen by the user processes.
The amount of resources which is located on the communication controller is an important characteristic which may have a direct influence on the performance.

Let us define the **station access rate** as the effective access rate which can be measured for a given station. We will distinguish the driver level and, for some controllers, the MAC level when this interface of service is accessible for measurement on the communication controller.

The station access rate at the MAC level depends upon the way the MAC protocol is implemented and in particular, upon the way the communication controller is able to use available access right. The station access rate at the driver level depends upon the way the station interact with its communication controller via the system bus.

The detailed discussion about the performance testing methodology may be found in [Marso,88].

To discuss these issues we will concentrate on Ethernet for which products have been available since 1981 and where we already are at the third generation of communication controllers.

**The First Generation of Ethernet Controllers**

This is today almost archeology but still of interest for understanding the interplay between communication resources and the station architecture.

The controller we will discuss here is a Q-bus Ethernet Controller where the MAC protocol was partially implemented in hardware and partially in software with 32 kB of dual port memory on board. The figure 7 indicates that the station access rate (driver level) is more than five times smaller than the MAC access rate of figure 3.

![Station access rate (emission)](image)

**Fig. 7.** Ethernet Controller - First Generation - Q-bus Station

Station Access Rate, Driver Level, in Emission

To achieve this result, the CPU was fully dedicated to the communication controller and it is interesting to notice that queued I/O primitives (PTIP = 1 to 3) do not bring a significative improvement.
The situation of figure 8 concerns the same controller but in reception. The situation is worst than in emission and it must be understood that, if the 32 kB dual port memory is able to absorb a small peak of packets coming at a higher rate than the average curve of figure 8, packets will be lost in the communication controller if the average rate of incoming packets is above the curve of figure 8.

Fig. 8. Ethernet Controller - First Generation - Q-bus Station
Station Access Rate, Driver Level, in Reception

The Second Generation of Ethernet Controllers

The second generation of Ethernet Controller is characterized by a processor on the communication controller and by the complete implementation of the MAC protocol on the board in discrete chips, the VLSI chip sets being not yet available at that time. The processor is not programmable by the user and is managed by firmware.

Fig. 9. Ethernet Controller - Second Generation - Q-bus station
Station Access Rate, Driver Level, in Emission
The figure 9 reproduces the station access rate (driver level) in emission as well as the CPU usage.

The comparison with figure 7 will be deceiving for those who believe that, by definition, a coprocessor will improve the throughput of the system.

From the curve of figure 9, it is clear that CPU usage is now lower than 100% and this is the only positive result of the coprocessor.

To understand this result, one must remember that onboard processors are managed by firmware and that firmware may create too much overhead if it is not carefully designed and implemented. It is therefore not uncommon at all that performances with onboard processor are lower than performances without it. This is not only true for Q-bus but also PC Ethernet controllers.

The station access rate in the reception case is reproduced at figure 10, and it is interesting to notice that the performance of this communication controller is better in reception than in emission.

![Station access rate (reception)](image)

**Fig. 10. Ethernet Controller - Second Generation - Q-bus Station**  
**Station Access Rate, Driver Level, in Reception**

However, controllers from the first and the second generation have been successfully used despite their limitations because the requirements of the station access rate what at that time (1981-1985) bounded by the capacity of the systems themselves. In [Rake, 84], the maximum value of the throughput between an LSI 11/23 and a VAX 11/750 was measured at 0.86 Mbps for memory to memory and at 0.49 Mbps for disk to disk.

**The Third Generation of Ethernet Controllers**

The third generation of Ethernet controllers may be characterized by:
- 802.3 chip set
- more than 100 kB of onboard memory
- an onboard processor, user programmable.

The two communication controllers we have tested were VME-bus controllers. One was using the chip set from Intel (82586) with a 80160 processor. The other was based on the chip set from AMD (LANCE AM 7990) with a 68000 processor.
The user programmable characteristic of the onboard processor has allowed us to develop a test program to evaluate the station access rate at the MAC Level. This test program issues a fixed number of Transmit Requests to the MAC, according to the ASAP (As Soon As Possible) generation scheme. It should be pointed out that the same frame is transmitted at each request and therefore the test program does not have to prepare the data to be passed to the MAC.

Figure 11 shows, for the two controllers, the station access rate at the MAC level and it is interesting to notice that the two curves are close to each other but still at a fraction of the theoretical value, even in the very simplify test condition (same frame).

The conclusion is that the controllers are not able to transmit, on a free network, two successive packets with an interframe spacing close to the minimum value of 9.6 μsec. In practice, the interFrameGap is equal respectively to 560 and 435 μsec for the A-controller and for the B-controller.

During the extensive test done on the two controllers [Danthine,88a], the transfer rate of the VME-bus in programmed mode has been measured equal to 7.0 Mbps and 6.0 Mbps, respectively for the A-controller and for the B-controller. Here, DMA transfer is not possible due to the potential conflict with the DMA mode used by the MAC chip to get data from the onboard memory.

Figure 11 shows also the system access rate of the two controllers at the driver level. The difference between the two controllers results from various factors [Danthine,88c], such as:
- in the A-controller, the off-the-shelf kernel does not allow for pipelining of transmission requests, the kernel itself being responsible of a substantial packet overhead;
- in the B-controller, the user-written code for the onboard processor was designed to allow the controller processor and the MAC chip to work in parallel insofar as the accesses to the shared memory allowed it. By doing so, the processing overhead involved in a packet transmission was limited to 525 μsec.

These measurements confirm what has been said earlier: offloading the MAC chip set to a controller board and adding an onboard processor does not necessarily result in outstanding performances. It is indeed not obvious at all to design a system where the activity associated to the driver and the transfer of the packet to the onboard memory could take place in parallel and at the same pace than the activity of the MAC chip set itself. It is also clear that a good solution for a maximum size packet may not be adequate for small packets.
High performance controllers are now becoming available but it is fair to indicate that many workstations are now being designed on a single board, the MAC chip set being integrated with one of the communication channels of the board.

**Required System Access Rate**

The limitation of the station access rate is not the only factor which explains why today workstations operate almost the same way on a network at 1 Mbps than on a network at 10 Mbps. We also have to take into account the way the upper network layers have been implemented. But, rather than discussing in details this performance limitation, let us try to discuss the required system access rate at the application level.

The basic rule is to have a system access rate in direct relationship with the volume of data to be transferred. If, for a diskless workstation, an access rate around 0.5 Mbps may be acceptable, for a workstation with a disk, 3 to 4 Mbps seems more appropriate.

Is it reasonable to require such a high access rate for a station which will require network access less often than the diskless one? We believe so, and we want to stress that the sharing of the network for accessing a file server must ideally be done on a basis of one at the times, in order to get the best of the disk access rate of the server. Any multiplexing of accesses to several files will result in important degradations of the server throughput.

**A Corporate Communication System**

It is clear today that a corporate communication system will rely heavily on LAN technology. From all the foreseeable trends, it is obvious also that within a corporation the coexistence of heterogeneous LANs will be the rule and not the exception. In many corporations, this set of heterogeneous LANs is not the future but the present situation. Some of these LANs have been installed to interconnect existing equipments and the others have been introduced by the manufacturers of newly acquired equipments. All manufacturers have today selected at least one type of LAN which is now an integral part of their system architecture.

However, if heterogeneous LANs are already the rule in many enterprises, very few can claim to have succeeded in their complete integration. The very concept of an entreprise communication system will require all these LANs to communicate with each other.

Such a corporate communication system will heavily rely on the concept of distributed processing and will be successful only if its architecture matches the structure of the organisation. It is important to recognize that the penetration of the LANs is closely related to subdivisions of the organisation and the corporate communication system will have to integrate the existing situations without trying to dictate a common and unique solution for all problems faced by the different parts of the corporation.

How is it possible to build a corporate communication system from a set of heterogeneous LANs? One possible answer is an hierarchy of networks.

This hierarchy of networks may be based on 3 levels characterized by the following categories of LANs:
- capillary LANs;
- medium speed LANs;
- high speed LANs.

The capillary LAN is generally homogeneous in equipments connected and is characterized by well-defined applications. It implies very often limited distance between equipments, bandwidth between 100 kbps to 10 Mbps, and is generally based on a non-OSI architecture. The existence of a gateway to medium speed LANs is a requirement for corporate
integration. Apple Talk and STARLAN are two examples of capillary networks. Thin Ethernet with Novell and 3-Com software are other examples.

The medium speed LANs are the fully standardized ones (IEEE 802.3, 802.4, 802.5). They are usable on distances exceeding the kilometer and offering network transmission rates between 4 and 16 Mbps. They support more heterogeneous applications and are characterized by a more standard approach in terms of the upper layers of network architecture. They follow either the OSI model or a de facto standard such as UNIX with TCP/IP. These medium speed LANs have bridges which may be used to separate the loads, gateways for interconnection with other medium speed LANs.

High speed LANs (HSLAN) are offering network transmission rate above 50 Mbps and are usable on distances exceeding 25 km. There exist today several possibilities of utilisation of this newly available technology.

The first utilisation of HSLAN is for back-end networks, i.e. networks used for the interconnection of equipments in a computer center. There is indeed a need for such a back-end HSLAN in order to update what has been available for many years; the Hyperchannel at 50 Mbps.

Another possible utilisation of HSLAN is as front-end network. Here the workstations have direct access to the network. If we agree with the idea that, with a high speed LAN, a system access rate at the MAC level will have to be above 10 Mbps and if we remember the previous discussion about the difficulties to achieve a very high throughput from the station through a bus structure, this type of utilisation is not very likely to happen soon. We have also to take into account that the cost of a high speed LAN attachment is much more higher than the cost of a medium speed LAN attachment.

The most likely utilisation of the high speed LAN will be as a backbone network allowing the interconnection of all medium speed LANs of a corporate communication system. The capillary LANs will be themselves connected to the medium speed LANs.

The backbone network may not be necessary in a company involving a limited amount of employees located on a small site but will be a must for a corporation located on a broad site. By broad site, we mean a campus with its partially organized distribution of buildings, an industrial site with its multiple workshops and plants or even a single huge building where hundreds or even thousands of employees are located. Today, few systems, if any, allow an efficient communication in terms of cost and performances within such a broad site.

The high speed networks will not only be used in the local area but also in the metropolitan area. Taking into account the strong trends to the B-ISDN, no MAN will be able to succeed if it is not at the very beginning integrated into the B-ISDN project.

**High Speed LAN Design Principles**

In this section, we would like to address the following question: is the design of a high speed LAN different from the design of a medium speed LAN?

We know that any LAN is characterized by:
- a transmission medium;
- a coding and a modulation;
- a topology;
- an access method.

For HSLAN, the only transmission medium which is considered is the optical fiber.

For the coding, taking into account the range of speed, it is not possible to use Manchester coding, which is perfect in term of number of transitions and of mean value of the coded
signal but requires a signaling rate which is twice the value of the transmission rate. Today the trend is for the 4/5 or 8/10 coding which introduces only an increase of 25% for the signaling rate. The digital signal is transmitted baseband.

For HSLAN, the only topology which fulfills the requirements is the ring topology [Danthine, 85]. Single ring and double rings have been proposed.

Coming finally to the access method, the only usable one is the token passing. And the questions to answer in order to define this access method are the followings:
- how long is the token retained by the station?
- when is the token released by the station?
- is there a priority mechanism?

We know that the token may theoretically be retained by the station for one frame, for a limited number of frames, for a maximum time or for all waiting frames. But if we decide to retain the token for more than one frame, it implies that we have a way to make back-to-back transmission feasible. The performance of the Ethernet controllers for this problem was not very convincing and it will be of interest to evaluate the performance of implementations of HSLAN where multiple frames are transmitted with each token acquisition.

Taking into account the domain of parameters associated with a HSLAN, a priority scheme like the one of 802.5 is not, in our opinion, feasible. The release of the token must take place immediately after the last frame is transmitted. It is interesting to notice that, for the recent announcement of the 16 Mbps token Ring, IBM introduced the idea of an early release of the token.

It is interesting to observe how the OSI Reference Model has been adapted to the new technology. With the initial OSI Reference Model, we had a Physical Layer and the Data Link Layer. The LAN has introduced a splitting of the layer 2 into two sub-layers, one dealing with the Logical Link Control and one related to the Medium Access Control. It was also agreed that the Physical Layer Protocol is tidely associated with its corresponding Medium Access Control.

With the standardisation of HSLAN, another splitting of another layer occurred. Now it is a layer 1 which is splitted into two sub-layers, the Physical Layer Protocol (PHY) and the Physical Medium Dependent (PMD).

To conclude this presentation, we would like to introduce the basic ideas behind an Esprit Project which allows a research and development in the area of HSLAN.

**The Esprit Project 73**

The aim of Esprit Project 73 is to solve the problem of the interconnection of medium speed heterogeneous LANs on a broad site. Its goal is to build a complete communication system based on a backbone network called BWN (Backbone Wideband Network) and on highly efficient gateways between this BWN and several kinds of medium speed LANs and some wideband public services offered by the PTT. Building a prototype in order to test such a system was part of the project.

This Esprit Project entitled Broad Site Local Wideband Communication System started in September 1983 and involved the following partners: *University of Liège, ACEC, Bell Telephone and ETB* from Belgium, Stollmann from Germany, *SG2* and *DNAC* from France, and *University of Athens* from Greece.

We will summarize in the following sections, the detailed presentations which will be found in [Danthine, 88a & 88b].
**The BWN Design**

In order to be able to interconnect LANs in any broad site environment, the backbone network must be able to work over distances of more than 25 km.

It has been decided, at the beginning of the project, to set the required performance of the gateways between any LAN and the BWN at an average throughput of 2 Mbps full-duplex. If, at least, 25 gateways have to be supported, it is clear that our backbone network must be wideband.

Fiber optic has been selected as transmission medium. A network signalling rate of 167 Mbps has been adopted which, taking into account the 8/10 coding, leads to a network transmission rate of 134 Mbps [Vyncke, 85a], [Durvaux, 85 &86].

The BWN is a fiber optic ring that uses long distance communication technologies: graded index glass fibre, laser transmitters and APD receivers. APD receivers are preferred to PIN-FETs for their higher dynamic range that can be required by various ring topologies. In the same way, LED transmitters can be used instead of LASERs when distance and topology do not require a high optical power.

![Functional Diagram of the BWN Physical Layer](image)

**Fig. 12** Functional Diagram of the BWN Physical Layer

174
The nodes are plesiochronous so that no network master clock is required. Every node has an elasticity buffer on the receiver side that allows to compensate for the clocks phase drifts (fig. 12). The chip implementing the elasticity buffer has been designed by SDM (an ACEC subsidiary) and manufactured by Philips.

The adopted access method [Danthine, 85, 86a, 86b & 86c] [Vyncke, 85b & 85c] is based on a token but without any priority mechanism in such a way that the release of the token takes place as soon as the station owning the token has completed the transmission of a single frame. The MAC layer offers a transmission service for data blocks of maximum 2 kbytes.

In the BWN design, a particular attention has been devoted to the system management [Vyncke, 85a & 85b][Hauzeur, 85c, 87a & 87b][Danthine[86d, 86e & 87]]. The Network Control Center, currently under development, is based on a microVax under VMS.

![Diagram](image)

**Fig. 13 The MAC interface**

The access to the BWN MAC layer is performed through an interface which has been highly specialized [Pesch, 87]. A precise specification of this interface has been elaborated since several partners had to build their own controller to access the BWN MAC layer. The hardware and the interface protocol used have been carefully designed to optimise the throughput. That BWN-MAC interface has proved very successful in the integration of the BWN-CC (Backbone Wideband Network Communication Controller) developed by the different partners with the MAC developed by ACEC. Integration was effective in less than a few hours and the measurements of throughput [Danthine, 88a] will be summarized in a following section.

**Network Architecture**

Figure 14 sketches a typical backbone topology. Many subnetworks can be identified: 802.3/Ethernet, token buses, token rings, and the BWN (Backbone Wideband Network) which is the central subnetwork. These subnetworks all together form the global network. The third interconnection level, namely capillary LANs, is not represented in this figure.
Fig. 14. The Global Network and its Subnetworks

The purpose of the internet gateways, sometimes called routers, is to pass packets from one subnetwork to the next, while dealing with the different subnetwork controllers and communication services provided by the particular subnetworks. In our case, all subnetworks are 802.x LANs with also a point-to-point 2 Mbps between Liège and Antwerp in connection with the multimedia experiment. Figure 15 sketches a simplified OSI model [ISO, 81] of the network architecture for two sample hosts located on an Ethernet and a token ring respectively.

Fig. 15. Network Architecture

So, the main role of the internet layer is to provide its users with a uniform communication service, spanning throughout the global network while hiding from the users the number and characteristics of subnetworks (LANs) and gateways that packets cross.

The internet layer plays a key role and may be the only common element of all the connected computers. On the one hand, various LANs are used. On the other hand, heterogeneous applications may be supported by different higher layer protocol architectures using the same internet service.
A Connectionless Internet Service

There are two main classes of communication services: connection-oriented services and connectionless services. The internet layer which has been specified for the Esprit Project 73 is connectionless, a choice tied to the high speed and high reliability of the BWN environment and to the required performances.

The advent of local area networks characterized by very low error rates has made the connectionless approach very attractive, the main role of a connectionless internet layer being to route packets towards destination in a simple and efficient manner.

When implemented in a LAN environment characterized by a low error rate, the connectionless communication service offered by the internet layer is suitable to most applications. Wherever a very high quality of service is needed, it is provided by the higher layers of the ISO Reference Model (fig. 15).

The first version of the Esprit Project 73 internet layer is based on IDP, the internet protocol of XNS [Xerox, 81], a de facto standard second only to TCP/IP. Our choice came from the very efficient routing table update protocol associated with XNS. However, we are committed in the long term to the ISO standards. An intermediate solution consisting in the ISO connectionless internetworking protocol, associated with the routing management of XNS has recently been specified [Hauzeur, 87c] and is currently being implemented.

The ESTELLE language has been used for the specification of the internet layer of the XNS, of the internet related protocols and of the internet management [Hauzeur, 85a,b&c], and more recently for the intermediate ISO version [Hauzeur, 87c].

The LOTOS language has been used to verify the internet service from the specification of the internet protocol and of the underlying service [Leduc, 86a & 86b].

The Internet Gateways

For the project, gateways to three different LANs had to be developed: CSMA/CD (IEEE 802.3/Ethernet), Token Passing Bus (IEEE 802.4) and Token Passing Ring (IEEE 802.5). Gateways also had to be developed to connect to the public communication services. The 2Mbps HDB3 point-to-point bearer service has been selected. It will allow to connect either a cluster of remote LANs or another BWN to the present BWN. All these gateways have to fulfil the requirements of 2 Mbps full-duplex average throughput (i.e. 4 Mbps of total traffic).

The main task within the gateways is the routing of datagrams from the BWN to the connected LANs and vice versa, in accordance with the internet protocol. Stollmann's gateway architecture is based on the assumption that data flows in each direction are asynchronous. Moreover, handling communication through each connected subnet is a rather independent process. Therefore a parallel architecture seemed appropriate for the construction of the gateway: each of the two main dataflow directions (LAN to BWN, BWN to LAN) is handled by an independent processor board which executes the internet protocol. In addition there is an intelligent I/O-board for each attached subnet. Other processes like internet management tasks and gateway internal functions are less time critical and evenly distributed on both CPU-boards.

The VMEbus has been selected as system bus. It offers a wide range of CPU-boards, it is fast enough to support gateway requirements, and it was expected also to offer a choice of controllers large enough for the LANs considered in the project.

The BWN to Ethernet Gateway is built with two CPU-boards, an Ethernet controller and a BWN-communication controller (BWN-CC), all based on 68000 processors (fig.16). The BWN-CC has been developed by Stollmann. It is a fast and DMA-oriented VMEbus board.
that interfaces to the BWN access controller (BWN-MAC) through the "standard" BWN-MAC interface introduced earlier.

![Gateway Architecture](image)

**Fig. 16 Gateway Architecture**

Apart from the development of the BWN communication controller, the main effort in developing the gateway was the optimization of the gateway software. A central role is being played by kernel which has been especially designed for the gateway as any time-consuming overhead must be avoided. It has the purpose to handle inter-process communication through simple queues and to provide facilities to manage a pool of fixed length buffers which contain the packets and a fixed header of system information. Scheduling of processes is only based on I/O events and internal exchanges, there is no round robin scheme, neither priority.

A mechanical layout of the structure of a complete LAN-to-BWN gateway is represented on figure 17.

![Internet Gateway Mechanical Layout](image)

**Fig. 17 Internet Gateway Mechanical Layout**
The Wideband Public Service

Public services capabilities have to be adapted to the capacities of our corporate system and it is therefore a big asset for our system to be connected to the wideband 2 Mbps terrestrial bearer service that is made available by some national PTT administrations [Barri, 86]. The most interesting applications for this bearer service are a 2 Mbps high speed link towards faraway sites.

The high speed link provides the possibility to build multi-site corporate communication system. Since the bearer service only covers the physical layer according to Recommendation G. 703 of CCITT, a layer 2 has been implemented to improve the reliability of the link [De Smet, 86]. This implementation took place in the public service gateway based on a SM90 [De Prycker, 86], [De Smet, 87].

A Multi-media BWN

One of the trends in networking is to have multi-media capabilities i.e. to have the communication capabilities for the data, the voice and the image. It is outside the scope of this paper to discuss in-depth the rationale behind this trend where very often the need of an integrated service is confused with the need of an integrated infrastructure.

Some multi-media solutions are based on a TDM system which privileges synchronous traffic and uses, for packet communication, the part of the bandwidth not allocated to circuits.

For the BWN, the voice transmission was not a goal but the transmission of images for video conference was part of the project. The video conferencing facility is based on two video Codecs that transform an analog video signal into a 2 Mbps compressed digital signal and vice-versa. To cross the BWN, the 2 Mbps compressed digital signal has to be packetized [De Prycker, 85]. It may look surprizing to see that a synchronized digital signal is first packetized and then sent on a connectionless network such as the BWN. It is clear that the inter-packet time will not remain constant after travelling through the BWN. Simulations have shown [Danthine, 86a & 86b] that distribution of the inter-packet time will not prevent the reconstruction of the synchronized bit stream from the received packet stream using a special protocol [De Prycker, 87]. The special hardware needed packetisation, resynchronisation and depacketisation has been integrated in the public service gateway.

Since it is possible to connect the video codecs to the wideband 2 Mbps bearer service, video conferences will be possible not only between nodes of the BWN but also between remote sites. The right part of figure 18 shows a local video codec connected to the BWN via the Bell Telephone gateway. The left part shows the remote video codec connected to the BWN through a public 2 Mbps link and the public service gateway. This configuration has been used between Antwerp and Liège to evaluate the system with several levels of data traffic on the BWN. These experiments took place in June 1988 and have demonstrated the ability to carry the synchronous signal of the video system using the techniques explained in this section even when the data load on the BWN was very high. Through artificial generation of background traffic, it has been possible to go above 100 Mbps of background traffic without any problem on the video transfer.

This system was also demonstrated at the exhibition associated to the Esprit Conference 1988 in November 1988.
BWN Access Rate at MAC Level

The same methodology for testing LAN controllers has been applied to BWN controllers, at the standard BWN-MAC interface (Figures 13 & 16).

The throughput through the interface was measured with a special hardware-based BWN-CC, built by ULg in order to be able to measure throughputs as high as 30Mbps. Results are summarized in figure 19 [Constantinidis, 87]. The figure shows that the interface reaches 11.5 Mbps in full duplex (i.e. 23Mbps of total traffic).

![Figure 19: Throughput at the BWN-MAC Interface](image-url)
BWN Gateway Performances

The equipments which have been developed to assess the performance of the various parts of the BWN system are fully described in [Danthine, 88a].

![Gateway Test Results](image)

**Fig. 20. Gateway Test Results**

The figure 20 summarizes the measurements done to evaluate the performances of the gateway at the internet service level. Let us stress that these curves demonstrate that the project goal of 2 Mbps full duplex (i.e. 4 Mbps of total traffic) has been reached.

Conclusion

The Esprit Project 73 implies the installation of a field size BWN and the assessment of its performances. Today, more than 15 km of fiber have been laid out at the Sart Tilman Campus of the ULg which will allow the installation of 13 nodes, (i.e. 13 BWN-MAC) with prepared location for an additional 12. It is expected that the complete integration of the BWN-MAC in the Sart Tilman site will be completed in April 1989 and will be fully tested before the end of July.

This network will also be the support for the development of a corporate message handling system based on the X400 series of protocols [Danthine, 88c]. This corporate message handling system (CMHS) will be built around the concept of departemental MTAs communicating through a corporate reliable transfer service (CRTS). This CRTS will access directly the network service and replace altogether the RTS, the session and the transport layers which have to be implemented for MTA communication in the public domain. This CMHS will allow us to investigate the problem of the distribution of the UA functionality between a workstation and a CMHS server with an MTA entity.

It is expected that such a wideband system when fully available, by the new opportunities it offers, will influence the characteristics of the traffic and deeply modify the today view about the corporate communication system.

References

[Barri, 86]

[Constantinidis, 87]

[Danthine, 85]
[Danthine, 86a]

[Danthine, 86b]
DANTHINE A. A Backbone Wideband Network for LAN Interconnection on a Broad Site, The International Conference on Information Network and Data Communication, Ronneby Brunn, Sweden, May 11 - 14, 1986

[Danthine, 86c]
DANTHINE A. A Backbone Wideband Network for LAN Interconnection, EFOCILAN 86 Amsterdam, June 25 - 27, 1986, pp 8

[Danthine, 86d]

[Danthine, 86e]

[Danthine, 87]

[Danthine, 88a]

[Danthine, 88b]

[Danthine, 88c]
DANTHINE A. & GODELAINE P. MHS in a Corporate Communication System offering Internet Service, IFIP WG 6.5 Working Conference on Message Handling Systems, Costa Mesa, October 10 to 12, 1988, 16 p

[De Prycker, 85]

[De Prycker, 86]

[De Prycker, 87]

[De Smet, 86]

[De Smet, 87]

[Durvaux, 85]

[Durvaux, 86]


LEDC G., A Formal Specification of the Internet Datagram Protocol of Xerox in


Authors' address
Institut d'Electricité, B28
Université de Liège
B-4000, Liège, Belgium
ABSTRACT:
This paper discusses software tools and associated methodologies for the design of digital electronic systems, focusing on the traditional types of tool that are widely available in commercial systems today, leaving aside the more recent developments in tools for synthesis of logic or layout. The areas examined are those of design capture, design verification through simulation and timing analysis, layout of boards and Application Specific Integrated Circuits (ASICs), and issues in testing with emphasis on ASICs. Some comments are given on the practical experience gained with these tools at CERN.

1. Introduction

The enormous advances in electronics of the last decades have been made possible by progress in the three inter-related domains of technology, tools and methodology. The principal driving force behind developments in electronics has been the steady advance in the technology of monolithic semi-conductor integrated circuits, which has resulted in an increase in gate count of over five orders of magnitude since the first ICs appeared. Other aspects of the improvements in technology are the dramatic reduction of manufacturing costs per gate, increased circuit speeds and enhanced reliability.

On the other hand, as system complexity has increased, design time and cost have also increased. The cost of developing and applying adequate tests to complex systems has also exploded. The stimulus of commercial competition has resulted in rapid evolution of tools, hand in hand with the development of new design methodologies, in an attempt to handle the ever increasing complexity of design and test. Our discussion of design tools will therefore also describe the associated design methodologies.

1.1 Scope of application areas and terminology

Development of electronic systems involves many different activities, most of which can be assisted by software tools (see Figure 1). The application of tools to the conceptual design process is often
referred to as Computer Aided Engineering (CAE). The application of tools to the design of physical implementations (board or integrated circuit layout) is categorised under the area of Computer Aided Design (CAD), whilst tools related to manufacturing fall under Computer Aided Manufacturing (CAM). Tools and methodologies for prototype and/or production testing come under the generic term of Computer Aided Testing (CAT). Industry is making great efforts to integrate tools for factory automation and management, an area of activity covered by the umbrella term Computer Integrated Manufacturing (CIM).

1.2 The Design Process

Figure 2 shows how the typical design process starts from an abstract, high level system requirements specification, is gradually expanded through successively more detailed levels, finally ending with a complete description of how the system shall be implemented and tested. It is essentially a top-down process using the concept of hierarchy to handle design complexity. Most design methodologies simplify the design process by using a library of pre-designed and tested modules that can be inserted into the design hierarchy at an appropriate level (e.g. at board level these modules correspond to standard, mass produced components; at the IC level they correspond to so called standard cells or macro cells). When the library of components is used, the design process is a combination of top-down design (starting from the abstract system description), followed by a bottom-up phase (starting from the available components or macros).

![Diagram of the flow of tasks in a typical top-down design process.](image)
In theory therefore, design is a linear top-down, bottom-up process with possible iteration between steps in order to correct errors. The goal of the design tools is to accelerate each of these steps, to eliminate trivial mistakes by automating the processing of large volumes of detailed design data, and to detect conceptual design flaws through simulation (or other verification processes) after each design step. When these goals are met, the number of lengthy, costly design cycles is reduced. However, the architecture of modern, integrated tool sets allows a more flexible flow with data feedback from one step effecting design decisions taken in another. In this way one can more easily realise highly optimised system designs and readily explore different design strategies.

1.3 Tool Classes

The traditional human-driven design style uses the top-down/bottom-up method enhanced with analysis tools for verification of each step. However, we are now witnessing the emergence of a class of tools that automatically synthesise detailed implementations of logic or structure (layout) from high level abstract representations\(^1\) (behavioural or structural), producing implementations that are correct by construction. The design is specified in a form independent of technology or process, and can be compiled for a particular technology or process by a Silicon compiler or a Silicon assembler. Many people believe that synthesis tools, such as silicon compilers, will eventually have profound consequences on hardware design, just as high level languages have had in software engineering. The exciting area of hardware synthesis is covered in detail elsewhere in these proceedings [1].

These lectures will concentrate on tools developed for the traditional design methodologies, where the decomposition of the high level design representation into successively more detailed levels is still largely a manual process. The designer can use tools either to automate those steps that can be achieved through repetitive application of an algorithm, or he can use tools that check the correctness of manual steps by ensuring design rules have not been violated, or by verifying that the behaviour of a lower level representation is equivalent to that of a higher level representation (simulation). Different types of analysis tool can be employed for design verification after each stage of the top-down design refinement process.

Until recently, most CAE tools have been strongly typed, i.e. not applicable to both analogue and digital designs. For example, simulation of mixed analogue and digital designs has been broken down into the separate study of the analogue and digital parts using quite different tools. Now we are beginning to see the emergence of tools that can handle mixed analogue and digital designs. These lectures will concentrate on tools and methodologies for design of digital systems.

During a design project it is possible to select the "best" tool for each task, but in practice, interfacing tools from different vendors is not without problems, and requires considerable investment for the development and support of interface software. For this reason many users prefer an integrated tool set supported by a single vendor, where all tools run on the same machine, have the same style of man-machine interface, avoid library compatibility problems by all using the same component libraries, and interface to the others through a shared design database (eliminating the need for format translation steps).

1.4 Process scaling, device and interconnect performance

Integrated circuit manufacturing processes can be characterised by their technology (CMOS, NMOS, ECL, ...etc.) and the minimum feature size (or minimum line width) that can be reliably man-

\(^1\) It is interesting to note that a simple form of synthesis tool has been in widespread use for many years. Programmable logic assemblers and compilers synthesise a device fuse map from a behavioural description in the form of boolean equations, truth tables, or state diagrams.
ufactured. Leading commercial CMOS processes today use minimum line widths around 1 micron. It is interesting to see how circuit performance scales as the minimum line width is reduced by a factor \( \alpha \). Under plausible assumptions, reference [2] shows that CMOS processes scale as follows:

- Transistor switching time \( \propto 1/\alpha \)
- Transistor power dissipation \( \propto 1/\alpha^2 \)
- Number of transistors per unit area \( \propto \alpha^2 \) (or less)
- Total power dissipation per unit area \( \propto \) constant
- Device power-delay product (a standard metric of performance) \( \propto 1/\alpha^3 \)

Most chip architectures contain global interconnections that must traverse from one side of the die to the other. As a result of the very strong decrease in production yields with increasing die area, the largest die size that can be manufactured remains roughly constant at about 1 cm square. Thus large IC designs, that pack as many devices on a single die as possible will contain global wiring of the order 1 cm long. As the process dimensions are scaled down the resistance of global metal interconnect lines increases like \( \alpha^2 \) and their total capacitance remains constant (it is assumed that dimensions scale by the same factor in the vertical direction in order to avoid etching problems). The result is that global wiring delays actually increase like \( \alpha^2 \), while transistor switching times decrease like \( 1/\alpha \). Figure 3 shows that, for die sizes of the order 1cm, global wiring delays dominate over transistor delays for sub-micron processes. Today large CMOS designs, using processes with minimum line widths around 1 micron, demonstrate gate delays of the order 400 ps and global wiring delays of the order of a few nanoseconds.

Although technology improvements may reduce the interconnect delays (e.g. by replacing aluminium by a metal of lower resistivity), it seems that semiconductor process technology has now advanced to a point where some fundamental changes in design methodologies and tools begin to be necessary. These changes are not only required in order to handle the large volume of design data and the difficulties of testing complex devices. The emerging design world, in which device interconnect delays begin to dominate the intrinsic device propagation delays, will no longer be handled by a simple set of decoupled tools used in a linear sequence. Design capture, verification and layout will become closely integrated so that important layout constraints and critical parts of the circuit can be specified by the designer at the design capture stage. This information will be automatically forwarded to the layout tool, which itself will be closely integrated with the design verification tools in order that post-layout verification can be done. The linear design process will become much more iterative. In the future, logic will become "cheap" and wires "expensive", reversing the situation with which most designers are accustomed, and probably acting as a major influence on developments in systems architectures, design methods and tools.

![Diagram showing the transition from interconnect dominates to devices dominate in CMOS technology](image)

**Figure 3:** Global wiring delays dominate device delays in sub-micron processes (the diagram is typical for CMOS technologies).
As device speeds and system clock frequencies increase, similar timing problems will show up at the level of board layout. Board designers will have to pay more attention to the layout of critical wiring paths, clock skew, cross-talk, noise and other analogue-like behaviour of digital circuits. At the same time board layout is being complicated by new components with hundreds of pins, housed in new packages that require board layout techniques not easily handled by most existing tools.

2. Design Capture

Design capture is the process of entering the basic design specification in a machine-readable form. A design can be specified structurally (i.e. as a network of interconnected components), or as a behavioural specification without reference to internal structure, or by using a mixture of the two previous approaches.

Traditional engineering practice consists of building up a structural specification in the form of a schematic diagram, or netlist (a textual specification of the component types and their interconnectivity). Most CAE/CAD systems include an interactive schematic capture tool.

The behavioural form of design capture can be used to establish a formal, unambiguous "black box" specification of the functionality of a system or sub-system. It can be verified by simulation and can then be subcontracted to another designer, or a synthesis tool, to be turned into a structural description that can be built to implement the desired behaviour. Hardware description languages can also be used to specify the structure of a system algorithmically. For example, a high level procedural layout description can be executed to generate detailed IC mask layouts.

2.1 Schematic Capture

Figure 4 shows how, using an interactive graphics system, a schematic is constructed by extracting component symbols from a component library, placing them and interconnecting them with wires and

![Diagram of schematic capture and compilation tool](image)

*Figure 4:* Components of a typical schematic capture and compilation tool.
busses. Wires and busses can be assigned names, and components can be assigned certain parameters (e.g. a physical location reference parameter denoting the position of the component on a printed circuit board).

Usually the library definitions of the components include additional information such as part type, stock part number, package type, package pin numbers for each pin of the symbol, etc. This additional information is stored in the drawing file for later use by other tools. Usually the symbol library will also contain pointers to entries in another library, where further information about the components is stored. For example, there may be a pointer to an entry in a library used by a printed circuit board layout system. This would describe package geometries, power and ground pins, rules for swapping logically equivalent pins during layout etc. Other pointers could reference entries in a library containing models for a simulator.

Many schematic capture systems allow the user to modify parameter values associated with a component instance (a library can be thought of as holding component declarations, copies of which are instanced in the schematic) and to add user defined parameters, or notes, in order to transmit specific information to another tool, or its operator. For example, one may wish to designate certain nets as critical to the layout system, or force a particular gate into a particular package during the gate packaging procedure.

Most schematic capture tools are implemented on workstations, or personal computers, that economically provide the medium to high resolution, medium speed, 2-dimensional graphics and comparatively modest computational resources required. Nearly all modern schematic capture systems have a user-friendly human interface, typically including the following features:

1. Operation under a multi-tasking operating system and a window manager allows simultaneous viewing of the schematic while running other applications (e.g. simulation).

2. Pop-up, or pull-down menus guide beginners and reduce the learning time.

3. Multiple, hierarchical menus reduce menu clutter and operator fatigue.

4. Use of icons in menus (of debatable usefulness).

5. Help windows are available to provide the user with context relevant information.

6. For experienced users, an optional command language interface (preferably allowing unambiguous command verb truncation) avoids tedious menu picking operations with the mouse.

7. Pan and zoom operations are simplified by providing an overview window that shows the complete schematic at reduced scale and outlines the working area, which is shown in detail in the main window. Panning is achieved by dragging the outline of the working area around inside the overview window. Changing the size of the working area outline in the overview window results in immediate zoom in (or zoom out) in the main window.

8. Operator fatigue can be reduced by using the technique of snapping a placed component onto a grid, or snapping a wire end point onto a nearby pin.

9. Individual objects can be collected into groups and then operations (move, copy, etc..) can be made on the group.

10. A sequence of commands can be captured in a user defined command macro and then the macro can be executed to speed up repetitive application of the same command
sequence. A more flexible system displays the command macro in the command window and allows the user to edit it before it is executed.

11. All operator key strokes and mouse operations are recorded in a key stroke file. The key stroke file can be used to drive the schematic editor in playback mode in order to recover a work session after a crash of the schematic editor or operating system. It also forms an ideal mechanism for reporting bugs to the vendor.

12. An *undo* command reverses the effect of the last executed command, allowing recovery from mistakes or operations producing unsatisfactory results.

Many schematic editors support multi-level drawing hierarchies, enabling the top-down design methodology to be applied (see Figure 5). Each level of the drawing hierarchy can include multiple schematic sheets. An off-line compilation procedure interprets the schematic graphics, checking for electrical design rule violations. A linking procedure then links signals that traverse between sheets or between levels of the design hierarchy. Normally each page can be compiled separately, and sub-trees of the hierarchy can be linked (and verified by simulation) before the complete design hierarchy is entered.

The page compilation process checks the drawing for simple rule violations, for example:

1. Duplicate use of component or signal names.

*Figure 5:* Hierarchical schematics support the top-down design methodology.
2. Multiple outputs driving the same net.

3. On-page signals that either have no drive, or no sink.

4. Signals that go off-page but have not been assigned a name (and therefore cannot be linked to their counterparts on other pages).

5. Named wires that join to a bus whose separately declared contents do not include the name of the wire.

6. etc.

The compiled and linked design is stored in a design database. In an integrated CAE/CAD system, all tools will be driven directly by the design information entered into the database by the schematic editor. This eliminates the need of re-capturing the design in a form suitable for each new tool, and, even more important, eliminates inconsistencies in the different design databases used by the tools (in a heterogeneous system, built from loosely coupled tools, these may creep in through errors made in re-capturing the design for each different tool, through bugs in interface programs, or through forgetting to update a design modification in one tool's database). Tools that are not part of the integrated tool set can be interfaced to the database by user written programs that extract and format information into a netlist format accepted by the external tool. Integrated systems use databases in binary format (for efficiency), but usually provide a means of extracting (and sometimes also entering) data through a set of routines that can be called from a user program, thus hiding the detailed internal structure of the database from the programmer.

Many integrated tool sets are provided with design management software that keeps track of design versions and modification history. Simple automatic checks can prevent much lost time; for example, warning that a modified page has not been recompiled when the user tries to run the linker.

2.2 Design Capture by Hardware Description Languages

The schematic capture package is a natural tool for capturing structure. Many CAE systems augment schematic capture with structural or behavioural design capture by use of hardware description languages (HDLs). Some examples are the ISPS language (see references [3] and [4]) for describing and simulating computer architectures, or the well known languages PALASM [5] and ABEL [6] used to capture functionality for programmable logic synthesis systems. We will use the IEEE standard VHDL \(^2\) [7] as an example of design capture by hardware description language. VHDL was developed as a means of capturing a digital design in a standard form for purposes of design verification, synthesis and testing. A second function of the standard is to provide unambiguous, verified (by simulation) documentation for procurement of systems. The present text does not aim to describe VHDL in detail, for which the reader can consult references [8] and [9], but uses VHDL to give the flavour of design capture by hardware description language.

A typical VHDL support system, shown in Figure 6, allows module descriptions in VHDL to be compiled and stored in a database. Hierarchy is supported by allowing module descriptions to reference other modules. Separate compilation of modules is possible. A linker is used to expand the hierarchy and prepare a design for use by an application tool. The VHDL support system therefore acts as a design capture and management system for the applications tools.

In VHDL a hardware module is described by an entity. In all hierarchical systems, a module can be used as a component at a higher level of the design representation hierarchy. Such components are

---

\(^2\) VHDL = VHSIC Hardware Description Language. VHDL was originally developed in the context of the United States' Department of Defence VHSIC program (VHSIC = Very High Speed Integrated Circuits).
represented at the higher level by an abstraction that defines only their interface to the outside world. In a schematic capture system this abstraction of the interface is the component's graphical symbol. In VHDL, the corresponding component interface is the entity declaration, as shown by the trivial example for a 2-to-1 multiplexer Mux in Figure 7. The entity declaration names the entity’s ports and specifies their direction and type (TTL_BIT). In the example, the user-defined type TTL_BIT identifies the technology and data carrier width (1). Type checking avoids erroneous clashing of different technologies or attempts to link data carriers of different widths.

The contents of a VHDL entity are described by the VHDL architecture construct. A VHDL architecture can include instances of other separately declared entities, so that a hierarchical system description can be built up. The VHDL language allows architectures to specify the contents of an entity in 3 different styles:

1. **Structural:**
   This style is equivalent to design capture by netlist. An interactive graphics schematic capture system with a programmable netlist extraction interface could be used to produce a VHDL structural style description. Figure 8 shows a trivial structural VHDL architecture describing a 2-to-1 multiplexer at the gate level. The component types instanced in the architecture are first declared, together with their port lists (which must match with the port lists made in the entity declarations of the components shown in Figure 7). Internal signals are declared and then used to specify the connectivity between the ports of component instances.

2. **Data Flow:**
   This style of architecture allows the modules to be described in terms of a data-flow diagram specifying the flow of data between a set of concurrent processes. In its simplest form, the data-flow style architecture can include a set of concurrent assignments to declared local signals, or the output ports of the entity. The order of execution of the assignment statements is driven by changes in the data carriers, not by the lexical ordering of statements in the source code, nor by flow-of-control constructs as found in procedural languages. The data-flow style description for the 2-to-1 multiplexer is shown in Figure 9. In this style the signals act as data carriers. The data-flow style does not define structure, it defines the flow and transformation of information. Using the possibility of having the assignments made only when a conditional guard expression becomes true, the dataflow style can easily describe functionality in terms of the familiar state machine model (see reference [9] for detailed discussion).
entity Mux is
    port (a, b, s: in TTL_BIT);
    port (c: out TTL_BIT);
end Mux;

entity Inverter is
    port (input: in TTL_BIT);
    port (output: out TTL_BIT);
end Inverter;

entity Nand2 is
    port (input1, input2: in TTL_BIT);
    port (output: out TTL_BIT);
end Nand2;

Figure 7: VHDL entity declarations for Mux, Inverter and Nand2.

3. Behavioural:
This architecture style specifies the function of a black box using an algorithmic description. The behaviour of the module is described by a process using procedural programming language constructs of the VHDL language. No reference is made to internal structure of the black box (except that internal states will need to be defined, in the form of local variables, to correctly model the behaviour of a sequential circuit). Figure 10 shows a behavioural style architecture for the 2-to-1 multiplexer. The arguments of the process statement define the process sensitivity list, i.e. when any of the signals in the sensitivity list changes the process will be executed.

It is possible to mix the structural, data-flow and behavioural styles in the same VHDL description. Thus, instead of limiting the data-flow architecture to simple assignment statements, multiple procedural processes can be invoked concurrently in the framework of a data-flow architecture in order to model the parallelism of hardware. Each of the processes executes its code sequentially, but runs concurrently with the others. Language constructs are available to synchronise concurrent processes (e.g. WAIT on signal_name).

The advantage of VHDL over a schematic approach is that the user can specify desired behaviour directly, instead of having to first "invent" a network structure of primitives that will exhibit the system behaviour he desires, and then capture that structure. In fact, one of VHDL's design goals was to support creation of technology independent system descriptions that can first be verified by simulation, then compiled by a synthesis tool into a structural implementation in a given technology. Thus using VHDL for behaviour capture for input to a synthesis tool is fundamentally different from using a schematic capture tool for input to verification and layout tools. At the moment the only application tools available for VHDL are simulators, but other types of tool, including synthesis tools, are under development [10].
architecture gate_level_structural_example of Mux is

  component Inverter
    port (input: in TTL_BIT);
    port (output: out TTL_BIT);
  end component;

  component Nand2
    port (input1, input2: in TTL_BIT);
    port (output: out TTL_BIT);
  end component;

  signal not_s, sig1, sig2: TTL_BIT;

begin
  I1: Inverter portmap (s, not_s);
  G1: Nand2 portmap (a, s, sig1);
  G2: Nand2 portmap (b, not_s, sig2);
  G3: Nand2 portmap (sig1, sig2, c);
end gate_level_structural_example;

Figure 8: A gate level structural VHDL architecture for the entity Mux.

architecture dataflow_example of Mux is

  signal not_s, sig1, sig2: TTL_BIT;

begin
  P0: not_s <= not s after 2ns;
  P1: sig1 <= not a and s after 5ns;
  P2: sig2 <= not b and not_s after 5ns;
  P3: c <= not sig1 and sig2 after 5ns;
end dataflow_example;

Figure 9: A data-flow VHDL architecture for the entity Mux.
architecture behavioural_example of Mux is

    initialize c: TTL_BIT to 'X';

begin
    process (a,b,s)
    begin
        case s is
            when '0' =>  c <= a after 12ns;
            when '1' =>  c <= b after 10ns;
            when 'X' =>  c <= 'X' after 10ns;
        end case;
    end process;
end behavioural_example;

Figure 10: A behavioural style VHDL architecture for the entity Mux.

3. Design Verification

The traditional method of verifying a design is to construct and test a prototype. However, in the case of integrated circuit design the cost and turn around time for manufacturing the prototype are sufficiently large to encourage designers to invest substantial efforts to produce designs that have a high probability of functioning correctly at production of the first prototype. This incentive, together with the availability of cheaper and more powerful workstation hardware, has led to the development and widespread use of simulation in IC design. The cost, sophistication and user-friendliness of these tools have now evolved to the point where their use in board level design is advantageous.

3.1 Overview of Simulation

Simulation is the time domain analysis of a network of interconnected functional modules, each of known behaviour. The purpose of simulating the network is to verify the design process; namely the (usually manual) decomposition of a high level representation of the system into a network of simpler modules. The design decomposition process may proceed through several levels as shown in Figure 5, and at each step the network may be simulated in order to check some, or all, of the following points:

1. it has functionality that is equivalent to that of the higher level system specification and reaches the desired level of system performance (maximum clocking rate, system throughput, etc.).

2. it does not contain timing problems (spikes, races, setup- and hold-time violations, etc.).

3. it will function correctly when constructed with any combination of component samples, taking into account the range of timing characteristics guaranteed by the component manufacturer. This application of simulation is commonly referred to as timing verification.
4. it can be effectively tested for the presence of faults by the application of a set of specially developed test vectors. Evaluating the effectiveness of the set of test vectors is carried out via fault simulation.

A crucial issue in simulation is the accuracy of modeling. Most simulators have built-in modeling assumptions over which the user has little or no control, for example:

1. the representation of logic levels,

2. the modeling of technology dependence via logic strengths and the resolution of signal contention,

3. the modeling of the flow of time and the delay model.

Modeling decisions made on the above aspects determine the accuracy with which the simulator mimics the flow of signals between the functional units of the system.

The behaviour of the functional modules of the network can be modeled at different levels of abstraction. Some commonly employed modeling levels are the circuit, switch, logic, functional, behavioural, and physical levels. These modeling levels are discussed in section 3.7 The high level models hide detail and can usually be evaluated with less computer time than the lower level models. Often the designer will not have enough information about a component to develop a detailed model. The choice of device modeling level is a trade-off of modeling accuracy against run time and difficulty of development of the model.

There exist many different simulators that are specialised for modeling at one level of abstraction (e.g. the behavioural level, logic level, or circuit level). However, there is another class of simulator that, within the same tool, can support several levels in the hierarchy of modeling abstraction. These hierarchical simulators allow a single simulator to verify the top-down design process after each stage as it moves from the architecture level down to the lowest level of decomposition. A hierarchical simulator that allows "mix and match" of different levels of modeling abstraction in the same run is known as a mixed-level simulator. A simulator operating at only one level of modeling abstraction may well be superior to a mixed-level simulator at that level, but on the other hand, there may be a considerable interfacing effort needed to port the design from one tool to another as the design process progresses top-down. The mixed-level simulator clearly has advantages in large projects, where different parts of the design have advanced to different levels of detailed implementation. The mixed-level simulator also makes possible the simulation of systems containing complex standard parts (microprocessors etc.) that can be modeled relatively easily at high levels of abstraction, but would be very difficult to model at a low level, because of the complexity of writing such detailed models (even if all the necessary information about the internal structure of the part were available).

Today one can find many hierarchical, mixed-level simulators that span the range architectural to logic or switch level. The inclusion of circuit level modeling into this range is much more difficult because analogue simulators use fundamentally different models (discrete states and times are replaced by continuous variables, simple boolean expressions or algorithmic descriptions by sets of differential equations). Nevertheless, some companies are now advertising so called mixed-mode analogue/digital simulators. It appears that these are just the kernels of a digital and an analogue simulator tightly coupled through internal communication [11]. Severe performance problems may be encountered when circuits contain feedback between analogue and digital parts and the time advance mechanisms of the two simulators are not synchronised [11].

3.2 Modeling Multi-valued logic

Almost all digital simulators model at least 3 different logic values, namely '0' (false), '1' (true), and 'X' (unknown). The unknown value X is used to represent the condition of a memory device that
has not been initialised, or the uncertain outcome of the operation of a device outside of its specified normal operating conditions (e.g. simultaneous assertion of the "clear" and "set" lines of a latch would be modeled by setting the latch with an 'X' value), or the clashing of different logic values of equal strength where the outcome of the contention is uncertain. The appearance of an X value in the simulator output signals a potential problem to the designer. Additional logic values are found in simulators that are used to verify system operation in the min/max range of device timing characteristics (see section 3.9).

3.3 Modeling Technology and signal contention

To be useful, a simulator should be able to model technology dependent behaviour accurately. For example, the outcome of clashing different logic values on a bus depends on the technology of the driving devices (open-collector TTL, tri-state TTL, ECL, etc.). Resolving signal contention is usually modeled by assigning a strength to each output and applying the rule that the strongest output dominates. Typically a simulator may work with 4 different strengths, in order of decreasing strength: Unknown (U), Forcing (F), Resitive (R), and High impedance (Z).

The full representation of the state of an output in a given technology then consists of the pair (Strength, Value). Table 1 shows the 12 possible states used by a simulator that models with 3 logic values and 4 strengths.

For example a totem pole TTL output stage can drive either an F0 or F1 state. An open-collector TTL output should be modeled by having it drive an F0 state when the output transistor is conducting and a ZX state when the output transistor is turned off (the output has to be pulled up to an R1 state by an external resistor)\(^3\). A tri-state output drives F0 or F1 while enabled and ZX when disabled.\(^4\)

The unknown strength U is used in those technologies where the strength of the 0 and 1 levels are different (e.g. NMOS or ECL); in such technologies any driver that drives an unknown level X must do so with an unknown strength U.

<table>
<thead>
<tr>
<th>Strength</th>
<th>U</th>
<th>F</th>
<th>R</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>UX</td>
<td>FX</td>
<td>RX</td>
<td>ZX</td>
</tr>
<tr>
<td>1</td>
<td>U0</td>
<td>F0</td>
<td>R0</td>
<td>Z0</td>
</tr>
<tr>
<td>2</td>
<td>U1</td>
<td>F1</td>
<td>R1</td>
<td>Z1</td>
</tr>
</tbody>
</table>

Table 1: Typical states used by a 12-state simulator.

---

\(^3\) Some simulators simplify open-collector modeling by driving a Z1 state to the output when the logic output value is true. However, this is inaccurate and may result in the designer not noticing a forgotten pull up resistor.

\(^4\) Some simulators model an assignable decay time after the removal of the tri-state enable during which the output state takes the Z strength but retains the previous logic value. After the expiration of the decay time the logic value changes to the unknown X. This mechanism models the leakage of residual charge which maintains a definite level during a short time.

\(^5\) Some simulators encountered difficulties when the use of tri-state and bi-directional pins became common.
Table 2 shows how a typical simulator might resolve contention. Note that almost universally the simulator designers adopt the strategy of assigning the most pessimistic outcome. This is in order that potential problems will not be overlooked by the user. This strategy is sometimes disputed by first time users; it is however based on many years of practical experience with these tools.

Most simulators have their state and contention resolving models hard coded into them. They are therefore designed for use with a specific technology or set of technologies. The user has no control over the accuracy of these aspects of modeling. It is interesting to note that the VHDL language, on the other hand, has been designed to be technology independent - the user can accommodate new technologies or, to some extent, control the accuracy of modeling since he can declare a new set of logic states, and write a contention resolving function that, when called with clashing states, returns to the simulator the outcome of the clash appropriate for the new technology.

<table>
<thead>
<tr>
<th>Z0</th>
<th>Z1</th>
<th>ZX</th>
<th>R0</th>
<th>R1</th>
<th>RX</th>
<th>F0</th>
<th>F1</th>
<th>FX</th>
<th>U0</th>
<th>U1</th>
<th>UX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z0</td>
<td>ZX</td>
<td>ZX</td>
<td>R0</td>
<td>R1</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
</tr>
<tr>
<td>Z1</td>
<td>Z1</td>
<td>ZX</td>
<td>R0</td>
<td>R1</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
</tr>
<tr>
<td>ZX</td>
<td>ZX</td>
<td>R0</td>
<td>R1</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
</tr>
<tr>
<td>R0</td>
<td>R0</td>
<td>RX</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>R1</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>RX</td>
<td>F0</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>F0</td>
<td>FX</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>F1</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX</td>
<td>FX</td>
<td>U0</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U0</td>
<td>U0</td>
<td>UX</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U1</td>
<td>U1</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UX</td>
<td>UX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The 12-state contention resolving matrix.

3.4 Modeling time and delays

Digital simulators may model time in such a way that an event (a change of state on a device output) can occur only at a discrete integer time value. For such simulators the basic time unit limits the resolution with which event timing can be simulated. However, if the time unit is chosen to be much less than the typical device delay, a fairly accurate simulation of the effects of differences in delays of different devices can be obtained. Other simulators allow events to occur at any time.

Simulators make various approximations to the propagation delays of devices. Commonly encountered delay value models are:

1. zero delay model
2. unit delay model
3. fixed delay model
4. assignable delay model

The zero and unit delay models were used in early gate level simulators; they trade off accuracy of timing for performance and simplicity of implementation by assigning the same delay (0 or 1 time unit) to all devices. The zero and unit delay models allow logic verification, but are ineffective at uncovering timing problems in the design. The fixed delay model accurately specifies a different delay for each
device type, but each device of a given type has the same delay. Assignable delays allow each instance of a device type to be assigned its own delay value (permitting the modeling of the effect of fan-out load on circuit speed).

Another important aspect of delay modeling is that of the signal transport mechanism through the device. The 4 principal models used are described in Figure 11.

![Diagram of delay models](image)

**Figure 11:** Common delay models.

(a) In the transport delay model a pure delay is added at the device output.  
(b) In the case of the inertial delay model, when the input pulse width is smaller than the input’s inertial delay the pulse is not propagated to the output, but a spike warning is generated, modeling the fact that the output does not reach a well defined logic level before returning to its original value. If the input pulse is wider than the inertial delay it is propagated to the output after the transport delay.  
(c) In the rise and fall time delay model the delay to a rising output edge (Dr) may be different from the delay to a falling output edge (Df). This model is important for MOS circuits.  
(d) In the ambiguity delay model precise delays are not known, but the delay is known to vary statistically from sample to sample over a range defined by a minimum and a maximum value. A pair of delay values are assigned to specify the time range in which the output may be either rising (R), falling (F), or changing (C denotes that the output may be either rising or falling). This model is used in timing verification.

### 3.5 Components of a simulation system

Figure 12 shows the basic elements of a simulation tool. The simulator requires information about the connectivity of the network under simulation and the models for the functionality of the components. Connectivity data is provided by programs that parse a netlist, or a set of schematics. Modeling data are provided by a modeling language compiler. Most simulators are driven by a set of
linked list data structures (called tables hereafter) containing fanout lists for each output, signal state tables, pointers to modeling primitives, timing parameters, etc. A linking stage builds the exact data structures required by the simulator. A simulator process executes the simulation algorithm and is controlled and monitored by a user interface process that mimics a software logic state analyzer. An interactive monitor task, analogous to a software debugger, allows the user to set break points, trace signals, inspect and initialize internal model states, etc. User friendly monitor tasks present waveforms graphically, and are often tightly coupled to the schematic editor (running in another window), so that the user can pick signals and devices from the schematic window and immediately see their state reflected in the monitor’s window. Not all simulators are truly interactive; some dump the trace of selected signals in a file which is subsequently analyzed by a post-simulation processor.

3.6 The event driven simulation algorithm

Most digital simulators use the principle of the event driven algorithm shown in Figure 13. Event driven simulation drives an input stimulus forward in time, propagating its effect through all branches of the fanout tree. System behaviour is modeled as a series of discrete events in each of which the new state of a device output is propagated to the associated signal. When an event occurs, the change is propagated through the signal’s associated fanout list to all device inputs connected to the signal.
Many events can occur concurrently, so all events at the current simulation time are first propagated to their fanouts. Next, each of these fanout devices is evaluated using the new states of their fan in signals, i.e. their future changed output states are calculated and scheduled to produce new events after the elapse of the device’s delay time by queuing them in an event queue. Once all fanout devices have been evaluated time is advanced to the time of the next scheduled event(s) found in the event queue. Events are removed from the queue and used to drive the simulation through another time step. The principle of selective trace is used to speed up execution by cutting off propagation of device outputs that do not change from their previous value (shown in the flow diagram of Figure 13).

The implementation details of the time advance mechanism and the event queue management algorithm also affect the efficiency of the simulator. Some simulators use a fixed time increment mechanism, which often requires many wasteful time steps to be executed in the event queue manager before simulation time is advanced to the time of the next scheduled event. Other event queue managers are able to jump simulation time directly to the next scheduled event time, so saving the overhead of searching the event queue at times when it contains no events for execution. There are many different detailed implementations of the event queuing algorithm in use in real simulators. Figure 14 indicates the principle of the widely used time wheel event queuing method.

**Figure 13:** Event driven simulation.

*Selective trace* cuts off propagation through the fanout list whenever an evaluated output does not change.
Event driven simulation is the most common, but not the only algorithm. For example, reference [12] describes a demand driven algorithm which starts from a user request for the state of a particular node, and drives this demand backwards in time through the network until it can be resolved in terms of the network’s input stimuli. This algorithm does not waste time propagating the simulation down branches of the fanout tree that the user will not look at. In addition it removes the overhead of maintaining an event queue. It is claimed to run 2 to 3 times faster than event driven simulation.

3.7 Modeling Device Behaviour

The applications for which a simulator is suitable depend on the level of modeling abstraction supported by the evaluation routines used to calculate new states at the device outputs and schedule them in the event queue. The main levels of device modeling abstraction in use are described below.

3.7.1 Circuit and Analogue Behavioural Modeling Levels

Analogue simulators model devices at the circuit level by treating time, voltage and current as continuous functions. A set of differential equations are solved using numerical integration methods and the circuit input stimuli as boundary conditions to derive the circuit’s behaviour. This method is fundamentally different from the event-driven digital simulation algorithm in which the variables are restricted to a small set of discrete values. It is briefly mentioned here for the sake of completeness; more detailed information can be found in [13].

The widely used program SPICE [14], originally designed for analogue simulation of full custom integrated circuits, has the analogue models of primitive devices built into the source code. SPICE primitive models include resistors, capacitors, inductors, transmission lines, voltage and current sources, diodes, bipolar junction transistors (BJT), junction field effect transistors (JFET), MOS FETs, etc. These generic device models are characterised for a particular device and process technology by supplying values for model parameters (e.g. the Gummel-Poon BJT model requires 40 parameters). A difficulty often encountered with SPICE is to obtain the set of parameter values that characterise a particular device or process. Another practical difficulty is that the numerical methods used to solve the set of simultaneous differential equations do not always converge.

More recent analogue simulators like SABER [15] are provided with a library of circuit level device models and a modeling language that enables users to develop their own analogue models (these can describe any analogue behaviour, viz: electronic, mechanical, etc.). The SABER modeling lan-
guage can be used to build models at the circuit level (by a composition of analogue primitives), or at the higher, analogue behavioural level (where a black-box transfer function is used without reference to circuit structure). With programs like SPICE, the computational complexity of circuit level modeling limits their application to relatively small circuits (a few hundred devices) in practice. Larger analogue systems can be modeled using the analogue behavioural modeling approach.

3.7.2 Switch Level

Switch level modeling is used to simulate larger digital MOS circuits with less accuracy, but higher execution speeds than circuit level simulators can achieve. The MOS transistor is modeled as a switch with some simple analogue behaviour. The switch level includes modeling of transistor delay, rise and fall times, loading effects and effects such as charge sharing (the redistribution of stored charge when a pass transistor opens a path between two isolated circuit nodes).

3.7.3 Logic and Functional Levels

A logic simulator contains logic level primitive behaviours hard coded into its evaluation process. Typical primitives encountered at this level are logic operators such as NAND, NOR, NOT, AND, OR, XOR, a DELAY primitive, MOS pass transistors, and memory elements such as various types of flip-flop and latch. These primitives can be directly invoked to provide the behaviour of a simple logic device; in addition the user will have to specify the required timing information (delays, and timing constraints such as maximum clock frequency, minimum pulse width, setup and hold times, etc). Table 3 shows the behaviour of the AND primitive used by a simulator employing 3-valued logic (0, 1, X).

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

*Table 3: Truth table for the AND primitive in 3-valued logic.*

The user models his system by specifying the structure of the network of logic level components. The VHDL structural architecture shown in Figure 8 describes a gate level model of a multiplexer. More complex components that cannot be directly modeled by one of the logic simulator’s primitives are handled using macros. An equivalent circuit of the complex component is declared using logic simulator primitives. This may be done graphically, or in a netlist style. The equivalent circuit is stored in a macro library. A macro processor will then expand any occurrences of the complex component in the system structural description into logic primitives. The simulator still operates at the logic modeling level and no performance improvement is achieved through higher level modeling.

A functional level simulator may have built-in, higher level primitive models such as random access memory (RAM), read only memory (ROM), programmable control structures (PLA), arithmetic logic units (ALU), etc. Often a functional level simulator will allow modeling of device behaviour using boolean expressions or truth tables. These will be directly used by the simulator’s evaluation routines and will not be broken down into an equivalent circuit. Performance is significantly enhanced by this modeling approach as only one evaluation pass is required and it avoids the over-
head of evaluating the multiple devices of the equivalent circuit and scheduling events through the fanout tree.

A functional model will be described using a non-procedural, data-driven language (i.e. the order in which statements are written has no influence on the sequence in which they are executed; execution order is determined uniquely by changes in the data and by data inter-dependencies). Figure 9 shows a simple example.

3.7.4 Behavioural Level

Although commonly called the behavioural modeling level, a better description for this level of modeling would be algorithmic. The behavioural modeling approach can describe the functionality of a black-box without reference to its internal structure. This is done using a behavioural hardware description language (HDL). Behavioural HDLs are usually based on extensions to familiar, procedural programming languages such as C, Pascal, Ada, etc. A trivial VHDL behavioral model of a multiplexor is shown in Figure 10.

Most HDLs allow the description of both structure and behaviour, so that a system can be described in terms of a network of black boxes. The behaviour of each black box is described using the behavioural description features of the HDL. The HDL contains extensions to the standard programming language that make the description of timing and concurrency of hardware possible. Generally, multiple processes can be declared and will be simulated pseudo-concurrently in order to model concurrency in hardware. Each process consists of code that is executed sequentially. Whenever the simulator needs to evaluate a black box its evaluation routines (see Figure 13) call the corresponding process code. Processes can be synchronised by using either statements that suspend execution of a process until a named signal is set (or reset), or statements that fire up a process whenever a named event occurs.

Thus the HDL will be compiled into code sequences that can be called by the simulator’s evaluation routines to return logic behaviour. At the same time the compiler extracts timing information into the tables used by the simulator’s event scheduling mechanism. For example, the sequential code fragment (consisting of only one case statement in this trivial example) that forms the body of the process declaration in Figure 10 will be used to evaluate the output of the multiplexor whenever one of its sensitive inputs (listed as arguments of the process statement) changes. The delay times specified in the after clause will be used to schedule the associated output event in the event queue.

Some systems actually compile the behavioural description into code using the simple primitive evaluation models and operators built into a logic simulator. This allows the behavioural description to be mixed with logic level descriptions and run on a logic level simulator. In this case the user works at a high level of abstraction but the potential for performance improvements through the high modeling level is not fully realised. The advantage of this approach is that the simulation can be accelerated by moving to an existing special purpose hardware implementation of the logic level simulator (see section 3.8).

3.7.5 Physical Modeling

In cases where insufficient information is available to allow the writing of a behavioral description, or the investment required to write the model is judged to be too high, the technique of physical modeling can be used. A real sample of the chip or board (the physical model) is stimulated with the input pattern generated by the simulator and provides the behavioural response to the simulator as shown in Figure 15.
Figure 15: The principle of physical modeling.

This technique is commonly used to model microprocessor ICs or other complex VLSI circuits. However, many VLSI circuits employ dynamic MOS technologies and will not function correctly if they are clocked at a frequency below some minimum value (which can be as high as several MHz). For these circuits the physical modeling hardware must buffer the entire stimulus pattern history in a FIFO buffer memory, from where it can be replayed at full speed. As soon as the simulator needs to evaluate the next state of a component being modeled with the physical modeler, it adds the current input stimuli to the input pattern history buffer, and then plays back the stimuli to the physical model at a selectable clock frequency (typically up to 20 MHz). The last output pattern generated by the model is picked up by the simulator.

The width and depth of the history buffer determine the number of pins the model can have and the maximum number of clock cycles that can be simulated. Some physical modeling systems allow compression of the history buffer by eliminating all patterns where only data pins are changing (only changes on store pins effect the internal state of the model) [16]. If a single physical model is used to provide behaviour for several instances of the component type, a history buffer needs to be kept for each model instance (even when a static MOS device is being modeled). In order to circumvent this limitation, most physical modeling systems are modular and allow multiple physical models to be used simultaneously.

The physical model only provides the simulator with response patterns; the designer must still write a description of the timing delays and constraints for each pin. In addition, because physical modelers only sample the value of model outputs, the user has still to write a description predicting the strength of the outputs. When physical modeling is used with a software simulator, the overhead of doing I/O in a multi-tasking operating system and the necessity of replaying the complete contents of the history buffer at each evaluation, can result in relatively slow performance. Nevertheless, the performance is likely to be faster than a behavioural model for complex devices, and probably most of the time will still be spent in the software simulating the other parts of the system. The I/O bottleneck can be avoided by integrating the physical modeling system into a special purpose hardware accelerator for simulation (described in section 3.8). For industry-standard microprocessor and peripheral chips, an alternative to the expense and complication of physical modeling is to purchase high level behavioural models from specialist modeling companies.

3.8 Hardware Acceleration of Simulation

Simulator performance is usually quoted in terms of events/sec, but sometimes in terms of evaluations/sec. With a typical network each event results in about 2.5 evaluations. A software implementation of the event driven algorithm typically results in execution rates of about 1000 events per second per MIPS of CPU speed. Thus, typical workstations today run event-driven simulations at speeds of a few thousand events per second. The amount of work done by one event depends on the
level of modeling adopted. When a high modeling level is used, more work is done per event and relatively less time is spent on the overhead of the event driven simulation algorithm, resulting in improved speed compared with what can be obtained at more detailed modeling levels.

The relative simplicity of the event driven algorithm lends itself to implementation in special purpose hardware. Various hardware implementations have been made resulting in execution speeds ranging from around 40,000 events/sec (cost $25K) up to around $10^9$ events/sec (cost $3M$).

At the low performance end of the hardware accelerated simulators are architectures like the Personal MegaLogician from Daisy Systems Corporation [17] shown in Figure 16. This consists of a straightforward implementation of the 3 main parts of the event driven algorithm in three microprogrammed units (the state, evaluation and queue units). Each unit pipes its results into FIFO buffers at the input of the next unit, enabling all three units to work concurrently. Each unit has an attached memory that contains the tables used by the simulator. The size of these tables limits the maximum size of the network that can be handled by the hardware engine. Behavioural models are compiled into descriptions using model primitives of the hardware simulator's evaluation unit. The Personal MegaLogician uses the same model primitives as Daisy's software simulator and produces identical results. The physical modeler is integrated into the accelerator as a special hard wired unit that can work in parallel with the evaluation unit. The accelerator integrates well into the software simulation environment of that company and is priced to offer an attractive cost/performance advantage over a software simulator running on a super workstation.

![Figure 16: The Personal MegaLogician architecture closely mirrors the data flow of the event driven simulation algorithm.](image)

At the high performance end of the simulation hardware engines we find several architectures which use multiple, concurrently operating, special purpose engines, each one with its local event queue, state unit and evaluation unit. The circuit under simulation is partitioned between the engines. Signals that cross between parts of the circuit assigned to different engines are handled by sending short messages over a high performance cross-bar network. The messages contain data that allow the sending unit to enter an event into the receiving unit's event queue, or to push an evaluation packet onto the receiver's evaluation stack. Each unit has to keep simulated time in lock step with a master time advance mechanism.

These architectures are modular; one adds modules until the hardware simulator has the capacity to handle the system to be simulated with the desired level of performance. Mapping the circuit to be simulated onto the multiple engines must be done in a way that minimises inter-engine communication and, at the same time, distributes the activity of the circuit uniformly over all engines so that none of them ends up idling while waiting for an overloaded engine to catch up. Reference [18] discusses the various performance related issues in parallel simulation. Some examples are the accelerators from Zycad [19], or Daisy's GigaLogician [20]. IBM's Yorktown Simulation Engine [21] is a highly parallel machine which, in contrast to the others, does not use event-driven simulation. Every gate is evaluated at every cycle, even if its inputs have not changed.
These high speed accelerators can be extremely fast, but one should be aware that most of them have sacrificed accuracy and flexibility for speed. Some only directly support low levels of modeling (e.g. the gate level), although some of them are provided with software that can compile behavioural models written in certain behavioural modeling languages into an equivalent gate level structural description. Others rely upon a relatively slow host computer to execute behavioural models, or to provide primitives for RAM, ROM, PLA etc. Others, like the GigaLogician, include multiple parallel hard wired units for logic level primitives, and multiple programmable processors for execution of sequential, high level language behavioural descriptions. Accelerators may sacrifice accuracy of delay modeling (using for example the unit delay model) for simplicity of implementation, perhaps forcing the adoption of a strictly synchronous design methodology. They may require time consuming translation and interfacing procedures in order to use them. They are typically used by large systems houses that can afford the necessary investment and support effort, and enforce a rigorous design methodology needed to make them useful.

Depending on the length of a simulation run, the extra compilation phase required to prepare a design for hardware assisted simulation may actually take more time than is saved by speeding up the simulation run. In these circumstances, a more powerful general purpose workstation may give better overall improvement, because it speeds up both compilation and simulation. The hardware simulation engine can be most useful towards the end of a design project, when very long runs may be needed to catch the few remaining bugs, or when the simulator is used as a test bench on which to develop prototype operating systems software for an as yet unbuilt computer, or to test applications software for an embedded processor. Another area where hardware simulation engines find application is in computation intensive fault simulation (see section 4.1 ).

3.9 Timing Verification

This is an important area in which a network is examined for correct operation taking into account the possible statistical spread of timing characteristics of each component. The *breadboarding* technique, in which a physical prototype is constructed and tested, may well conceal potential timing problems that later show up during production when a batch of components with slightly different timing characteristics is used. Costly redesign and field modification may then be necessary to correct the design error and, in addition, much system time may be lost tracking down the problem and getting it fixed. For example, using a timing verifier to check the design of data-acquisition electronics might save the extremely costly use of beam and experiment time to track down design errors.

As already mentioned in section 3.2 the timing verifier uses more logic values than the simulator in order to identify times in which a signal’s state is not guaranteed to be known. Figure 11 (d) shows how a transition from 0 to 1 on the input of a device is propagated to the device’s output. During the time slice defined by the device’s ambiguity delay the output state is not known with certainty, all we know is that somewhere in this interval it will change from 0 to 1. In the timing simulator, the output signal is assigned the value *rising* (r) during this period. Other possible logic values used by timing verifiers are typically *falling* (f), *changing* (c) (when the signal could be either rising or falling), or *stable* (s) (the level is known not to be changing, but it is not known whether it is 0 or 1). Table 4 shows the truth table for an AND gate as used by a typical 6-valued logic timing verifier. The timing verifier would typically use 6-valued logic together with 4 strengths, i.e. 24 states.

Most timing verifiers allow the user to choose the timing range values for the ambiguity delay from the *minimum*, *nominal* and *maximum* delay values for the components. Thus the timing verification can be done using min/max, min/nom, or nom/max ranges. Potential timing problems will show up, for example, when the data input (D) of a flip-flop is falling, rising or changing within a time window defined by the range in which the clock is in the rising state (plus and minus the setup and hold times of the flip-flop).

For synchronous designs, the designer can apply the *stable* state to all primary inputs of the circuit except the clock input. All possible timing problems associated with clocked elements will then be detected. Value independent timing verification saves the designer the task of inventing an exhaustive set of test patterns to exercise all possible delay paths to the clocked elements.
Table 4: Truth table for the AND primitive in 6-valued logic.

\[(s = \text{stable}, \ c = \text{changing}, \ r = \text{rising}, \ f = \text{falling})\]

3.10 Using Simulation Tools in practice

Although, in principle, it is now possible to simulate almost any digital design by using the combined arsenal of techniques outlined above, not every designer works in an environment where the necessary financial investment and support is available or can be justified. Nevertheless, experience at CERN has shown that a large fraction of board level designs can be profitably simulated using affordable software simulators on CAE workstations. Table 5 shows job statistics for a design project that was carried out at CERN using CAE and CAD techniques.

<table>
<thead>
<tr>
<th>ACTIVITY</th>
<th>NUMBER OF WORKING DAYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility study (choose between analogue or digital solutions).</td>
<td>14 (12%)</td>
</tr>
<tr>
<td>Schematic capture, modeling FADC chip, simulation and modification of design.</td>
<td>33 (28%)</td>
</tr>
<tr>
<td>Design FADC daughter boards, setup test bed, test analogue daughter boards, etc.</td>
<td>20 (17%)</td>
</tr>
<tr>
<td>Design 2 wire wrapped CAMAC boards using netlist driven software (WRAP4 [22]).</td>
<td>10 (9%)</td>
</tr>
<tr>
<td>Write test programs, design small test circuits, carry out tests, file documentation and deliver tested module to the experiment.</td>
<td>40 (34%)</td>
</tr>
</tbody>
</table>

Table 5: Statistics for a design job using CAE and CAD tools.

The job was the design, construction, testing and documentation of a Missing Momentum Trigger Module for the CERN UA2 experiment [23]. It consisted of 150 ECL integrated circuits and 12 Flash Analogue to Digital Converters (FADCs) implemented on two CAMAC boards using wire wrap. The project was completed in an elapsed time of 6 months and no design errors were detected during the testing of the prototype.
Simulation can be viewed as a valuable management tool as well as an engineering tool. In industry, large designs are often partitioned between many engineers, and the interface specifications (behaviour and timing) can be precisely defined using simulation top-down. The project coordinator can guarantee successful system operation as long as the interface specifications are met. Using simulator results, he can supervise the work of each engineer and get an early warning if part of the design cannot meet the specifications, perhaps deciding to revise system architecture. Although at CERN we have not used simulation for co-ordinating large scale projects, it has been found to be a very useful tool for supervision of an individual's design work [24], allowing identification of problem areas and fast feedback on alternative solutions.

Apart from possible run time limitations for large designs, the software simulation of semi-custom digital ASIC designs is usually relatively trouble free, largely due to the absence of device modeling problems as a result of the rather simple building blocks provided in the average silicon vendor's cell library. However, approaching the design top-down, one should first develop a behavioural model of the ASIC that allows the ASIC architecture to be validated in its final system application. Many ASIC designs successfully implement the designer’s intent, but have to be changed because the design goals were wrong. Depending on the other components used in the system design, simulation of an ASIC embedded in a system could be a relatively complex and costly task if it requires use of physical modeling or hardware accelerators.

Although simulation can demonstrate substantial benefits in appropriate design projects, it does have a few problems that need to be understood before designers can begin to reap the full benefits. In the remainder of this section we will discuss some of these practical problems. Simulation involves modeling decisions in several areas as outlined above. It is therefore an approximation to the truth. Simulation results can only be evaluated and weighed if the designer has a good understanding of the approximations that have been made in the simulator itself and in the models for the components that have been used. Due to inadequacies in the simulator, or the models, unexpected results can be produced in certain circumstances. The results of the simulator should always be critically reviewed, and after some experience the user will know the weaknesses of his simulator.

A simulator will not of itself find all the design errors in a system. The user still has to invent an appropriate set of test stimuli that will exercise the system and show up design errors. As with most tools their effectiveness depends very much on the skill of the user.

Almost all designers of simulators have adopted the policy of pessimism, i.e. when the exact outcome of an operation is not well defined the simulator will assume the most pessimistic outcome possible. The aim is to make sure that any possible error does not get by unnoticed. Most experienced users prefer the occasional trouble of trying to understand why the simulator reported a grave error when in reality there may not be a problem in the real circuit, to having a real problem slip through undetected.

First time users of simulators do not always have a thorough understanding of the limitations and approximations involved in the tools, or of the philosophy of pessimism built into the simulator. The considerable effort required to develop reliable models for components is often underestimated. The first time simulation user lacks experience to trade off the advantages of using a “new” component against the cost of introducing it into the simulation model library. In fact, even today, many hardware designers lack the basic software skills and understanding that is necessary to develop models. They may adopt the unsatisfactory solution of relying on an “expert” to write the model for them, in which case they will not understand the limitations and approximations made by the modeling expert, and maybe have problems interpreting simulation results. In short, initial expectations are often too high and, depending on the user’s skills, the learning curve for introducing simulation into the design method can be quite long. Nevertheless, once the user has progressed up the learning curve, the “pay back” can be very substantial.

We will next describe a few specific problems or limitations that may be encountered in some simulators. A common problem is an over pessimistic treatment of unknown states in certain cases, as
illustrated by Figure 17. In this example the unknown state X on the signal A produces unknown states on the two inputs of the AND-gate, and hence leads the simulator to assign an unknown state to the output of the AND-gate. In reality the output of the AND-gate will always be 0 (once the circuit has settled to a stable state) because its two inputs always take opposite logic values due to the inverter. In order to correctly simulate this circuit the simulator would have to recognise that the two inputs to the AND gate are correlated because they reconverge from a single fanout source. The problem is not serious because a well debugged circuit should not usually generate unknowns (except immediately after initialization). However, it does contribute to the propagation of unknowns through a circuit during the debug phase, making it more difficult to find the source of unknowns in a large circuit.

**Figure 17:** Incorrect output when reconvergent fanout is not recognised.

A reconvergent fanout problem can occur with timing verification of circuits containing clocked devices when the timing verifier does not recognise the correlation that exists between the clock edge and changes on the device outputs. A time range in which the clock is rising will produce an output signal which is changing in an even greater time range. If the output signal reconverges with the clock signal at another device, the timing verifier may not recognise that the two signals are correlated and it may therefore report a false timing problem. Figure 18 shows a very simple circuit that may not be correctly simulated on some timing verifiers due to the reconvergence (in this case on the same device) of the flip-flop output with the clock. To function correctly, the timing verifier needs built-in knowl-

**Figure 18:** Reconvergent fanout problems may occur with feedback.
edge that tells it that the flip-flop's output will not change before the clock edge rises, and it must recognise the reconvergence in the circuit connectivity. Although most timing verifiers handle the reconvergent fanout problem correctly, certain pathological circuit topologies may be incorrectly handled by some of them, leading to false reports of timing problems.

Another problem, occurring with value-independent timing verifiers, is that they are usually too pessimistic, reporting timing problems that can be ignored because the circuit operating state in which they appear is never reached under actual operating conditions. Timing verifiers that allow verification under specific stimulus patterns circumvent the effort of sorting out whether a reported error can be ignored. On the other hand, they place the onus on the user to invent a set of stimuli that explore all operating conditions that will be encountered in practice.

A common problem is the one of initialisation of a circuit before simulation starts. Before starting to execute with the user-defined stimuli the simulator will run through an initialisation phase where it assigns a consistent set of initial values to every node in the circuit. Normally any memory elements (RAM, latch, flip-flop etc.) are preset to an unknown state before simulation starts. In reality these memory elements will quickly fall into one of their two stable states on power up. The design may not apply a master reset signal to all memory elements because the application of the design may be insensitive to the initial states of the memory elements. The simulator will then propagate unknown states through the network, and in cases where the unknown state forms part of a feedback loop (e.g. a divide by two circuit) the simulated circuit can remain locked in the unknown state. In this case, the user will have to explicitly initialise the memory elements in a consistent way before starting simulation. Identifying all such feedback loops and correctly initialising them could be quite time consuming in a large design.

These are typical examples of minor shortcomings in the tools that the beginner will encounter and must learn to circumvent by adapting his working methods. Unfortunately many simulator vendors neglect to document the limitations and modeling accuracy aspects of their simulators, leaving the users to find out by experience and perhaps resulting in some initial disappointment. User groups and informal contact with experienced users can be invaluable for spreading this type of expertise.

4. Testing

In the manufacture of integrated circuits only a certain fraction (the yield) will be free of defects and fully functional. It is important to be able to rapidly reject faulty circuits with a simple set of tests that guarantee to filter out (in principle) 100 per cent of the faulty devices. Improvements in process technology permit ever more complex ASIC designs to be produced, and so testing has become a much more critical, difficult, and expensive issue. As a result many software tools are now available to help with the development of test patterns, and design methodologies have been developed to ease the problems of testing. Of course testability is also very important for board level designs wherever large quantities will be produced.

4.1 Fault Simulation

Fault simulation techniques are used to study the behaviour of a digital system in the presence of manufacturing faults. A single fault is injected into the simulated circuit to produce a faulty machine.

---

6 A system that cannot be brought into a known state with a master reset signal is usually regarded as untestable. However, for board designs which will not be tested on an automatic board tester, designers sometimes leave out the master reset in order to gain board space, or to be able to build a faster circuit.
Each possible faulty machine is then simulated and its behaviour is compared with that of the fault-free machine.

The inputs and outputs through which a system can be accessed for testing are called the primary inputs (PIs) and primary outputs (POs) respectively. In production an IC can only be tested via its input and output pins, whereas testing a board is in principal somewhat easier because a bed-of-nails tester or a special test connector can be used to drive or sample almost any circuit node. A tester will be used to apply a series of test patterns to the PIs of the device (or board) under test, and for each one compare the pattern produced on the POs against that calculated by the simulation of the fault-free machine. A single input test pattern, together with the expected device output pattern, is called a test vector.

The set of test vectors required to adequately test complex VLSI circuits can be very long and extremely hard to construct. Some software tools are available for automatic test pattern generation (ATPG). Reference [25] describes algorithms for ATPG. ATPG tools are very expensive and do not remove the need for the designer to understand testing issues as they still require much designer guidance and a testable design.

Once a set of test patterns has been generated, the technique of fault simulation is used to:

1. Calculate the fault detection coverage of the test vector set, i.e. the percentage of all possible faults that are detectable with the test vectors.

2. Grade faults; i.e. to produce a list of faults that are detectable by each test vector. This may allow some test vectors to be eliminated because they only detect faults that have already been detected by a previous test vector.

3. Construct diagnostic dictionaries. A single test vector may detect multiple faults, so that it cannot be used to determine why the device failed. However, if the information provided by the results of all the test vectors is combined, it may be possible to identify unambiguously the single fault that caused the device to fail. Depending on the quality of the test vectors, it may happen that multiple faults lead to the same set of failing test vectors. A diagnostic dictionary can be built up to map the set of failing test vectors, and the actual pins that failed, into a short list of candidate faults responsible for the failure. Although this might be expected to be very useful for diagnosing failures, it appears in practice to be of limited usefulness due to the inaccurate assumptions made in the fault models (see next section); often the diagnosis is misleading or gives a large number of candidates.

4.1.1 Fault models

There are many possible faults that can occur in an integrated circuit. Some examples are that nodes may be stuck-at-one (SA1), stuck-at-zero (SA0), two nodes may be shorted together, a node may be floating, etc. Fault simulation is almost always limited to modeling only a single SA1 or SA0 fault at a time. In practice, the computational complexity of fault simulation excludes the study of multiple faults and more sophisticated fault models. However, it appears that test vectors generated using the single SA0/SA1 fault models are quite good at detecting multiple faults and most of the other types of fault.

4.1.2 Fault Simulation Algorithms

Fault simulation is enormously expensive in computing resources. A circuit with N nodes will require 2N faulty machines to be simulated with the test vector set in order to evaluate all possible single SA0/SA1 faults. To stand a chance of detecting structural failures in the system, the fault simula-
tion must be done at a detailed level where all N nodes will be exercised (usually, but not necessarily, at the logic level).

In serial fault simulation each faulty machine is analyzed in a separate simulation run, requiring \((2N + 1)\) runs (including the run for the fault-free machine). It is easy to understand how development of tests for VLSI ICs can take hundreds of hours of mainframe CPU time [26].

Parallel fault simulation [27] uses a technique of packing node values (2 bits are sufficient to carry the values '0', '1', 'X') for several faulty machines into a W bit wide word and then running them all through the fault simulator in the same run. This reduces the number of runs required by a factor \((W - 1)/2\).

However, the most efficient algorithms use the technique of deductive fault simulation, in which a fault list is propagated through the circuit, rather than trying to propagate the effects of the faults through the circuit [28]. All faults are then handled in one pass. Although only one pass is required compute times are still very long. Nearly all fault simulators on the market today use the concurrent fault simulation algorithm [29], an optimised form of the fault list propagating class of algorithms.

Finally, hardware accelerated fault simulation is also used to bring fault simulation run times down to manageable proportions. An example is the Megafault [30] from Daisy Systems Corporation, that runs on the same microprogrammable hardware as their MegaLogician.

4.2 Tools and methods for design for testability

Even for moderately sized semi-custom IC designs a major part of the development effort may be spent on developing an adequate set of test vectors [31]. Consequently, a class of tools has been developed that quickly give the designer a measure of the testability of his design. With relatively little effort these tools give an early indication of the feasibility (and effort required) to produce a set of test vectors providing an adequate fault coverage. The designer can modify his design until the testability analysis indicates that the subsequent effort required for test pattern generation and fault simulation is acceptable.

Some testability analysis tools evaluate a design by looking for the presence (or absence) of features that are known to normally enhance (or reduce) testability (e.g. test points, controlled breaking of feedback loops, initialisation of flip-flops to known states, etc.). Each identified feature contributes a positive or negative score to a total testability score.

Other packages make more quantitative measures. Bennetts [32] defines the testability of a circuit node in terms of its controllability and its observability. The controllability measures the ease (or otherwise) of driving the node to a 0 or 1 state from the primary inputs. This takes into account all possible controlling paths to that node. The observability measures the ease (or difficulty) of observing the effect of a fault (stuck at 0 or stuck at 1) on that node by setting up sensitive paths that will propagate the fault effect to the primary outputs. The testability of the node is defined as the product of its controllability and its observability. The measure of system testability is the average node testability.

A number of ad hoc design for testability techniques can be used at either the board or IC design level. Reference [33] gives a good review of many of these techniques, and in addition describes a number of systematic design methodologies that result in greatly improved testability at the cost of a small overhead in circuit area, speed and pin count. These methodologies include the scan path and Level Sensitive Scan Design (LSSD) techniques, as well as a number of Built-in Self-test (BIST) methods.

In the scan path method, testing is simplified by consistently using a special latch element for all flip-flops. In normal system operation these special latches operate as standard, simple latches. However, they have an extra control line which, when active, causes them instead to shift out their current
state on a "scan out" line and to shift in a new state from a "scan in" line. The scan in and scan out lines of all latches are chained together to form the scan path. The scan path begins at a primary input and ends at a primary output. The scan path can be used by a tester to serially shift out the state of the device or system and shift in a new state. This simple method increases the controllability and observability of the circuit, reducing the difficult problem of testing sequential logic to the simpler problem of testing combinational logic.

The LSSD method was developed by IBM and is used in their products. It uses the scan path principle combined with a latch design and some additional design constraints which guarantee raceless circuit operation.

There are several BIST techniques, all of which integrate the test pattern generation circuitry and circuit response evaluation on-chip. The test pattern is either provided by a counter, or linear feedback shift registers (LFSRs) that generate a pseudo-random pattern. The circuit response for all of a large number of patterns is captured and compressed into a signature word which is compared against the expected signature (either calculated by simulation or by measuring a known good circuit). The result of the comparison produces a 1 bit "go/no-go" status indicating whether the self test was passed.

The scan path and BIST techniques are combined in the Built-in Logic Block Observer (BILBO) circuit. It is a simple circuit that has two control lines causing it to operate in 4 different modes:

1. As a normal set of latches.
2. As a pseudo-random pattern generator.
3. As a signature generator.
4. As a set of registers connected into a scan path.

The BILBO cell, or scan path latches are available in some ASIC vendor's cell libraries and are supported by their design tools.

4.2.1 Other testing problems

Redundant logic makes it impossible to detect faults in those parts of the circuit because the circuit redundancy masks the effect of the fault at the POs. Unintentional introduction of redundancy in a circuit can be tracked down and eliminated. However, circuits often contain statically redundant logic that is added to prevent a timing problem, and this will limit attainable fault coverage to less than 100 percent. Understanding these effects, mastering design for testability methodologies, and developing the skills for generation of test vectors requires considerable experience.

5. Tools for Board Layout

Modern CAD systems for Printed Circuit Board (PCB) layout typically perform some, or all, of the following functions:

1. Initial assignment of logic functions to packages.
2. Initial placement of component packages (manual or automatic).
3. Placement optimisation in order to improve board routability.
4. Optimisation of package and pin assignments in order to improve board routability.
5. Routing (manual or automatic).
6. Postprocessing router results to clean up traces, minimise use of vias, etc.
7. Production of manufacturing data and documentation.

Each of these steps will be described in more detail below. The major advantages of using advanced PCB layout tools over hand layout methods are:

1. The placement and routing process is driven by a netlist automatically generated from the schematic. It is therefore certain that the board produced will have the connectivity of the designer's schematic. This cannot be guaranteed with a hand driven layout.

2. The layout system ensures that manufacturing design rules are not violated. These can be simple rules such as the minimum pad-to-trace clearance, minimum trace-to-trace clearance, or more complex rules that, for example, forbid trace routing under certain types of component, or restrict the allowed angles at which traces can join to pads, etc. By enforcing a well established set of design rules the CAD system ensures that an optimum manufacturing yield will be obtained.

3. Once the layout itself has been completed, the CAD system can automatically produce a complete set of errorless manufacturing documentation and numerical control tapes for the manufacturing machines (e.g. drill tapes, photoplottor control tapes for generation of artwork films, solder resist masks, interface data for automatic component insertion equipment, or automatic test equipment, etc.).

There are three major classes of PCB CAD system in use:

1. The simplest class are the digitising systems that capture a hand made layout sketch, check it for design rule violations and produce manufacturing data. These systems cannot verify that the board is electrically equivalent to the designer's schematic.

2. The next class are the interactive graphics layout systems, that are driven by a schematic netlist. All layout decisions are made by the layout technician. Like the digitising class, these tools check for design rule violations, produce manufacturing data and documentation, and in addition ensure correct board connectivity by rejecting operator attempts to connect points that are not consistent with the connectivity defined in the netlist.

3. The most powerful class are the schematic driven, interactive layout systems described above, but enhanced with software (or special purpose hardware) for automatic component placement and automatic track routing. The automatic routines speed up the layout task. The best systems allow interactive monitoring and manual intervention in the autoplace and autoroute operations.

We shall only describe class 3 PCB CAD/CAM tools. A typical PCB layout tool will draw information about the design rules from a user written technology file. Data for specific component types will be extracted from a PCB component library. The connectivity to be routed (and possibly layout constraints imposed by the designer) will be available in the design input file, previously generated by compiling and linking the designer's schematics (as described in section 2.1).

5.1 Initial package and pin assignment

The designer's schematic usually employs functional elements (i.e. gates, flip-flops, ALUs, etc.) rather than complete packaged components containing multiple functional elements. The layout sys-
tem will first have to pack functional elements into specific component packages of the corresponding type, as indicated in Figure 19. This is done by assigning each logic element a package reference designator (e.g. IC24). Functional elements assigned to the same package are differentiated by assigning the elements' ports the physical package pin numbers of one of the elements within the package. Once assignment has been made, the schematic can be automatically back-annotated with the package reference designator and pin number information.

![Figure 19: Assignment of functional elements to packages.](image)

The initial assignment of elements to packages is usually made either pseudo-randomly in the order in which the elements are stored in the design database, or by grouping like elements that are interconnected on the schematic. Normally the designer can control critical assignments by pre-assigning the reference designators and pin numbers on the schematic and "freezing" them before running the initial assignment routines.

5.2 Component Placement

The objective of the placement function is to place components in a way that results in an easily solved routing task. At the same time it must satisfy other (possibly conflicting) considerations such as circuit performance (clock skew, cross-talk, minimal critical delay path, etc.), uniform distribution of heat dissipation during operation in order to avoid hot spots, sufficient clearance for an automatic insertion tool to be used, etc. Reference [34] gives a good overview of placement techniques.

Normally the user can interactively place critical components, or components that must be located in predefined positions (e.g. edge connectors). He can then use the autoplacement software to place the remaining components. Components are usually placed on a user defined placement grid. Once critical components are placed interactively they can be "frozen", preventing the placement optimisation software from moving them.

Regular structures like memory arrays, or bus-oriented structures, are best placed by hand. In these cases the best placement is obvious to the human operator, while the autoplacement software will consume considerable compute resources only to come up with a less satisfactory placement. Autoplacement is best suited to those parts of the design where there is no obvious regular layout topology.

5.2.1 Metrics for placement algorithms

Algorithmic placement uses a metric to evaluate the effectiveness of trial placements (quantified by a score (S)), and subsequently chooses the best trial placement. It will be necessary to perform
placement by hand in those cases where the algorithms do not use a metric that takes into account other constraints that are critical to the particular design. Metrics used to evaluate placement effectiveness can be divided into two major classes:

1. Simple metrics that ignore possible interactions between net routings.

2. Metrics that measure the likely interaction between net routings due to congestion of the board. These metrics often give placements of superior quality to those given by the first class.

The commonest metric in the first class is total wire length summed over all nets. Another metric in the first class models pin to pin connections as springs, tries to find the equilibrium positions of all components under the action of the springs, and then moves each component to the nearest available placement grid point. These class 1 metrics reduce the total length of wiring and therefore tend to minimise occupancy of available routing space, which in turn should make the routing problem easier to solve. However, they do nothing to prevent wiring density building up in congested areas, consequently leading to poor routing results.

The second class of metrics include, for example, measures such as the number of nets that cross a "cut" line. The well known rats nest display shown in Figure 20 gives a quick qualitative measure of the effectiveness of a placement by allowing visual identification of potential areas of routing congestion.

![Figure 20: The rats nest display identifies congested areas.](image)

5.2.2 Net-to-wire partitioning

Because placement must position components in a way that optimises board routability, it will be sensitive to the way in which the signal nets are partitioned into wires. Thus placement and net-to-wire partitioning are interwoven problems. Several net partitioning strategies are shown in Figure 21.
In order to reduce the computational complexity of the problem (which in principle requires nets to be repartitioned afresh for each trial placement evaluation), many systems use either the complete interconnection graph, or one particular partitioning for all trial placements. After a placement has been chosen, the nets may be repartitioned into the minimum spanning tree, or minimum length chain, by solving the travelling salesman problem [35]. After partitioning nets into wires, further placement optimisation may be profitable; in general the layout expert iterates between placement optimisation and net partitioning until he is satisfied that the board can be routed.

Some systems allow steiner tree routing, where a track can join onto an existing track belonging to the same net by making a T-junction. It turns out that the minimum spanning tree is already a near optimal partitioning for finding the minimal Steiner tree [35]. The result of the net partitioning step is a wirelist (not netlist) ready to drive the router.

5.2.3 Initial Placement Algorithms

Initial placement is made using the so called constructive placement algorithms, some of which are outlined below:

1. Random initial placement of components is sometimes used. It relies on the iterative placement improvement algorithms to converge to a good placement.

2. The cluster development class of algorithms choose the next component to be placed by selecting the one with the strongest coupling to already placed components. It is placed at the position that gives the best score using one of the metrics described above. Note that it does not use all the interconnect information available because as yet unplaced components do not contribute to the current component placement decision.

3. The min-cut class of algorithms considers all interconnections in parallel by dividing the set of components into two subsets such that the number of interconnections between the two subsets is minimised and the area of components assigned to each subset is approximately equal. The partitioning process is repeated on the subsets until each partition contains only one component. These algorithms have the advantage of using the interconnection information globally, top-down, deferring local decisions to last. They produce good results but are compute intensive.
4. Another type of global placement method places all components simultaneously by arranging them in a way that minimises the sum over all components of the directed force vectors representing the strength of inter-component coupling (the force is the number of connections between the pair multiplied by the distance between their centres). The components are then moved to the nearest available placement grid point.

5.2.4 Placement Improvement Algorithms

After initial placement the nets will partition into wires in a different sequence. Thus the initial placement will no longer be optimal for the new wiring sequence. Another reason why the initial placement is less than optimal is that components are placed either by taking into account only the information about already placed components, or by ignoring effects of different sized components, or the constraint that all components must fall on the user defined placement grid, etc. Thus initial placement is usually followed by an iterative placement improvement phase in which components are moved to new trial positions and the placement score is re-calculated. If the trial placement shows an improvement it is kept, otherwise the components are returned to their previous positions and a new trial placement is tried.

Some of the algorithms used for placement improvement are briefly described below:

1. The *Pairwise Interchange* algorithm selects a each component in turn and successively interchanges it with all other components. Each interchange is evaluated by calculating its score using the total wire length metric. The interchange with the lowest score is retained.

2. The *Force Directed Interchange* method uses the mechanical analogy of attaching a spring between components for each connection between them. The tension in the spring is proportional to the distance between the components. The resultant of adding all the force vectors defines a direction in which the component should move in order to reduce the total wire length. The pairwise interchange method is then applied uniquely with the neighbouring components in the direction of the resultant force.

3. In the *Force Directed Relaxation* technique, a component is selected and moved to the placement grid position nearest to its "zero force" equilibrium position. Any other component that was occupying this grid position is itself moved to a new zero force position. The process is repeated until the chain of displacements is broken when an unoccupied zero force position is found. The new trial placement is evaluated by calculating its score and is retained if the score improved over the previous placement’s score. All components are selected in turn for relaxation.

4. The *Simulated Annealing* method avoids a problem that is common to all the other optimisation methods described before. All the other methods only accept a trial placement if its score is better than the current placement; they usually do not converge to the global minimum, but get stuck in some local minimum. The simulated annealing algorithm allows the minimisation to climb out of a local minimum by accepting poorer scoring trial placements with a probability $e^{-\Delta S/T}$ (where $\Delta S$ is the (positive) change in score between the trial placement and the current placement, and $T$ is the "temperature"). Starting from an initial placement and an initial temperature $T_0$, components are moved at random and the temperature is slowly reduced. If the temperature is not reduced too quickly, the placement will gradually move towards the global minimum. Since the process is probabilistic, convergence to the global minimum is only guaranteed as the number of iterations tends to infinity. In practice good results are obtained for a tractable number of iterations, although some claim that they are not superior to those achieved by other methods. Despite relatively long run times, this method is very popular.
5.3 Package and Pin Assignment Optimisation

The initial packaging process described in section 5.1 assigns schematic elements (i.e. gates, flip-flops, etc.) to packages (each package may contain multiple identical elements) and assigns package pin numbers to the ports of the elements. These assignments can be optimised to improve the routability of the board. Elements can be iteratively swapped between placed packages of the same component type, or they can be swapped with other elements within the same package. Permuting logically equivalent pins (e.g. the two input pins of an AND gate) can also improve board routability. Thus, because swapping element or pin assignments results in changes in the interconnection topology, just as it does with package swapping, the assignments are usually optimised using techniques similar to iterative placement optimisation. In fact, placement and assignment decisions are all related and ideally should all be minimised together. In practice, in order to reduce the computational complexity of the problem, they are treated as separate optimisation problems.

As with package placement optimisation, after assignment optimisation the nets will need to be repartitioned into wires (this is not usually done dynamically as part of the optimisation process). Several iterations of assignment optimisation and net partitioning may be necessary before the board is judged to be routable.

5.4 Routing

After placement, assignment optimisation and net partitioning have been iterated several times the designer will start routing the connections. Routing can either be made interactively, or by invoking automatic routing software. Many advanced PCB CAD tools are equipped with several different routing algorithms, some being fast but less effective at finding routes, others being slower but more effective, and yet others being optimised for routing particular topologies efficiently. Results from many of these algorithms can be controlled by defining cost functions, limits or other parameters as described in this section and for example in section 5.4.4 An algorithm can make several passes at the problem, each pass using different cost functions or control parameters. The operation of a routing algorithm can often be restricted to certain areas of the board, chosen so that they contain topologies for which the algorithm is optimised, or for which the pass parameters have been specially chosen. Most autorouters route on a pair of routing layers, using one layer for vertically oriented tracks and the other for horizontally oriented tracks. However, some routers approach a 3-dimensional routing capability by allowing routing on an arbitrary number of layers simultaneously (in practice the number of simultaneous routing layers is limited by run time and memory requirements and is typically not more than 4 or 6).

Critical routes can be autorouted first, or they can be interactively prerouted. Some autorouters run in batch mode and give the user little or no feedback as to how the routing is progressing. More user friendly autorouters display routes graphically as they are found and allow the user to interact with them by interrupting them, changing parameters or modifying their results and then allowing them to continue. These re-entrant routers should give better results and higher productivity than a batch router by combining the speed and accuracy of the computer with the experience and intuition of a layout specialist.

For batch autorouters, the user plans a routing strategy for tackling the particular routing problems of each board, in which he defines the order in which the algorithms will be applied, the routing parameters for each pass, etc. Setting up effective routing strategies requires much previous experience with the autorouter. However, most systems provide default strategies that have been found to give reasonable results on typical designs.

One of the choices systems usually give to the layout technician is on the order in which the router will attempt to make the connections. Typical ordering options are:

1. Short connections first.
2. Long connections first.

3. Connections from a selected component, or group, first.

4. Connections to components selected by order of their placement:
   a. from left to right
   b. from right to left
   c. from top to bottom
   d. from bottom to top

5. By order of local estimated routing density (route to "surrounded" pins before they are boxed in by other tracks).

6. Autorouting one net at a time (using interactive graphics to select a net to be "one shot" routed).

7. etc.

Ordering strategies can effect routing results, but there appears to be no preferred ordering that is good for all types of layout. The main types of routing algorithm encountered in PCB CAD systems are discussed next.

5.4.1 Pattern Routers

The pattern router is used to route regular topologies of the kind found in memory arrays or bus structures, as shown in Figure 22.

![Pattern Routers Example]

*Figure 22:* Memory arrays are easily routed by a Pattern Router.

In this case the two templates A and B were sufficient.

The router stores a certain number of standard routing templates suitable for applying in these cases. The list of templates is searched until one is found that matches the point to point connection to be made and bypasses any intervening obstructions. This algorithm is very fast. However, there is relatively little advantage in using this type of algorithm when the interactive routing software allows step and repeat operations. In step and repeat methods, the pattern is manually routed once and can then be copied many times over between different pin pairs (by selecting a new start point to repeat the pattern with a simple click of the mouse button). Arrays can be interactively routed even quicker when step and repeat can be applied with a group of tracks (e.g. to replicate all the connections between two ICs of the array in one operation).
5.4.2 Line Probe Routers

The line probe router shown in Figure 23 sends out a probe line from the source pin (S) along the longest side of the smallest rectangle enclosing the source and target (T) pins. The probe line either reaches the far side of the rectangle, or it encounters an obstacle (e.g. a pad, a previously routed track, a user-defined 'keep out' area, etc.). It then proceeds by selecting one of the following options:

1. Add a via and continue on another layer.
2. Change direction on the current layer.
3. Backtrack and probe in another direction.

This strategy is applied repeatedly until the probe hits the target pin, or is terminated by some other criteria (too many bends or vias, CPU time allotted to each connection search exceeded, etc.). Each of the possible actions that can be taken at a decision point can be assigned a priority (either hard wired in the code, or under user control). There are many variations on this basic algorithm (for example allowing the probe line to overshoot the target pin in order to find possible paths that approach the target from behind, or starting the search simultaneously from both pins and continuing until the two probe lines intersect.

The quality of the results will depend on the way the possible actions are prioritised at the search decision points. The line probe algorithm executes rapidly because checking for intersection with obstructions can be accelerated by using a binary search technique on lists ordered by co-ordinate values [35]. The major shortcoming with this class of algorithms is that the decisions made at each point where the probe line is blocked are based on a set of predetermined rules; there is no comparison of possible alternatives with an eventual choice of the best candidate. The algorithm tends to produce routes that unnecessarily obstruct routing channels and box in other pins.

![Figure 23: Simple line probe routing algorithm.](image)

Using the priority of decisions shown, the track is found by the line probes 1, 6 and 7).

5.4.3 Maze Routers

In this approach, originally due to Lee [36], each routing layer of the board is divided into a large number of cells, as shown in Figure 24. Cells are marked as "occupied" at those positions where there are plated-through holes for component pins, tracks that have already been routed, or user-defined "keep out" areas. The simple maze search algorithm then searches for a path through the maze in two steps:

1. Starting from the source pin (S), adjacent, unoccupied cells in the vertical or horizontal direction are provisionally marked as being occupied by the current search. This
step is then repeated using each cell that was marked in the previous step as a source for the second step. By repeating the operation a "wave front" is made to flood through all accessible parts of the maze. Each newly marked cell is stamped with the iteration number of the wave front expansion process. The shortest possible path is found when the wave front first reaches the target pin. However this path is not necessarily the "best" route because length is not the only criteria for judging its quality.

2. The router now enters a back tracing phase, in which it traces a path backwards from the target pin to the source pin, laying down copper as it does so and marking the corresponding cells as definitively occupied. Backtracing is made by looking for an adjacent cell having a search iteration number one less than that of the current cell. When more than one candidate is available, a choice is made by applying simple rules (e.g. continue in the same direction as the last step).

The maze router can be adapted to find diagonal tracks by allowing the wave front to expand to diagonally adjacent neighbours. It can be applied to routing on several layers simultaneously by defining a 3-dimensional maze and allowing the wave front to expand to neighbouring cells in the third dimension in order to explore paths which change from one layer to another.

![Wave expansion from source to target.](image1)

**Figure 24:** The Maze search algorithm floods a wave front from source to target pin.

Wave expansion is first made from the source to the target pin. In the example the target is reached on the 13th step. The track is laid down by retracing from the target using simple rules. In the example, track A is found by starting the retracing downwards and changing direction only when blocked. Track B would be found if the retracing made its first step to the left instead of downwards.

Search times can be very long, depending on the search area and the size of the cells. Search times can be reduced by starting the search from the pin that is furthest from the centre of the search area (wave front expansion will be cutoff by the search area boundaries in some directions sooner than when the wave front starts from the centre of the area). Another technique starts the search simultaneously from both pins and stops when the two wave fronts collide. Yet another technique limits the search area to the smallest rectangle enclosing the source and target pins, and only increases the search area when a path cannot be found within the rectangle. Execution times may be reduced and routing completion rates increased by allowing the search to connect to any previously routed track belonging to the same net. Steiner tree routing is simply implemented by defining all cells on the previously routed track(s) as targets to be searched for in parallel. The first target hit makes the shortest possible T-junction connection.
Although the simple maze search algorithm will find a path if one exists, results are unsatisfactory either because they include too many bends, or because horizontal track segments block vertical routing channels (and vice-a-versa), or because too many vias are inserted to change layer (each via blocks routing channels on all layers of a multi-layer board and increases board manufacturing costs).

5.4.4 Costed Maze Routers

The costed maze search algorithm is a development of the simple maze search algorithm due to Rubin [37]. Each step in the wave front expansion process adds a cost increment to a running cost that is propagated from cell to cell. Wave front expansion is only done from the cell on the wave front that has the current minimum cost. If several wave front cells have the same minimum cost, they are expanded in reverse order (i.e. the last one that was assigned a cost is expanded first). In this way the wave front spreads out quickest in directions of lowest cost gradient, and the minimum cost (but not necessarily the shortest) path reaches the target first. In order to facilitate back tracing of the path, each cell remembers the direction from which the wave front entered it.

By assigning a small cost increment for steps in one direction and a large cost increment for steps in the orthogonal direction one can bias wiring on a given layer to run in a preferred direction. A pair of layers will be assigned orthogonal preferred wiring directions and routed simultaneously, with paths changing from one layer to the other when a change of direction is necessary. In this way channel blocking by wires running orthogonal to the preferred wiring direction is eliminated. By balancing the cost of changing layer against the cost of stepping orthogonal to the preferred wiring direction, one can trade the number of routing channels lost through inserting a via against the number lost by allowing short wire lengths orthogonal to a layer’s preferred wiring direction.

Most commercially available systems use a costed maze search algorithm because of its flexibility. They offer a large number of cost factors that can be set by the user in order to control routing results. When the cost factors are well chosen, the costed maze router gives better results than the line probe class of algorithms. Note that, by appropriately choosing the cell sizes, track-to-track and track-to-pad clearances are automatically guaranteed and no design rule checking is required after autorouting. Runtimes can be long (typically tens of hours of CPU time for large, dense boards with a fine cell grid). The host machine will also require a large real memory (typically in excess of 8MB) in order to accommodate the cell array without catastrophic loss of performance due to virtual memory paging to/from mass storage. As with compute intensive simulation tasks, attempts have been made to construct special purpose hardware for acceleration of maze routing algorithms (see for example references [38] and [39]).

5.4.5 Completing Routing by Rip-up and Re-route

Autorouters usually lay down new paths without taking into account their effect on the routability of connections still to be routed. Once routing channel occupancy has risen to the point where router success rates start to drop off dramatically, it may be worthwhile reviewing previous routes and perhaps modifying them in order to open up channels for routing the remaining connections. Figure 25 shows an example where the blocked route AB can be routed after the blocking route CD is re-routed along an alternative path.

The rip-up and re-route process can be performed interactively by the layout technician, or semi-automatically with a re-entrant router by manually moving the obstructing route CD and then invoking the autorouter to “one-shot” route the connection AB. Some systems offer automatic rip up and re-route algorithms that identify blocking tracks, rip them up and re-route them along a different path by using different cost functions than those used in the initial router pass. Blocking tracks are ripped up and re-routed until the blocked track can be successfully routed. Sometimes the ripped up track cannot be re-routed and the rip-up and re-route algorithm is called recursively in order to route the ripped up blocking track. Recursive rip up and re-route does not always converge, so one can end up increasing the number of unfinished routes.

224
Batch routers that use multi-pass rip-up and re-route techniques, use the results of one pass as input to the next pass. Each pass reworks the previous one using different routing parameter values. Although individual passes may rip up more tracks than they re-route, the tendency is to converge. Run times for large boards are measured in days. If the process does not achieve 100 percent routing success, it can be very difficult to route the few remaining connections by hand.

An alternative approach to completing the routes left over by the autorouter is to let the autorouter route tracks as close to the target pin as it can get. It may be easier for the operator to interactively finish the dangling track than to route the complete connection. If this proves to be too difficult the operator may try modifying component placement and re-routing the board. A third option for squeezing in the remaining tracks is to re-route the layer using finer tracks and smaller clearances so that 2 or 3 tracks can be routed between pins of an IC instead of 1 or 2 tracks. If all else fails, extra routing layers must be used.

5.4.6 Choice of Routing Grid

A consequence of developments in packaging technology is that many board designs now contain a mixture of packages with different lead spacings\textsuperscript{7}. In these cases it is difficult to find a uniform grid on which all component pins fall, and which guarantees sufficient clearance between tracks and pins in all cases and at the same time does not result in the router using too much memory or processor time.

Figure 26 shows how the use of non-uniform grids allows high density routing (2 or 3 tracks between pins with 100 mil centre-to-centre spacing) without the memory space and processing time penalties that would be incurred by using a uniform high density grid.

The variable grid technique uses different grid sizes in different parts of the board, each grid being chosen optimally for the local routing problem. Other systems can switch from the normal grid to a finer routing grid whenever the router is working near an off-grid pin.

Gridless routers which check design rules (clearances with neighbouring tracks, pads etc.) as they route have also been tried. However, the many complex geometry calculations involved cause them to be very compute intensive and slow in practice. In addition the absence of a grid makes interactive routing without violating design rules very difficult.

\textsuperscript{7} e.g. dual in-line packages (DIPs) usually have 100 mil lead spacing, whereas surface mounted devices (SMDs) usually have 50 mil spacing and some connectors have metric pin spacings.
5.5 **Postprocessing, Manufacturing Data and Documentation**

Once the board has been placed and routed there follows a postprocessing step that cleans up the results of the autorouter in order to improve board manufacturing yield or reduce manufacturing costs. This is either done by a batch program, interactively, or by using a mixture of batch and interactive processing. Typical postprocessing actions are:

1. Replace "staircase" routing by smooth diagonal tracks.
2. Move tracks to increase clearances between tracks or between tracks and pads.
3. Suppress unnecessary vias (drilling vias is one of the most expensive operations in manufacturing a board; they are also often the source of manufacturing defects).

Often a final pass through a design rule checking program is made in order to catch any design rule violations introduced in postprocessing (or by bugs in the autorouter).

The next step is to produce data and documentation for manufacturing:

1. Numerical control (NC) tapes for drilling machines.
2. NC tapes for photoplotting:
   a. Artwork
   b. Solder resist masks
   c. Silk-screen printing, etc.
3. Data for driving Automatic Component Insertion Equipment.
4. Data for set up of board test equipment.
5. Pen plots, parts lists, etc.

The production of NC tapes is done by programs that first optimise the sequence of machine operations in order to minimise machine operation time and wear, and then produce data in the required format. In principle all the information exists in the design database, thus the documentation can be produced with little manual effort and without errors.
Finally, if gate or pin swapping were made during the layout of the board, the changed package reference identifiers and pin numbers must be back-annotated to the logic schematics. The better integrated schematic capture and layout systems automate the back-annotation step, but many of the cheaper systems leave this job to be done manually. One disadvantage of working with schematic capture and layout systems from different vendors is that in many cases there is no support for automatic back-annotation.

5.6 Software for other board technologies

Although printed circuit board technology is used for the production of the vast majority of boards, there are other board manufacturing technologies that have their place. The wire wrap technique is used at CERN where large dense boards are prototyped or only made in very small quantities. As the wire wrap technique uses a machine to make point to point connections using insulated wire there is no difficult routing problem to be solved. The NC tape for driving the wire wrap machine is produced by a relatively simple netlist driven program [22]. Although printed circuit board manufacturing costs are lower, for very small volumes the overall cost (layout design and manufacturing costs) is smaller for the wire wrap technique. The wire wrap technique has the advantage that rewiring during prototype debugging is very quick, easy and reliable.

The Multiwire technology uses insulated wire that is routed between obstructing pins and laid down by a special machine. Component placement, routing, analysis and production of manufacturing and test data are supported by a special Multiwire design software package [40]. As the insulated wires can cross over each other no vias are required for changing from one wiring layer to another layer. These two characteristics mean that a layer of Multiwire routing can be much denser than a typical layer of printed circuit board routing. Like the wire wrap technique, Multiwire technology is suitable for small volumes. The high density wiring also minimises the total number of wiring layers and, in the case of controlled impedance ECL designs (where each layer has to be spaced at a certain distance from a ground plane in order to provide the correct transmission line impedance) helps to contain the finished board thickness within allowed limits.

5.7 Practical experience with Board Layout CAD

At CERN we have experience with two different types of PCB layout system; one uses a re-entrant autorouter providing powerful interaction and guidance possibilities, the other uses a multi-pass, rip-up and re-route, batch autorouter. Users of both types of system agree that a good component placement is the key to successful routing. The placement task is approached in a similar fashion by users of both types of system. A typical placement strategy is as follows:

1. Place and lock in place connectors and other critical components.

2. Automatic initial placement of the remaining components.

3. Interactive adjustment of placement in critical areas (e.g. near connectors or around busses) to improve routability. The interactive use of the rats nest display or force vectors display gives dynamic, real-time, graphical feedback on the placement adjustments.

4. The adjusted components are locked in place, and automatic placement optimisation is used to improve the placement of the remaining components. One or more further iterations through steps 3 and 4 may be made.

5. Obvious candidates for manual gate swapping are made on the basis of the rats nest display.
6. The "well swapped" gates are locked and optimisation by automatic gate swapping is invoked.

7. Optimisation by pin swapping is made manually and/or automatically.

8. Steps 5 through 7 are repeated one or more times if required.

The job is now ready for routing. In the case of the re-entrant router a typical strategy is:

1. Decide on the number of routing layers for the job. A rule of thumb allows this to be predicted from the average number of pins per unit area and the board size.

2. Route power and ground connections by hand.

3. Manually route the most critical areas (near connectors or in bus structures).

4. Use the autorouter in re-entrant mode to route the remaining critical areas with operator guidance and interaction where necessary.

5. Manual clean up of tracks (shorten long tracks, display tracks that are likely to block the remaining connections from being routed).

6. Batch autoroute the remainder of the board using multiple passes with each pass gradually relaxing the constraints used in the previous pass.

If only a small number of connections fail to be routed, the layout technician will interactively finish the routing; otherwise he chooses one of the following options:

1. Review the placement and re-route the board.

2. Unroute part of the board and re-route it using a finer grid.

3. Add an extra pair of routing layers to the job.

The strategy is that the technician recognises regular structure in the layout and is better able to place and route those parts manually with assistance from the machine\(^8\), while on the other hand the machine is more efficient in optimising layout in those areas with little regularity of structure.

In the case of the multi-pass, rip-up and re-route batch autorouter a batch job is submitted to a dedicated routing server. This may run for several hours, overnight, or even for several days. During the batch autorouting process the operator works on other jobs. If the batch autorouter does not achieve 100 per cent routing completion it will be extremely difficult for the technician to interactively finish the routing (the powerful rip-up-and-re-route algorithm leaves very little routing space unused); in this case the technician will choose one of the following options:

1. Improve the placement and try to re-route the board.

2. Route with different design rules.

3. Add an extra pair of routing layers to the job.

---

\(^8\) assistance in the form of quick graphical feedback of changes, graphical presentation of several possible routes found by the autorouter for final selection by the technician, on-line design rule checking etc.
In our experience autorouters are useful for non-critical digital TTL designs with random structured layouts. Their results are easily surpassed in quality by an experienced human operator whenever the layout problem contains some degree of regularity. Furthermore, they do not (yet) give satisfactory results for high speed ECL layouts where transmission line characteristics are important. They are also unsuitable for most analogue designs. However, the autorouter is only one aspect of the layout system, and although many jobs may not make much use of the autorouter, they do benefit from other aspects of the CAD system.

One should not forget that layout systems are very sophisticated and complex. Installation of a new system will be followed by a fairly long period where technicians learn to use it proficiently (especially if they have no previous experience with CAD systems) and it is tailored to the users' environment by capturing in technology definition files the local design rules and other technology related parameters reflecting local design practices. Component libraries have to be built up, or adapted to local standards.

Another area that usually requires considerable effort to set up initially is the manufacturing interface. The plethora of incompatible machinery in use world-wide requires the CAD vendor to support interfaces to many different brands of machinery. Inevitably the quality of the average interface package suffers; they are often user-unfriendly, error prone to operate, and contain residual bugs that need to be identified and corrected before a smoothly operating manufacturing interface is established.

Users that mix schematic capture and layout systems from different vendors will probably encounter a number of additional problems related to the fact that the two packages work from different component libraries. This is especially true when the interface supports automatic back annotation. The component library data will need to be modified so that all component data (e.g. gate and pin swapping rules) are compatible between the two sets of libraries. In order to preserve library compatibility, guidelines for creation of new library components will need to be worked out and put into practice. Ideally the responsibility for library development and maintenance should be concentrated into the hands of a library manager or small group of experts.

A large number of problems have to be solved initially and important organisational and management changes will need to be carried out. For this reason it typically takes between 6 and 12 months before a new system comes up to full productivity.


Lack of space precludes a detailed discussion of layout tools for integrated circuits. Reference [2] can be consulted for more details. The semi-custom IC layout problem is in principle similar to the PCB layout problem, but nevertheless differs from it in several important aspects.

Figure 27 shows three common architectures used for semi-custom ASIC layouts. In semi-custom gate array design the macros blocks (the logical building blocks, or components, supplied in the vendor's library) can be placed only at predefined positions corresponding to the underlying prediffused, uncommitted transistor array. The gate-array is customised by designing metal layer masks that define the inter-transistor connections that turn an uncommitted array of transistors into a specific functional macro block (this pattern is supplied in the vendor's macro library) and at the same time define the metal interconnect paths between the macro blocks. In the case of standard cell

---

9 Examples of other benefits of layout CAD systems are: The autoplacement and interactive placement functions, connectivity driven layout eliminating wiring errors, design rule checking ensuring manufacturability, automatic production of error-free manufacturing data and documentation, ease of modifications to the design, etc.
designs prediffused wafers are not used, so cells (i.e. pre-designed functional blocks) can be placed with greater, but not complete, freedom. Usually all the cells have the same height and have to be laid out in rows as shown in Figure 27(a), each cell abutting its neighbour, and sometimes alternate rows having their cells mirrored about the row axis with respect to the neighbouring rows.\(^{10}\)

For channeled gate arrays (Figure 27(b)) routing is restricted to run in channels of fixed width. For standard cell designs the channel width is determined by the user or automatically by the autorouting software, the cell rows being placed so that sufficient routing space is available to guarantee the successful routing of the design. Small and medium sized gate arrays use 1 or 2 (sometimes 3) metal layers for routing, while standard cell layouts use a metal layer to route along the length of the channel, and route across the channel on the polysilicon layer.\(^{11}\) Polysilicon has a much higher resistance than metal, so routing lengths in polysilicon must be kept as small as possible. The placement optimisation and autorouting algorithms are optimised for solving the channel routing problem efficiently.

The relative slowness of global communication on silicon (see section 1.4) makes good placement essential. A class of software tools known as floor planners can be used to approach the initial placement problem hierarchically. First the major functional blocks are placed and their shapes modified in a way that optimises connections between blocks. Each functional block can then be laid out internally, using a hierarchy of one or more placement steps.

The existence of fixed routing channels in gate arrays leads some systems to adopt a hierarchical approach to routing. Connections are first assigned to run in specific channels, then the assignment of connections to channels is optimised globally before the detailed routing within channels is attempted. Global routing prevents inefficient use of channels and increases the effective gate utilisation (the fraction of available gates that can actually be used to implement logic and yet still be routed with the fixed routing channel resources).

In general one cannot ignore the effect on system performance of the capacitive load (and for polysilicon the resistance) of the interconnections. Most layout suites therefore include an extractor

---

\(^{10}\) When these layout constraints are observed, power is automatically distributed to all cells in the row without need for explicit power routing, and the danger of CMOS latch up is avoided.

\(^{11}\) The polysilicon layer cannot be used to route over a cell because it is used inside the cell to build transistors. Connections that must pass over a cell row can do so by using a special feed-through cell explicitly placed in the row by the designer.
that calculates the capacitance (and sometimes the resistance) of each interconnection, and feeds this
data back into the logic simulator as an effective extra delay. Post-layout simulation gives a more real-
istic estimate of system performance.

For large gate arrays the restrictions of the channeled array architecture make layout very difficult.
For this reason the large arrays have abandoned the channeled architecture in favour of the so called
sea-of-gates architecture shown in Figure 27(c). This consists of a uniform array of prediffused cells
with no space explicitly reserved for routing. Cells can be used for implementing logic, or they can be
"sacrificed" for routing.

Layout generators exist for the automatic generation of specific types of functional module (e.g.
PLA generators, RAM generators, etc.). Silicon compilers exist for the generation of layouts from
high level descriptions, but these topics are outside the scope of this paper (see reference [1]).

7. Conclusions

Software tools for assisting the design process are now powerful, user-friendly and even becoming
affordable. In some areas they are already mandatory (e.g. the moderately priced and quite sophisticat-
ed packages available on personal computers for the design of programmable logic devices, or logic cell
arrays are effectively the only path to the use of these important devices). In the future CAE and
CAD will be the key to the use of the most advanced technologies for applications in high energy
physics.

On the other hand there are still a number of areas that are not well served by existing tools. In
addition their efficient use requires some changes in design methods, work styles and management
techniques. Reference [41] gives an excellent discussion of these issues as they apply to the electronics
activities characteristic of high energy physics laboratories. We can however be quite confident that
the rapid pace of improvement in design tools and methods will continue into the future, and that
most of the existing gaps will be soon filled.

8. Acknowledgements

I am grateful to the many colleagues who have contributed to the environment that stimulated
the acquisition of the information presented in this paper and have shared with me their practical
experience with the tools. Special thanks go to Hans Anders, Francois Bourgeois, Serge Brobecker,
Endre Futo, Burkhard Heck, Joop Joosten, Bert vanKoningsveld, Sandro Marchioro, Alain Monfort,
Paulo Moreira, Ludwig Pregernig, Karl Zumbrock and Rudi Zurbuchen.

9. References

1. J. Rabaey, "Silicon Compilation and Design Synthesis for Digital Systems", in these proceed-
ings.


3. M.Barbacci, G.Barnes, R.Cattell, D.Siewiorek, "The ISPS Computer Description Language",


15. SABER Reference Manual, Analogy Inc., P.O. Box 1669, Beaverton, OR 97075-1669, USA.


24. F. Bourgeois, EF Division, CERN, private communication.

232
40. PCK Technology Division, Kollmorgen Corporation, the MDS and DWIRS Multiwire design software.
ABSTRACT

In this paper, an overview will be given of the current activities in the field of the automatization of the design process for digital integrated circuits. This field is now commonly called design synthesis. The idea of design synthesis is to automatically generate the physical layout of a circuit starting from a textual or graphical specification of the circuit’s expected behavior. The design synthesis process can be divided into two phases. In a first step, the specification of the circuit is translated into a chip architecture. This phase is generally called behavioral synthesis, since it translates the behavioral specification of the circuit into a more structural description. In the second phase (called silicon compilation or structural synthesis), the final circuit layout is generated starting from the architectural definition. Common tools used at this level are module generators and floorplanners. The paper will discuss a number of the existing synthesis systems (including examples) in more detail.

1. Introduction

The advances in integrated circuit technology encountered in the last couple of decades have created a real 'design crisis'. The complexity of the circuitry, which can be implemented on a single chip, approaches the system level. Nowadays, it is almost possible to implement a complete computer, telephone or digital audio system on a single chip. On the other hand however, the lifetime of such a device becomes shorter and shorter. The lifetime of a circuit for consumer applications for instance is a couple of years. This should be compared with the TTL devices of the sixties, which are still used at present. This situation becomes even worse when considering the so called ASIC’s (Application Specific Integrated Circuits). An ASIC can be considered as a special purpose device, designed to perform one particular task (this in contrast with general purpose devices such as micro-processors and memories, which can be used for a wide variety of applications). A lot of those ASIC’s serve as partial replacements for the printed circuit boards, required in earlier systems. The typical lifetime of such a device is at most a couple of years. In this way, we approach a situation in which the design time of a device almost approaches the lifetime. Hence the design crisis.

This explains the intensive research and development activities of the last ten years targeted at speeding up the design process. In the late seventies, Johannsen [Joh79] introduced the idea of silicon compilation. He used this term to describe the automatic generation of parameterizable pieces of layout. Since then, the term silicon compilation has been used and misused to describe almost every single phase in the design automation process. Currently, there seems to grow an agreement to restrict the scope of silicon compilation to the original definition of Johannsen. The overall process of the automatic generation of the layout of a circuit starting from a high level behavioral specification is called design synthesis.
After a global overview of the design synthesis process, I will discuss the different design abstractions and the associated tools in more detail. Existing tools and compilers will be used to explain the basic synthesis operations. The paper will conclude with a projection of where and how far synthesis might go in the future.

2. The Design Synthesis Process: An Overview

In order to define and differentiate the different steps and levels in the design synthesis process, let us look at one particular design scenario, as illustrated in Figure 1. Suppose we have to design a device, whose function it is to improve the contrast in an image, received in a line-by-line fashion at a rate of 10 Mbps/second.

- The first task of the designer is to select an algorithm, which is likely to meet the specifications and whose complexity is such that an integrated implementation is feasible. In the case of the image contrast enhancement, a variety of algorithms is available in literature, spanning a wide range of complexity and quality. The selection of a suitable algorithm is a complicated task and relies heavily on the experience of the designer. Most of the time, the process is iterative: an algorithm is selected, a

![Diagram of design synthesis process]

Figure 1: Representation levels in the design synthesis process.
rough estimation of the implementation cost is made and the performance is analyzed using extensive simulation. Based on this information, the algorithm is either rejected, accepted or modified (which is most often the case). For our example, we have selected a histogram based algorithm: a histogram of the picture is taking, plotting the number of pixels/intensity level. The histogram information can be used to adjust the pixel intensities so that a better contrast is obtained. No substantial automatization of this part of the design process has been achieved at present and I believe that this will be true in the foreseeable future. Some topics which are being addressed at present are the automatic estimation of the implementation cost of a given algorithm and the speedup of the algorithmic simulation and verification process.

- As a result of the algorithmic selection process, a so called behavioral specification of the circuit is obtained. This (textual or graphical) definition describes the functionality of the device without describing how is going to be implemented. The behavioral synthesis process will map this description into a chip architecture, being a composition of blocks such as memories, data paths, controllers and input/output devices. This description is generally called structural, since it describes the structural composition of the device under design. The synthesis process analyses the complexity of the algorithm, estimates the amount of hardware needed, defines the controller structure and generates a potential chip structure (as shown in Figure 1b). Behavioral synthesis is still in its infancy. Efficient synthesis is currently only possible for very restricted application areas. However, the basic understanding of the issues involved is growing and we will see a broadening of the scope in the very near future.

In the case of the contrast enhancement chip, the input to the behavioral synthesis step is a description of the histogram algorithm using a programming language such as C or Pascal. The selected architecture consists of a number of memories, a controller (based on a Programmable Logic Array) and an arithmetic and address arithmetic unit. Besides the definition of these basic building blocks and their parameters, the structural description also contains a set of netlists, defining the connectivity in between the blocks. Notice that a structural description for a larger design is normally constructed in a very hierarchical fashion.

- The last step in the synthesis process, which we will call structural synthesis, corresponds to Johannsen's original definition for silicon compilation. It generates the final layout artwork (also called the physical representation) from the structural definition obtained in the behavioral synthesis step. This is where the majority of the development work was situated in the last decade and this has resulted in a wide variety of industrial available tools (such as described in [Bur88], [Law88], [Che88]). An example of such a structural tool is a generator, which produces the layout of a multiplier, given the word length of multiplier and multiplicand. Structural synthesis also includes floor planning, placement and routing and cell generation. Given the large amounts of data involved, good data base management and design flow control is also important at this level. The final layout of our contour enhancement chip can be generated using a module generator for the memories and the controller section, a data path compiler for the arithmetic and address arithmetic units and a floor planner to place and route the individual components.

It must be understood that the above design flow is simplified. Most of the times, the design synthesis process is an iterative one. Depending upon the results of a syn-
thesis step, the designer might decide to backtrack and modify the descriptions at either the structural, behavioral or specification level.

In the rest of the text, the structural and the behavioral synthesis steps will be discussed in somewhat more detail. We will use the LAGER system, developed in Berkeley [Rab85] [Rue86] and the CATHEDRAL systems from IMEC, Leuven [Dem86] [Rab88], as demonstrators for the basic synthesis techniques. Both LAGER and CATHEDRAL are synthesis systems, targeted at applications in the field of Digital Signal Processing (DSP). The techniques used in both systems are however applicable in a much wider context.

3. Silicon Compilation or Structural Synthesis or Silicon Assembly

As mentioned, the silicon compilation process translates a structural description of a design into a physical layout. A structural description defines the composition of a circuit in terms of the composing building blocks and their interconnections (netlists). Each building block (or module) is specified by a number of parameters, which may be word lengths, definitions of the operators or boolean equations. For larger designs, those descriptions tend to be set up in a hierarchical fashion.

An example of a chip structure (sometimes also called architecture) is given in Figure 2, where the structure of a simple processor chip is defined. At the highest hierarchy level, the chip is composed of a controller, an ALU (Arithmetic Logic Unit) and a RAM (Random Access Memory). The ALU consists of two registers, an adder and a shifter, while the controller is composed of a program counter, a ROM (Read Only Memory) and some random logic. The processor word length is 16 bits, which is also true for the RAM and the ALU.

A variety of languages, called hardware description languages or HDL’s are available to describe a circuit at the structural level. Well known examples are VHDL [Lip86] and EDA [Mor85]. In the LAGER system, a language called SDL (Structural Description Language) is used to describe the chip architecture. The structural compo-
sition at the processor level (Figure 2) is described by the sdl file of Figure 3. Notice the basic elements of the structural description level: parameters passed from the next higher hierarchy level, composing subcells with their parameters, netlists and selection of the generation methodology.

```
(parent-cell processor)

; Definition of parameters
(parameters WordLength MemoryDim)

; Definition of the composing subcells (and instantiation of their parameters)
(subcells (Controller CONTROL)
  (DataPath ALU ((NrOfBits WordLength)))
  (Memory RAM ((Dim MemoryDim) (Width WordLength))))

; Netlists
(ioBus (NETWIDTH WordLength) ((parent In) (ALU In)))
(DBus (NETWIDTH WordLength) ((ALU Out) (RAM Data)))

; Selection of layout generator for this level.
; A Floorplanner called Flint is selected.
(layout-generator Flint)

(end-sdl)
```

Fig. 3: Structural description of the processor hierarchy level for the example of Figure 2.

In general, the structural synthesis tools can be divided in three main classes: cell generators, module generators and placement and routing tools. A cell compiler generates the physical layout of a leaf-cell (such as an adder, register or NAND-gate) starting from a transistor diagram or boolean equations. Those leaf-cells can be combined into larger entities such as memories, PLA's, multipliers or data paths using module generators. Multiple modules can be combined into a chip or sub-circuit using floor planners and placement and routing tools. Every single design most likely will use a variety of structural compilers. Below we will try to give you an overview of the available compiler tools. This overview is however by no means complete.

3.1. Cell Generators

Until recently, the leaf cells of a VLSI circuit were invariably drawn manually on a graphical workstation. This becomes increasingly expensive, since those cells have to be completely redesigned with every major technology change. It must be realized that the efforts to generate, maintain and document a comprehensive cell library are excessive. A number of approaches, which make it possible to generate technology independent cells and also speed up the cell design time are currently available. The techniques presented below are arranged in order of increasing automatization (and quite naturally decreasing circuit density).
• Symbolic Layout: This layout style is the closest to the manual technique. The designer is still responsible for the placement and the routing of the transistors, but defines only the relative positions of the devices and the wires. Symbols can be used to represent devices such as transistors, wires and connectors (Figure 4a and b). A compiler (also called compactor) will translate this relative placement into an absolute one, taking into account the design rules, a set of cell parameters and user defined constraints. Another advantage of symbolic cells is that the cell shapes can later be adapted to fit within the environment in which they will be used. This is demonstrated in Figure 4c, where the shape of the cells in a data path are adapted in such a way that the terminals connecting neighboring cells neatly abut.

The input to the system can be either graphical ([Hsu79], [Wes85], [Cro87]) or textual ([Bur88]). In the textual case, the symbolic layout tool is often called a procedural compiler, since it describes a cell as a piece of procedural code. An example of such a system is the GDT system of SCS [Bur88]. GDT uses a procedural language, called L, to describe the cell composition in function of elements and parameterizable constraints. Figure 5 gives an example of an inverter described using the L language. The sym-

Figure 4: (a) Loose Symbolic layout of a simple flip-flop. (b) Corresponding uncompacted mask-level layout. (c) and (d): Automatic pitch matching (before and after).
bolic layout style is sufficiently close to the manual style that the layout densities are comparable to the manual ones.

- **Compiled Cells**: Instead of leaving the relative placement to the designer, one can also attempt to perform this task automatically. A typical example of such a compiler is Topologizer [Koll85], which uses a rule based system to derive an optimal transistor positioning and inter-transistor routing. Until now, these techniques are still in a premature state and have not really been used in an industrial context.

- **Gate Matrix**: In almost every circuit, somewhere a piece of messy glue logic will be required. This is for instance the case at the edge of a bit-sliced data path, where some random logic is needed to gate and buffer the clocks and the control wires. In this case, the density of the layout is not of an extreme importance. More important is to align the logic to the data path so that a minimal space is required for the interconnect. The gate matrix [Lop80] technique is ideally suited for this task. It assumes a fixed layout strategy for the cells as shown in Figure 6. Transistors are aligned in rows along vertical poly wires, while the horizontal connections are made in metal. Software packages are available to minimize the number of transistor rows required. The gate matrix technique may not be that efficient (layouts are typically a factor of N larger than custom designs), but it allows for a fast generation of missing pieces of logic or for the design of cells which can be extremely well adapted to the environment in which they will be used.

Figure 5: L Program for Inverter.
3.2. Module Generators

A module can be defined as a basic architectural building block, designed to perform one particular function. It is constructed by assembling a set of leaf cells into a regular or irregular array. Typical examples of modules are RAM or ROM memories, PLA's, multipliers, data paths or controllers. In order to make modules generally applicable and reusable, they have to be made parameterizable. This means that the same module generator should be capable of generating an 8 x 8 as well as a 16 x 16 multiplier.

A typical module generator is a piece of procedural code, which describes the topology of the module in function of the leaf cells and the parameters. Depending upon the the internal structure however, we can consider three different type of module generators: standard cells, tiled modules and data path compilers.

- **Standard Cells**: The standard cell technique can be considered as one of the first types of silicon compilation [Mat74]. It is used to produce a random logic function of arbitrary complexity. In the standard cell technique, the layout area is divided in rows of library cells, separated by routing channels. The library contains a variety of cells, each implementing a standard logic function, such as an NAND or NOR gate or a flip-flop. All the cells must have the same height, but can have different widths. An example of a standard cell layout, implementing a digital filter, is shown in Figure 7. The input to a standard cell compiler consists of an enumeration of the library cells needed and their interconnectivity (or netlists). The task of the compiler is to partition the cells into rows and order the cells within a row (placement) and to route the interconnections in the routing channels. The overall objective is to minimize the total area and the length of the routing wires. The standard cell technique is the most popular layout generation technique at present and has been used extensively in high performance devices such as micro processors or signal processors. (Note: I consider the standard cell technique as a module generator, since it produces a system function by assembling basic library leaf cells).
**Tiled Modules**: A tiled module is constructed by tiling library cells in such a way that the internal connections are made by abutment. This technique is used to construct modules with a regular construction pattern. Typical tiled modules are memories, PLA's, multipliers, counters and adders. The first module generators to become popular were Weinberger Array [Wei67] and PLA-generators [Dem83]. Both of those array types are used to implement random logic functions.

The topology of the module is described by a procedure, which describes the positioning of the leaf cells in function of the parameters. The generator procedures can be either entered in textual format (as is the case in TimLager [Rue86] or GDT [Bur88] with the L-language) or using graphical templates (SDA [Law88] and MGE [Six86]). The compiler will input the module procedure and a set of parameters and generate a full layout. The generator may also generate other information about the module such as simulation models and speed and power consumption estimations.

An example of a module generator procedure is shown in Figure 8. The composition of the module is described in the C-language, extended with a set a library routines such as AddRight() and Addup(). These routines are used to define the relative position of the cells and the terminals. For instance, the first procedure in Figure 8 describes how a register of words bits wide can be constructed by abutting words cells called reg in the horizontal direction. The full power of the C-language can be used to describe more complex constructions. The module generator described in Figure 8 is called TimLager and is part of the LAGER system [Rue86].

**Data Paths**: A data path is a special module class, which exploits the word oriented nature of operators which act on data. An example of a data path is shown in Figure 9. It implements a simple adder/accumulator with reset (Figure 9a). Such operators can in general be organized in a bit sliced fashion and well defined layout strategies can be applied. An example of such a layout strategy is demonstrated in Figure
PARAMETERIZED MODULE GENERATOR

- STORAGE MODULES: RAM, ROM, REGISTERS
- ARRAY LOGIC: PLA, COUNTER

PARAMETERS
bits=4, words=2

PROCEDURE

for (i=1; i<=words; i++)
Addright("reg");

for (i=1; i<=bits; i++)
if (i EVEN)
Addup("reg_cell",LEFT|RIGHT,MY);
else
Addup("reg_cell",LEFT|RIGHT);

Figure 8: Procedural Generator for Tiled Modules.

Figure 9: Data Path Compiler: (a) Data Path Structure.
(b) after placement and feed-through assignment. (c) layout.
9b. Here it was selected to have the data flow in the horizontal direction, while control and clock signals are routed vertically. Internal connections between non-neighboring cells can be implemented by introducing feed-throughs in the intermediate cells. Data path compilers offer automatic placement (so that the number of feed-throughs and the overall wire length is minimized) and routing. The compilation process starts from a list of operators and the interconnections. Figure 9c shows the automatically generated layout for the data path defined in Figure 9a. Typical examples of data path compilers can be found in GENESIL [Che88] and LAGER [Rue86].

3.3. Floor Planners / Placement and Routing

The last steps in the chip design process consists in the interconnection of the generated blocks and the routing of the external connections to the bonding pads. The tools used at this level are called floor planners. The main tasks of a floor planner are the placement of the blocks, the definition of the routing channels and the routing of the interconnections. A large variety of floor planning tools are currently available, using automatic as well as user interactive approaches. Figure 10 shows an example of a simple processor, consisting of a data path, a PLA based controller and some random logic, implemented with standard cells. All blocks have been generated automatically using a suite of module generators, while the floor planning was performed by Flint, an interactive placement and routing tool [Rab85].

3.4. Design and Database Management

Silicon compilation environments tends to be fairly complex, as you might have noticed from the above descriptions. The amount of data to be stored (cell libraries,
simulation models, intermediate physical and symbolic views) tends to grow excessively with the size of the device under design. Furthermore, the number of software tools (compilers, placement and routing, interface programs, graphical editors) to be mastered by the designer becomes unmanageable. A streamlined and integrated design and data management system is therefore of prime importance for successful silicon compilation. The data management environment is responsible for the storing, the updating, the protection and the display of the design data, while the design management controls the flow of the design process, fires up the required compilation tools and generates the simulation models when requested. The data- and design management of the LAGER system is displayed in Figure 11. Central to the system is the object oriented OCT database with its graphical display tool VEM [Har86]. ALL compiler tools interface directly with the data base. OCT currently interfaces to more than twenty compilers for a wide variety of layout styles. The design manager (called DMoct) reads the hierarchical sdl-files, supplied by the user to describe his design, and sets up the database appropriately. The appropriate compilers are fired in the correct order, starting from the lower hierarchy levels and moving gradually upwards until the total chip is generated. In this way, the design only interfaces with the system through the sdl-files and the design manager. Using this type of approach, it is only a matter of weeks for an inexperienced designer to produce his first designs.

Figure 11: The Design Manager acts as an interface between the designer and the database and assembly tools.

4. Behavioral Synthesis

The behavioral synthesis task takes as an input the specification of the behavior, expected to be performed by the system under implementation, and a set of timing and area constraints. It generates a (set of) structure(s), which implements the required behavior and satisfies the specified design constraints.

What exactly is understood by behavior is a subject of a lot of controversy and depends heavily upon the target application area of the synthesis tools. In general, we
can state that a behavioral description describes what the system has to do, not how it actually performs these operations. The function of the system is described by a set of input/output relations in function of time. For example, a behavioral description of a micro-processor could consist of a definition of the instruction set the microprocessor has to execute. A digital filter specification for telecommunication applications consists either of the boundaries of the frequency domain response of the filter or a definition of the filtering function in the discrete z-domain. A controller function can be defined by a state diagram, while a logic function can be defined as a set of boolean functions. Especially the last class of synthesis applications, which is rather located at the block than at the system level, has received wide attention in the research environment and is called logic synthesis. In this chapter, we will first treat the languages and assorted synthesis tools for logic synthesis. This will be followed by a treatment of the so called architectural synthesis tools, which are oriented towards synthesis at the system level.

4.1. Logic Synthesis

The logic synthesis process translates a boolean description of a combinatorial function into a network of logic gates in such a way that either the area or the delay of the network is minimized (or a combination of both).

The research on techniques for logic synthesis has started almost simultaneously with the introduction of digital design (Karnaugh maps, Quine-McCluskey). Originally, the focus was mostly oriented towards minimization techniques for multi-level logic circuits (circuits, which consist of multiple levels of logic gates). With the advent of integrated circuits in the late sixties and the introduction of the PLA (Programmable Logic Array) as the most regular structure to implement random logic functions, the attention shifted to the minimization of two-level logic functions. An example of such a function (in the and-or representation) is given in (1):

\[ f = a \cdot b + c \cdot d \cdot e \quad (1) \]

where \( \cdot \) and + stand for and and or respectively. Research at IBM and Berkeley especially resulted in a set of powerful tools (such as MINI [Hon74] and ESPRESSO [Bra84]). These tools accept either a boolean or truth table description of the function and return a minimized truth table, which can then be mapped directly onto a PLA.

In the early eighties, a number of researchers showed that a substantial gain can be achieved by implementing logic functions in a multi-level fashion. Multi-level implementations allow for a much wider trade-off between area and time than their two-level alternatives. These multi-level functions can be efficiently compiled into hardware using the standard cell or gate-matrix techniques described higher. Examples of existing systems are the MIS system of Berkeley [New86] and the Yorktown Silicon Compiler [Bra85] from IBM. Both systems use algebraic or boolean manipulation techniques (combined with a set of heuristics) to minimize the logical function. The MIS system uses a language, called BDSYN [Rud86] as the input description to the logic synthesis process. An example of a piece of BDSYN code, describing the controller of a simple processor is given in Figure 12. The MIS system will first translate this description into a set of logic equations, before applying a script of subsequent minimization operations.
Another approach is to use rule based techniques and local transformations in the minimization process. Examples of such transformations are the removal of two inverters, which are connected in series, or the replacement of one three-input The SOCRATES system [Gre86] is an example of a logic synthesizer, based on this philosophy.

MODEL finite_state_machine

nextState, SETINVZ, SETZ, LOAD2, LOAD1
= presentState, INST<1:0>;

ROUTINE controller;

nextState = 0;
SETINVZ = 1;
SELECT presentState FROM
[0] : BEGIN
    SELECT INST FROM
    [0] : RESET = 1;
    [1] : BEGIN
        LOAD = 1;
        ADD = 1;
        END;
        LOAD = 1;
        SUBTRACT = 1;
        END;
    [3] : WRITE = 1;
    [4] : nextState = 1;
ENDSELECT;
END;
[1] : BEGIN
    LOAD = 1;
    END;
ENDSELECT;

ENDROUTINE;
ENDMODEL;

Figure 12: BDSYN description of a simple controller with two states.
The instruction word INST determines the action to be taken.

Figure 13: Examples of local logical transformations.
The above described systems are mainly targeted towards pure combinatorial circuits. Unfortunately, most real system always contain some memory elements and are thus sequential. The synthesis of sequential systems proves to be even more complicated. Approaches and techniques have been presented to handle state assignment for two-level logic circuits (for example the KISS system [Dmi87]). Active research is going on to develop the same type of capabilities for multi-level logic circuits.

4.2. Architectural Synthesis

Any system can be considered as a huge sequential circuit, consisting of a memory space and a chunk of combinatorial logic. Therefore, it should be possible to use logic synthesis techniques to synthesize complete systems. This approach suffers from severe setbacks however: modern day designs can approach a complexity of almost 100,000 gates. Using the logic synthesis techniques described above, it would take days or weeks to get circuits of this complexity minimized. A partitioning of the system in sub-systems or blocks is therefore essential. Furthermore, a system normally consists of a variety of blocks, some of them being random logic (such as controllers), other being composed of highly regular arrays of cells (such as a parallel multiplier), for which dense and efficient layouts are available in a library. These two topics describe fairly accurately the tasks of an architectural compiler: starting from a behavioral description, partition the system into a set a modules (sub-systems), which are available in the library or for which another compiler is available, and determine the necessary parameters for each block in such a way that the final architecture executes the required functionality within the user defined constraints of time and area.

4.2.1. Behavioral Description Languages

A variety of behavioral specification languages are currently available. The type of language selected depends heavily upon the type of application under consideration: A first possibility is to use conventional programming languages such as C, Pascal or Ada. Most of those languages are procedural and are optimized for Von-Neumann type processor architectures. They describe the algorithm as a sequence of operations to be performed or, in other words, they prescribe a a rigid control flow. This makes it harder for a synthesis tool to capture the inherent parallelism of an algorithm or to increase the throughput by introducing more concurrency. An example of a special purpose procedural behavioral language is ISPS [Bar81]. ISPS has been developed to describe the functionality and the operation of micro-processors. An example of ISPS code (from [Gaj88]) is given in Figure 14. It describes a conditional subroutine call instruction for a microprocessor.

Currently, the tendency exists to move towards so called data flow or signal flow oriented languages. Examples of such languages are applicative languages such as LISP and Ella [Mor85]. The advantages of applicative languages are the ease of manipulation and the lack of constraints on the use of concurrency and parallelism. Applicative descriptions can be translated directly into a Signal Flow Graph (SFG), which is the preferred data representation for most of the synthesis operations which are described below. Finally, in fields as digital signal processing, the signal flow graph is the preferred medium of system designers to describe their algorithms. As an example, Figure 15a shows the block diagram (SFG) of a first order digital filter. Figure 15b shows a description of the same filter using the SILAGE language [Hil85]. SILAGE is
Cond.call(c.bit<>):=
Begin
  DECODE c.bit
  Begin
    0 := pc = pc+1
    1 := Begin
      Dbuf = M[pc] next
      M[sp] = pc + 1 next
      sp = sp - 1 next
      pc = Dbuf
      end
  end
end

Figure 14: ISPS description for conditional subroutine call (from [Gaj88])

\[
\begin{array}{c}
\text{In} \\
+ \\
\text{State} \\
D \\
\text{Out} \\
+ \\
\text{State@1} \\
a \\
b
\end{array}
\]

(a)

func filter (In : Word16) := Out : Word16
begin
  Out = State + b * State@1;
  State = In + a * State@1;
end;

(b)

Figure 15: Block Diagram and SILAGE description of first order filter.

an applicative language, developed especially for the field of Digital Signal Processing. As can be seen in Figure 15, the SILAGE description of the filter is a direct one-to-one mapping of the graphical SFG into a textual format.

4.2.2. The Architecture Selection

The task of the architectural synthesis process is to map the algorithmic description into an architecture. In general, an architecture is composed out of a set of concurrent operating processors, where each processor consists of a controller, a set of data paths, memory and input/output or interface logic. (I define a processor here as a unit, which is characterized by a single control flow). Within this definition however, we can define millions of possible architectures, each of them optimized for one partic-
ular application area: data paths can use bit-serial or bit-parallel arithmetic, memory can be distributed or centralized, controllers can be single finite state machines or complex microcoded processors, input-output operations can be synchronous or asynchronous and the interconnection between the building block can proceed over a single or over a set of multiple busses. It turns out that the optimization and minimization procedures, used in the synthesis process, depend heavily upon the selected architectural template. Therefore, in the past we have seen the emergence of the so called vertical synthesis environments, each of them targeted at one particular application area.

Some examples of such compilers are: The FIRST [Den82] and CATHEDRAL-I [Jai86] for bit-serial signal processing, the LAGER [Rab85] and CATHEDRAL-II [Dem86] systems for multi-processor bit-parallel digital signal processors, the ADAM system [Kna84] for high-performance pipelined designs, and the CMUDA [Hit83] and DAA [Kow85] systems for micro-processor design.

At this present time, the selection of the target architecture and the global composition of that architecture is a task left to the designer. The designers experience seems to be an important factor in this process. This will make automation non-trivial. However, it seems feasible to develop a set of estimation tools, which analyze the complexity, the throughput, the concurrency and the memory requirements of an algorithm and which can suggest a possible architectural selection.

4.2.3. Synthesis Steps

Once an architecture has been selected, the synthesis process consists mainly of a set of transformation, translation and optimization steps (identical in some way to the phases of a classical software compiler). As mentioned higher, the order and the type of the different steps heavily depends upon the selected target architecture and the input language used. Discussing all possible operations would lead us too far. Therefore, we will restrict us to one particular set of operations, being the one used in the CATHEDRAL-II [Dem86] [Rab88].

CATHEDRAL-II is targeted towards medium speed signal processing algorithms (speech, audio, telecommunications), which are described using the SILAGE [Hil85] language. It maps those algorithms onto a bit-parallel processor architecture, in such a way that the processor structure is optimized to execute that particular algorithm. In order to reduce the search space and to guarantee efficient solutions, the possible processor architectures have been restricted as shown in one example in Figure 16. The data path of the processor consists of an interconnection of functional units, selected from a restricted library (containing an ALU, a multiplier, a comparator, a divider, a normalizer and an address arithmetic unit). These units are interconnected by a set of busses, selected in such a way that the data throughput is optimized. Every functional unit contains a (variable size) local register file. A multi-way branch microcoded controller architecture has been selected (Figure 16b). For example, the data path shown in Figure 16a is optimized to perform the following operations:

\[
\text{Max}[0] = 0; \\
\text{for ( } i = 1 \text{ to } N) \text{ begin} \\
\quad \text{Amplitude}[i] = \text{Real}[i] \times \text{Real}[i] + \text{Imag}[i] \times \text{Imag}[i]; \\
\quad \text{Max}[i] = (\text{Amplitude}[i] > \text{Max}[i-1]) ? \text{Amplitude}[i] : \text{Max}[i-1]; \\
\text{end}
\]
Figure 16: CATHEDRAL-II target architecture:
(a) Dedicated data path constructed from restricted set of operators.
(b) Multi-branch controller architecture.

The CATHEDRAL-II synthesis steps will now be discussed briefly. Similar types of operations can also be found in other synthesis systems (however in a different order and with a different flavor, depending upon the targeted architecture). Certain architectures or applications may require extra optimization steps, which are not mentioned in this list (e.g. filter synthesis and delay minimization in CATHEDRAL-I [Jai86]).

- **Partitioning**: A first synthesis step partitions the algorithm into sub-systems, which have to be implemented as separate processors (processes). Different processes are characterized by separate control flows (which means different controllers from the architectural point of view). The partitioning process is a very delicate operation and depends upon a large number of factors: the load balancing, the minimization of the data flow between processors, the minimization of the buffering and storage requirements and the identification of algorithmic subtasks with diverging computational requirements. An automatic partitioning is not straightforward. Therefore, this operation is performed manually in the present version of the CATHEDRAL-II. The designer can enforce a partitioning by adding so-called *pragma’s* (architectural hints to the compiler) to the SILAGE code. Research is currently going on in various places to at least partly automate this task (using high level complexity and concurrency estimations).
• **Transformation and Translation**: The first step after the partitioning consists in the compilation of the input description (for each sub-system) into an internal format. The most common internal representation is the data flow graph (DFG), sometimes extended with a set of control and timing constraints. For example, the SILAGE code of Figure 15 is translated into the DFG, shown in Figure 17. The data flow graph shows the ordering (or precedence) between the different operations, imposed by the data relations expressed in the behavioral description. For instance, the variable State has to be computed before we can start the computation of Out. The translation process from input description into internal data flow graph consists of an architecture independent and an architecture dependent part. After the parsing of the input description, a number of classical compiler optimizations and transformations can be performed, such as removal of manifest expressions, common sub-expressions and dead code. All these operations are generally valid and do not depend upon the selected target architecture and the available functional units.

![Data Flow Graph](image)

**Figure 17**: Data Flow Graph for the SILAGE code of Figure 15.

Other operations are however dependent upon the selected architecture: a multiplication has to be expanded into a set of adds and shifts when no parallel multiplier is available in the architecture. However, when a multiplier is available (and when the cost constraints allow for the introduction of a multiplier), the multiplier can be considered as a primitive operation and no expansion is needed. Subroutines and functions have to be in-line expanded when the selected controller architecture does not support subroutines. The same types of controller based decisions have also to be made regarding loops and conditional operations. Since these operations are architecture and library dependent, CATHEDRAL-II uses a rule based translator mechanism, which allows for a simple expansion of the basic architecture and the function library. Once again, user **pragma**'s can be used to overrule the decisions of the translator. Architecture specific local optimizations are also possible in this phase, such as the implementation and the minimization of delay elements.

The result of the transformation phase is a so called **primitive flow graph**, since all the operations in the graph are primitive and can be implemented directly in the available hardware.

• **Scheduling, Allocation and Assignment**: The major task of the synthesis process is the definition of the structure of the hardware. In general, a processor consists of an **interconnection** of **data operators** and **memory**, combined with a supervising **controller**. Therefore, following subtasks can be defined:
- the *hardware allocation*, which defines the number of execution units, memory units and busses will be needed to perform the algorithm within the time and area constraints.

- the *hardware assignment*, which binds operations to specific hardware units (such as ALU's, registers or busses).

- the *scheduling*, which determines the exact time slot in which each operation will be performed and which indirectly determines the composition and contents of the controller.

These three subtasks are intimately interwoven and clearly effect each other. The complexity of algorithms considering hardware units, memory usage and interconnection cost at the same time however is staggering. Therefore, all of the synthesis techniques published until now they are minimized separately (which tends to result in solutions which are far less than optimal with respect to at least on of those elements). This is true also in the CATHEDRAL-II system, which adopted following synthesis strategy:

1) Schedule the operations, performing the assignment of execution units at the same time within the defined hardware constraints.

2) Given the defined schedule, minimize the required number of register fields (by merging variable with non-overlapping lifetimes into the same register position).

3) Minimize the number of communication paths. A rescheduling might be necessary, when this causes conflicts.

Scheduling and allocation has been treated extensively in the literature (for example [Tse86], [Par86], [Pau87], [Goo87]). Instead of going into detail on all those approaches, I will use a simple example to demonstrate the issues involved. Consider the data flow graph of Figure 17. A first possibility would be to schedule each operation as soon as possible (ASAP), which means that an operation is performed as fast as its inputs are available. This results in the solution of Figure 18a, which requires 2 multipliers and 1 adder and performs the total algorithm in 3 clock cycles. It is easy to see that a better solution can be obtained as shown in Figure 18b. This solution uses the fact that the result of the second multiplication is not needed immediately and can be delayed till cycle 2. This reduces the hardware requirements to 1 multiplier and 1 adder (and probably requires also less registers). This type of scheduling makes use of the degrees of freedom on each operation to reduce the hardware requirements. A last

![Figure 18: Possible schedules for the DFG of Figure 17.](image-url)
possibility is shown in Figure 18c, where all the operations are scheduled sequentially and one single general purpose unit (as an ALU) is used to perform both the addition and multiplication. This solution is the cheapest in terms of hardware but is also the slowest (4 cycles). From the above, it can be seen that the search space for even a modest design grows pretty large. In fact, scheduling and allocation is an NP-hard problem. Most of the published techniques use specific heuristics to direct the search.

- **Controller Generation**: The result of the scheduling operation is a definition of the controller in the form of a finite state machine (as shown in Figure 19) for our simple filter. A last step in the synthesis process consists in the definition of the controller architecture and the instantiation of the parameters of the controller components, given this control flow graph. This state machine can be implemented in various ways: using random logic (and standard cells), as a PLA or using a microcode ROM with program counter. The optimality of an architecture depends upon the nature of the algorithm and the control state machine. For instance, when the processor contains large sections of sequential states (without branches), the program counter + ROM architecture may prove to be very efficient. A piece of code with a large number of branches is better off with a simple PLA implementation.

Most of the present synthesis systems (such as CATHEDRAL [Rab88] and LAGER [Azi88]) select one particular controller architecture and try to optimize the mapping of the control flow graph into that architecture. This approach can be based on an ad hoc partitioning of the control flow graph, followed by the generation (and minimization) of boolean expressions, truth tables or ROM tables. Not much results have been booked in the field of controller architecture selection given a certain control flow graph.

![Control Flow Graph corresponding to the schedule of Figure 18c.](image)

5. Example

An example of a complete synthesis process is given in Figures 20 and 21. Figure 20 shows the block diagram of a fifth order PCM filter for telecommunication applications and its corresponding SILAGE code. Using the CATHEDRAL-II tools, a variety of solutions can be generated. Figure 21a shows the default solution, containing only a single ALU. The execution of the filter on this processor would take 36 cycles. In an optimization step, the numbers of busses can be reduced from 4 to 1 (as shown in Figure 21b), resulting in an increase of the number of processor cycles to 39. The generated layout for this case is pictured in Figure 21c (from left to right: ALU with register file, controller circuitry, RAM). Adding pragma’s to the input code, the user can change the composition of the data path by adding either a multiplier (Figure 21d) or two more ALU’s. This reduces the number of processor cycles to respectively 18 or 20.
Figure 20: SILAGE description of fifth-order PCM filter

a. Signal Flow Graph
b. SILAGE Code
Figure 21: Processor synthesis example using CATHEDRAL-II
a, b, d and e: possible data paths, including the number of cycles
to execute the PCM filter.
c: automatically generated layout for the processor structure of b.

6. What to expect?
As can be deduced from the above elaborations, it is clear that structural synthesis
is in a far more mature state than behavioral synthesis. Most CAD houses are offering
at least a limited selection of structural compilers. The most important breakthrough,
being the acceptance and application of these tools by system designers (instead of IC
designers), has still to happen though.
Behavioral synthesis on the other hand has still a long way to go. Limited successes have been booked in restricted areas such as logic synthesis and synthesis for certain digital signal processing applications (where synthesis is actually making it to the design community!). The large research efforts which are currently being invested in the area and the results being booked recently however promise some substantial advances in the near future. The most important result of the present investigations is a more thorough and clear understanding of the different representations, phases and operations in the system design process, just the same as what happened to physical design in the last decade.

I would like to express my conviction though that system synthesis will remain a user interactive operation, where the software acts as a Design Aid to the designer: given the constraints defined by the designer, the synthesis system fills in the details (such as an exact timing of the operations, assignment of variables to registers and generation of the connection network). Major design directives can still be given by the user.

Acknowledgements

This paper is based on the work of a large number of people. In particular, I would like to mention H. De Man, R. Brodersen, R, Jain, J, Van Meerbergen, S. Chung, F, Catthoor, J, Vanhoof, G.Goossens, P. Hilfinger, S. Pope, B. Richards and C. Chu, whom I had the pleasure to collaborate with in the past or at present.

* * *

References


1. INTRODUCTION

These lectures will discuss the high-energy physics programme at CERN. The emphasis will be on the future, in particular on the programme for LEP, the large electron-positron collider due to start operation in 1989. A short introduction to high-energy physics and some general background information on CERN will be given first.

Inevitably, much of the material will be reasonably familiar to those who work on experiments at CERN, or who, in some other way, are associated either with the laboratory or with high-energy physics. However, almost half of the participants at this year’s School of Computing do not have any clear link with CERN or with high-energy physics, and it is for them that these lectures are primarily intended.

2. HIGH-ENERGY PHYSICS

High-energy particle physics is the study of the basic constituents of matter and of the forces which act between them and hence control them. It is one of the areas of basic science operating at the frontier of our knowledge of the physical world. It addresses questions of a fundamental nature, directed towards the understanding of our Universe and its origins.

2.1 Constituents

The present studies of the basic constituents are the natural extensions of the work that led to the discovery of the atom, the electron, the atomic nucleus, and the neutron. There is now compelling evidence that these constituents, the so-called elementary particles, comprise six leptons and six quarks plus their antiparticles (see Fig. 1). Leptons, of which the neutrino
and the electron are examples, can be either electrically neutral or charged. Quarks are electrically charged and are the basic building blocks of hadrons (i.e. protons, neutrons, pions, kaons, and Ω⁻, to name a few of those that are reasonably well known, but there are hundreds more).

At the present limits of measurement (~ 10⁻¹⁸ m or ~ 10⁻³ of the size of the proton), both quarks and leptons are consistent with being point-like, which is in line with the notion that they are fundamental.

2.2 Forces

The forces—or interactions—we know are the gravitational, the weak, the electromagnetic, and the strong forces (Fig. 2). Of these, the electromagnetic interaction, which is responsible for the force between electric charges, is, by far, the best understood. It is fully described by the highly successful relativistic quantum field theory known as quantum electrodynamics (QED), which has been extensively tested and is in agreement with all experimental data.

Within the framework of QED the forces between electric charges are transmitted by photons. The source of the electromagnetic field is the electric charge, and the quantum of the field, the mediator, is the photon. Such a field in which the source and the quantum are different entities is called Abelian.

The success of QED has led us to use it as a model on which to base our theories of the other interactions, according to which interactions always occur via the exchange of certain particles. There is strong experimental evidence that these exchange particles or mediators of the interactions are bosons, i.e. particles with integral spin in units of ħ. To date, only gravity, which is so well understood classically, does not fit comfortably into this picture. There is no experimental evidence for the graviton, postulated to be the mediator of the gravitational force, nor have we as yet been able to formulate an adequate theory of quantum gravity. Fortunately, for most practical purposes, gravity can be ignored when considering the interaction between particles, and therefore it rarely creates problems for research in high-energy physics.

The quantum theory of the strong interaction is known as quantum chromodynamics (QCD) by analogy with QED. It describes the force between quarks for which the mediator is

---

![Diagram](image)

**Fig. 2** The four known forces in nature

261
the gluon. Like the photon of QED, the gluon is massless. The first experimental evidence for gluons was obtained in 1979 at the $e^+e^-$ collider PETRA, at DESY in Hamburg. According to QCD, the gluon is both the source of the field and its mediator. Such a field is non-Abelian, and in this respect it is fundamentally different from QED. All experimental tests of QCD so far devised have agreed with its predictions.

The mediators of the weak interaction are the weak intermediate vector bosons known as $W^+$, $W^-$, and the electrically neutral $Z$. They were discovered at CERN in 1983, and this led to the award of the Nobel Prize for physics to C. Rubbia and S. van der Meer in 1984. The mass of the $Z$ was found to be $\sim 93$ GeV/c$^2$, and that of the $W$, $\sim 81$ GeV/c$^2$.

The electromagnetic and the weak interactions had, over the years, been shown to have many similarities. These similarities had led Glashow, Salam and Weinberg to formulate the idea that the two interactions were simply different manifestations of just a single underlying interaction and so could be merged into a unified electroweak theory in which the photon, the $W$, and the $Z$ are treated on an equal footing. On the basis of this idea they predicted the existence of the $W$ and the $Z$ and their masses long before they were discovered experimentally.

2.3 The Standard Model

The two theoretical models—QCD for the strong interactions, and the electroweak theory for the electromagnetic and weak interactions—work well with, as yet, no inconsistencies with experimental data. Together they make up what is known as the Standard Model of elementary particle interactions.

Built into the Standard Model is the idea of continuous internal symmetry. This is an extension of an old idea proposed by Heisenberg in the 1930s to explain the close similarity between proton and neutron interactions. Furthermore, the symmetry should, as in QED, be preserved under local gauge transformations. Thus the Standard Model is a gauge theory and has the important feature that the symmetries actually determine the interactions.

However, for the electromagnetic and the weak interactions to emerge as features of a single theory producing, at the same time, the massive mediators of the weak interaction plus the massless photons, the symmetry has, in the end, to be broken and so also must gauge invariance. Fortunately, all this can be done in a mathematically consistent way, yielding the remarkable result that the mediators of the electroweak interaction are the two charged vector bosons $W^+$ and $W^-$, and the two neutral vector bosons, $Z$ and the photon, as observed experimentally.

There is nevertheless a penalty, which is that a scalar field, known as the Higgs field, must be present so that the theory preserves, as it must, exact gauge invariance for QED. A consequence of this is the existence of a neutral spinless particle, labelled H for Higgs boson, which couples to particle masses. So far, the H has not been observed experimentally, perhaps because the particles in beams normally available from accelerators are relatively light — of the order of the mass of the proton or less — and so have only very weak couplings with the Higgs field. Until the H has been observed we cannot claim to understand from where the $W$ and $Z$ — or for that matter any other particles — get their masses.

The existence of the Higgs scalar particle H is a crucial prediction of the Standard Electroweak Theory, although the theory does not give the mass of the H. Other important predictions of the theory include: the existence of the Z boson — the mediator of weak neutral currents — in addition to the $W^\pm$ bosons, which mediate the weak charged currents; the presence of interference effects in processes where Z and photon (γ) exchange are possible; and the masses of the $W$ and $Z$, given by
\[ m_w = \left(\frac{\pi \alpha}{\sqrt{2} G_F}\right)^{1/2}/\sin \theta_w \]
and
\[ m_z = m_w/\cos \theta_w, \]

where \(\alpha\) is the fine-structure constant, \(G_F\) is the Fermi weak coupling constant, and \(\theta_w\) is the angle known as the electroweak mixing angle. Typical examples of the sort of reactions which should occur according to the theory are illustrated by the Feynman diagrams shown in Fig. 3. Clearly, interference effects in reactions such as those given by diagrams (a) and (b) should exist.

Many experiments have by now been carried out to test the electroweak theory. These include the observation of weak neutral currents at CERN in 1973, in an experiment done in the heavy-liquid bubble chamber, Gargamelle, to study \(pN\) scattering; subsequently, there have been measurements of \(\nu\) and \(\bar{\nu}\) scattering off nuclei and off electrons. The existence of \(\gamma-Z\) interference has been demonstrated in the measurement of asymmetries in \(e-D\) scattering with polarized electron beams and by the observation of parity violation in transitions between certain atomic levels. Most importantly, there is, of course, the discovery of the \(W\) and \(Z\) bosons and the measurement of their masses in the UA1 and UA2 experiments at CERN’s Proton-Antiproton Collider. The results of these experiments, and of many others not mentioned here, can be used to give values of the electroweak mixing parameter \(\sin^2 \theta_w\). Significantly, all determinations are consistent with one another. Bearing in mind that the measurements span a range of about \(10^9\) in energy, this is a remarkable result in support of the theory.

Turning finally and briefly to QCD or the strong interaction part of the Standard Model, the basic interaction can be described in terms of the Feynman diagram (Fig. 4) whereby the interaction between two quarks is mediated by a massless vector gluon which couples to a hidden symmetry of the quarks known as ‘colour’ and which, in some sense, is analogous to the electric charge in QED. Gluons themselves carry colour and so can interact with each other, unlike photons. Colour, which in the theory comes in three different varieties, is not observed in hadrons, since they are all colour neutral. According to the theory, which so far is
consistent with experiment, quarks and gluons do not emerge as free particles from high-energy collisions, owing to colour confinement. This gives to this interaction the very interesting property of asymptotic freedom, whereby the coupling between two coloured objects increases as the distance increases, and vice versa. Thus in trying to separate, say, a quark from a gluon, the interaction energy increases, leading to the production of $q\bar{q}$ pairs, which multiply to produce jets of normal colour-neutral hadrons.

The Standard Model describing the electroweak and strong interactions is consistent with the results of all experiments published so far. Nevertheless, it is unsatisfactory in that it contains more than 20 free parameters. Also there are many fundamental questions that remain unanswered and many aspects of the theory that must be more rigorously tested experimentally. Arguably, the most important amongst these is whether the Higgs particle $H$ exists and, if so, what is its mass. Then there is the problem that the top-quark has not yet been discovered; does it indeed exist? and again, what is its mass? Is the number of quark and lepton types limited? And, if so, what is the limit? For example, is the number of neutrino types limited to three? Are quarks and leptons point-like or do they have a substructure? Can the electroweak theory and QCD be combined into a single theory, which could have consequences such as the protons having a finite lifetime and the possible existence of a new breed of so-called supersymmetric particles? Finally, can all the forces in nature be unified into one theory, which includes gravity? These are just some of the challenges which face theorists and experimentalists now working in the field.

3. CERN AND ITS FACILITIES

3.1 History

CERN was founded during the years 1953 and 1954, but its origins date back to the late 1940s. At that time many European scientists wished to re-establish Europe as a place of scientific pre-eminence, and to halt and hopefully reverse the ‘brain drain’ to the USA. They hoped also that by establishing a truly European laboratory it would help to rebuild the bridges in a Europe torn apart by the Second World War.

Originally there were 12 founding states, namely Belgium, Denmark, France, Germany (Federal Republic), Greece, Italy, the Netherlands, Norway, Sweden, Switzerland, the United Kingdom, and Yugoslavia. The CERN Convention was signed in Paris on 1 July 1953 and came into force on 29 September 1954. Thus CERN is older than the EEC. The following changes have taken place since then. Austria joined CERN in July 1961 and Yugoslavia withdrew at the end of the same year. Spain also joined in 1961, but withdrew in 1968 and then re-joined in November 1983. Most recently, in 1986, Portugal joined CERN, bringing the number of Member States up to 14.

CERN's governing body is the CERN Council. Each Member State is represented on the Council and has one vote. The President of Council is appointed from one of the Member States. Its Secretary-General is the Director-General of CERN. The Council is assisted in its work by a Finance Committee and a Scientific Policy Committee.

The funds for CERN are provided by the Member States according to scales that are based on the average net national income of each. There is, however, an upper limit of 25% on the contribution that any one state is required to pay. The scales are decided by Council every three years. The budget itself is fixed each year in Swiss francs. For 1988 it amounts to 792 million Swiss francs, comprising 395 MSF for materials costs and 397 MSF for personnel costs, including contributions to the Pension Fund.
CERN has about 3500 staff on its payroll and, in addition, supports about 500 Fellows, Paid Associates, and students, who typically have contracts ranging from a few months to about two years in the case of Fellows.

There are currently in the region of 4500 registered users of CERN (so-called Unpaid Associates) from universities and institutes located principally in the CERN Member States. In general, their salaries, travel costs, and subsistence allowances are paid by their home institutes. These users comprise physicists, engineers, computer specialists, technicians, etc., some of whom are able to work full time at CERN for periods of a few years. However, most of the users stay at CERN for only short periods. On the average, there are approximately 2000 users from outside CERN on the site at any given time.

3.2 The facilities

CERN's role is to provide, within Europe, the basic facilities, the infrastructure, and the environment for carrying out research in particle physics at the frontier of the subject. Its prime responsibility is to the European community of high-energy physicists from universities and research institutes in the Member States. Its function is to serve that community. Accordingly, it should endeavour to provide the best research facilities that its funds will allow, and an exciting scientific and technical environment for its users.

The basic tools required for HEP research are very high energy particles. Very short distance phenomena are being studied and this requires the use of probes of comparable wavelengths. Also, as mentioned earlier, the forces can be carried by very massive particles, which in turn need high energies to create them.

Many high-energy accelerators have been built to meet these needs. The highest energies are produced by colliders in which two beams of particles travelling in opposite directions are accelerated and brought into collision with one another. Electron-positron, proton-proton, proton-antiproton, and electron-proton colliders have been built or are under construction.

Accelerators are also used to provide beams of secondary particles (e.g. $\nu$, $\mu$, $\pi$, $K$, $n$, $\bar{p}$, $\Lambda$, $\Sigma^+$, $\Xi^-$, $\Omega^-$ have all been used in experiments) for a wide range of so-called 'fixed-target' experiments, which are complementary to those done at colliders.

At present there are four basic accelerators, which are (or will be) the source of all high-energy particles at CERN. These are listed below:

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Nominal energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Synchro-cyclotron (SC)</td>
<td>600 MeV protons,</td>
</tr>
<tr>
<td>The Proton Synchrotron (PS)</td>
<td>28 GeV protons,</td>
</tr>
<tr>
<td>The Super Proton Synchrotron (SPS)</td>
<td>450 GeV protons,</td>
</tr>
<tr>
<td>The Large Electron-Positron Collider (LEP)</td>
<td>50–100 GeV per beam</td>
</tr>
<tr>
<td>(under construction)</td>
<td></td>
</tr>
</tbody>
</table>

3.2.1 The Synchro-cyclotron (SC)

The SC, the first accelerator to be built at CERN, started operation in 1957. It accelerates protons up to an energy of 600 MeV and has also been used to accelerate $^3\text{He}^+$ and $^{12}\text{C}^{++}$ ions. It is now used principally for studying short-lived isotopes and other nuclear properties, for condensed-matter physics with low-energy muons, and for radiochemistry.

3.2.2 The Proton Synchrotron (PS)

Initially the PS, which came into operation in 1959, was built to provide 28 GeV protons for fixed-target experiments at CERN. Now it is the heart of a complex system of machines
that have been developed around it to provide beams of protons and antiprotons and of oxygen and sulfur ions, suitable for injection into the SPS. Antiprotons are also provided—but on this occasion after being decelerated in the PS—for the Low-Energy Antiproton Ring (LEAR) for studies with intense, exceptionally pure beams of low-energy antiprotons. The other important function of the PS is to act as a pre-injector for LEP; electrons and positrons will be accelerated to 3.5 GeV and sent via the SPS into LEP. Thus almost all of CERN’s programme is entirely dependent on the PS, and furthermore, although fixed-target experiments at the PS are no longer carried out, secondary beams are still heavily employed to test detectors for use at the higher-energy machines. The complete arrangement is illustrated schematically in Fig. 5.

No discussion of the PS can be complete without at least a brief mention of the Intersecting Storage Rings (ISR). Based on two storage rings which intersected in eight regions, the ISR was fed from the PS and enabled experiments with colliding beams of protons to be carried out in a range of energies up to 31 GeV per beam or 62 GeV in the centre-of-mass system. The ISR machine operated successfully and with increasingly sophisticated detector systems from 1971 until 1984 when it was closed down in order to release funds for the construction of LEP. By that time, it had operated not only for pp collisions but also for p̅p, dd, and αα collisions. Also, there can be no doubt that it enabled CERN to develop the expertise, experience, and confidence needed later to build the SPS p̅p Collider and LEP.

3.2.3 The Super Proton Synchrotron (SPS)

The SPS first came into operation in 1976. It is built underground, in a tunnel of 7 km in circumference. Initially designed to accelerate protons to 300 GeV, its performance has been steadily improved to the point where it now regularly operates at 450 GeV. The protons are extracted to produce a rich variety of secondary beams (ν, ¯ν, μ±, π±, K0, K±, n, ¯p, hyperons,
γ, and e+) for studying many different particle interactions and decays. More recently, beams of oxygen and sulfur ions have been accelerated in the SPS; after extraction from the machine, they have been used in experiments to search for phase transitions in collisions with lead nuclei. The aim is to produce conditions of high nuclear density in which a quark-gluon plasma can be created and detected.

Undoubtedly the most significant development at the SPS has been the advent, in 1981, of the SPS Collider, which enables collisions between 315 GeV protons and antiprotons to be studied. (When run in a special pulsed mode, energies up to 450 GeV per beam have been reached.) A vitally important feature of the Collider complex, and one on which its success hinges, is the ability to collect, store, and ‘cool’ low-energy antiprotons produced initially by the PS.

This is done in a small antiproton accumulator ring (AA) by a method invented and developed by S. van der Meer in which antiprotons are first produced by bombardment of a tungsten/iridium target by a beam of 26 GeV/c protons from the PS. Antiprotons which are produced in a small momentum range about 3.5 GeV/c are cleverly focused and transferred to the AA where they circulate and accumulate for periods typically up to 24 hours. During this time they are subjected to a process known as stochastic cooling whereby properties such as the momentum spread and size of the circulating beam are gradually reduced. The method is to use special pick-up electrodes to sample the state of the circulating particles as they pass a particular point in the accumulator ring and then to transmit correcting signals across the diameter of the ring (see Fig. 6) so that appropriate correcting fields can be applied, half a turn later, to the same particles in order to improve the spread in the values of the beam properties sampled.

Fig. 6 The Antiproton Accumulator ring
By this means, with more recently the addition of a second large-aperture collector ring (ACOL), up to $8 \times 10^{11}$ antiprotons have been cooled and stored at 3.5 GeV/c for subsequent use in the Collider or in LEAR. For the Collider they are extracted from the AA, transferred to the PS for acceleration to 26 GeV/c, and then passed to the SPS where, with a counter-rotating beam of protons also obtained from the PS, they are accelerated up to 315 GeV/c for servicing the experiments.

3.2.4 The Low-Energy Antiproton Ring LEAR

The availability of large numbers of antiprotons at CERN has made feasible the provision of a low-energy antiproton ring, LEAR, for special experiments requiring exceptionally pure beams of low-energy antiprotons. A wide diversity of experiments are now being carried out including tests of CP violation, a comparison of the gravitational and inertial masses of the antiproton with those of the proton, and searches for evidence for the existence of gluonic matter, which in turn would provide direct evidence for gluon self-coupling—an essential ingredient of QCD.

4. LEP

The SC and the elaborate PS/SPS complex, including LEAR, constitute the platform on which today's experimental programme at CERN is built. As from 1989, CERN will be adding the Large Electron–Positron collider (LEP) to the range of accelerators which it makes available to its user community. About half the members of this community are planning to base their research on LEP in the coming years.

Located in a tunnel almost 27 km in circumference and at depths varying between 50 and 170 m below ground level, LEP straddles the Swiss-French border between the Jura mountains and Geneva airport (see Fig. 7). When it first comes into operation it will be

![Fig. 7 Map of the region in which LEP is located](image-url)
equipped with sufficient RF power to accelerate the electrons and positrons up to 55 GeV per beam, i.e. well above the energy required to produce the Z particle. Thereafter, it is planned to increase the energy to about 95 GeV, which is comfortably above the $W^+W^-$ pair-production threshold. Some of the more important design parameters at these two energies are given in Table 1.

| Table 1 |
|---|---|
| **LEP design parameters** |  |
| Beam energy (GeV) | 55 | 95 |
| Circumference (km) | 26.66 | 26.66 |
| Dipole field (T) | 0.0645 | 0.1114 |
| Injection energy (GeV) | 20 | 20 |
| RF frequency (MHz) | 352 | 352 |
| Dist. between supercond. quads (m) | ±3.5 | ±3.5 |
| r.m.s. bunch length (mm) | 17.2 | 13.9 |
| r.m.s. beam radii: $\sigma_x$ (µm) | 255 | 209 |
| $\sigma_y$ (µm) | 15.3 | 10.8 |
| Bunch spacing (µs) | 22 | 22 |
| Nominal luminosity (cm$^{-2}$ s$^{-1}$) | $1.6 \times 10^{31}$ | $2.7 \times 10^{31}$ |
| Beam lifetime (h) | 6 | 5 |
| r.m.s. energy spread | $0.92 \times 10^{-3}$ | $2.06 \times 10^{-3}$ |
| Current (4 bunches) (mA) | 3 | 3 |
| Synchr. rad. loss per turn (GeV) | 0.263 | 2.303 |

4.1 Injection into LEP

One of the important features of LEP is that it will use two of CERN’s existing accelerators, namely the PS and the SPS, operating in tandem, as part of the injector system. For this purpose the PS has been equipped with two electron linacs as shown in Fig. 8. The

![Fig. 8 The electron and positron injection system for LEP](image-url)
first of these is a high-current, 200 MeV machine, producing an output current of 2.5 A for electron-to-positron conversion in a tungsten target. From here, a positron current of 12 mA is produced in a form suitable for subsequent acceleration to 600 MeV in the second linac. The first linac is also used to provide electrons for LEP by detuning its electron gun in order to produce 110 mA at the linac output: these electrons are then accelerated to 600 MeV, again in the second linac.

The next stage in the injection chain is to transfer electrons or positrons, as appropriate, into an Electron–Positron Accumulator ring (EPA), where they will be stored and accumulated before injection into the PS. Acceleration to 3.5 GeV takes place in the PS, followed by transfer to the SPS for acceleration to 20 GeV and, finally, injection into LEP. Figure 9 shows a picture of the section where transfer to the EPA takes place.

The full scheme is illustrated in Fig. 10. Electrons and positrons are injected into LEP in separate, sequential cycles within a 15 s period or supercycle. This operation is matched to the normal fixed-target mode of operation of the SPS, and is carried out without affecting it. Within a supercycle (see Fig. 10), the positrons are produced and accumulated in the EPA for 10.8 s, then transferred to the PS in two equal slices separated by 1.2 s. The electrons are accumulated for 1.2 s in each of two cycles before transfer into the PS. In each 1.2 s interval, the positrons or electrons are accelerated to 20 GeV, as described earlier, and injected into LEP, where they are stacked in four equally spaced bunches for particles of each charge. The whole process is then repeated in successive supercycles until the required beam currents in LEP are reached and acceleration to full energy takes place. On the basis of the performance achieved so far, this is expected to take 12 minutes.
Construction of the injection system has now been completed and the individual components have been fully commissioned. In July 1988, a series of injection tests were carried out with positrons, enabling the most significant milestone in the LEP construction programme to be passed. On 12 July a beam of four positron bunches was successfully extracted from the SPS, steered along the complex transfer system, injected into LEP, and transported to the end of the first completed LEP octant. This was achieved during the first shift of a scheduled six-day run for the injection tests, all of which were done parasitically during normal running of the SPS for fixed-target physics. A picture of the beam spot as it appeared on a luminescent screen at the end of the octant is shown in Fig. 11. During the test period it was possible to study the beam dynamics under various conditions in the first LEP octant, to check the effective aperture, to investigate the response time of the computer control system and network, and to study the behaviour of much equipment with the beam on, e.g. radiation monitors, pick-up electrodes, interlocks, beam probes, etc. Also the part of the injection chain from the SPS onwards, namely the beam ejection system, the transfer line (see Fig. 12), and the system for injection into the LEP octant, could be tested as a whole for the first time.

All the results obtained were fully compatible with the design goals. In particular, the first LEP octant was shown to be satisfactory both electrically and magnetically, and also as regards alignment, vacuum, water cooling, and diagnostics. The beam optics down the transfer tunnel and into the first octant, involving a complicated arrangement of bending magnets and quadrupoles, were good, and the transfer efficiency from the SPS to the end of the octant was 100% within the accuracy of the measurements. Overall, the reproducibility of
all parts of the system was excellent, and beam intensities throughout the tests were typically $10^{16}$ per pulse, i.e. about 1.4 times the design intensity.

4.2 The LEP ring

As for the rest of LEP, it is worth recalling that another important milestone was passed on 8 February 1988, when the excavation of the LEP tunnel was completed. The last stage of the
tunnelling, which was under the Jura mountains, had been particularly difficult. It was known from the beginning that there were geological faults in this region, with the likelihood of striking underground water at high pressure. Despite elaborate precautions and careful planning, serious problems with flooding were encountered, which delayed completion of this part of the tunnel by over a year. At one stage, water under a pressure of 10 atm was flowing into the tunnel at a rate in excess of 150 l/s.

The present status of LEP is now as follows. The tunnel, the access shafts, and the underground experimental areas are all excavated, and the concreting is complete everywhere, except for 2.8 km of the tunnel under the Jura and two of the less important access shafts. Installation of the infrastructure (lighting, ventilation, monorail, elevators, electrical services, etc.) is well advanced. Almost all machine components are now on site and have been tested, including all the magnets (3400 dipoles, 760 quadrupoles, 512 sextupoles, and 630 correcting dipoles), the RF system (128 cavities), and the vacuum system.

As regards installation, as much pre-assembly as possible is being done on the surface. For example, the dipoles are assembled in pairs together with their 13 m long sections of vacuum pipe and excitation bars. They are then fixed to a special transportation girder and lowered as a unit into the tunnel for transfer by monorail to the place where they are to be installed in the tunnel. Much the same thing is done for the short straight-section elements, each comprising, typically, a quadrupole, sextupole, correcting dipole, vacuum pipe, and pick-up electrodes.

As of 10 August 1988, about half of the machine components had been installed. The plan is to have all components in seven out of the eight octants installed by the end of 1988, and all seven under vacuum by mid-March 1989. All of the RF system should be installed and fully tested by May 1989. As for the final octant under the Jura, the installation of the infrastructure and the machine components will follow closely behind the civil engineering. The hope is to have two-thirds of the octant installed by the end of this year, with completion by mid-April 1989.

It is planned to have the whole machine connected up and under vacuum by mid-June, ready for the start of commissioning in July 1989. Photographs of parts of the magnet and RF systems now installed in the tunnel are shown in Figs. 13 and 14.

Fig. 13  Magnets installed in the first LEP octant
5. **THE DETECTORS FOR LEP**

LEP will be provided with four large underground experimental areas to house the four experiments that have been approved. An artist’s impression of a LEP experimental hall and associated surface buildings is shown in Fig. 15. These areas are now equipped with cranes and general infrastructure, and have been made available to the experimental groups for the installation and assembly of their detectors.

Each of the four LEP detectors contains the usual ingredients that are essential for modern detector systems at $e^+e^-$ colliders, namely: a solenoidal magnet, a central tracking chamber, electromagnetic and hadron calorimeters, a muon detector, a luminosity monitor, and a high degree of hermeticity. Also, they all make use of multilevel trigger systems which aim to reduce the data-taking rates to 1–2 Hz, with at least the first-level triggers operating within the 22 $\mu$s gap between successive bunch crossings. Despite these obvious similarities, there are very significant differences in the techniques used from one detector to another, depending on the particular expertise that exists within the respective collaborations and on the aspects of the physics that they wish to emphasize. Taken together, the four detectors provide a powerful combination of systems with which to study the many physics questions which can be addressed at LEP.

Installation of the detectors in the underground experimental halls is progressing well. The magnets are assembled, many of the local electronics huts have already been lowered into the halls, and the first of the detector subsystems are being installed. Barring serious unforeseen problems, all four detectors should be assembled and in position in the ring by the time the commissioning of the full LEP machine starts.

A brief description of each of the detectors is given below, emphasizing a few of the many interesting features which characterize each of them.
5.1 ALEPH

A schematic drawing of the ALEPH detector is shown in Fig. 16. A large superconducting solenoid, 6.4 m long and 5.3 m in diameter, is used to provide a magnetic field of 1.5 T for the detector. The following subsystems are located within the solenoid. Moving outwards from the centre of the detector, there is first a microvertex detector adjacent to the beam pipe. This is followed by an inner tracking chamber, used primarily for triggering; then there is a large time-projection chamber (TPC), and finally an electromagnetic calorimeter. The solenoid is enclosed in an instrumented iron return yoke in which layers of streamer tubes are interleaved with 5 cm thick iron plates for use as a hadron calorimeter. This system is also used, in conjunction with two double layers of streamer tubes placed outside the yoke, for muon detection.

The TPC is 4.8 m long and 3.6 m in diameter. Wide-angle tracks are each sampled 300 times, enabling momenta to be measured to an accuracy $\Delta p/p = 1.3 \times 10^{-3}$ p (GeV/c) and $\text{d}E/\text{d}x$ to $\pm 4.5\%$. There are approximately 3000 wires and 20,000 pads on each end-plate of the TPC, which is built in 18 sections, with radially stepped boundaries. An unconventional pad structure built up in concentric circles helps to avoid deterioration in resolution for high-momentum tracks by ensuring that they make only small angles with respect to the pad axis.

Both the electromagnetic and the hadron calorimeter are built up with pad readout arranged in towers pointing towards the interaction vertex. The electromagnetic calorimeter is
Fig. 16 The ALEPH detector

designed for fine granularity in order to give good identification of electrons in jets. It has 45 layers of lead, 33 of which are 2 mm thick and 12 (the outer layers) 4 mm thick. Between each layer there is a plane of wire chambers filled with a gas mixture of xenon (80%) + carbon dioxide (20%). In all, there are 48,000 towers in the barrel section and 24,000 in the end-caps. Results from measurements in a test beam have shown that the spatial resolution for locating shower centres will be ±2–3 mm and the energy resolution ±18%/√E.

Most of the subsystems of the detector have been delivered to CERN. The magnet with its superconducting coil has been operated successfully at full current, and the field has been mapped. It has since been dismantled and reassembled, along with the hadron calorimeter built into the yoke, in the underground experimental hall. Figure 17 shows a picture of this part of the detector during reassembly. The various modules that make up the electromagnetic

Fig. 17 The ALEPH detector during assembly in the underground experimental hall
calorimeter (12 for the barrel section and 12 for each of the end-caps) will be installed next. The uniformity of response within each module and between different modules has been shown to be within ±2%. The other parts of the detector are well advanced. The inner tracking chamber is ready and tested; the resolution obtained in $r$-$\phi$ is ±100 $\mu$m and in $z$ is ±3.2 cm. The TPC is currently under test, using cosmic rays and a laser calibration system, and will be installed towards the end of the year. Pictures of the field cage for the TPC and of the chamber itself during preparation for inserting the inner cylinder are shown in Figs. 18 and 19.

Fig. 18 The field cage of the ALEPH TPC

Fig. 19 Assembly of the ALEPH TPC
5.2 DELPHI

The DELPHI detector is unique amongst the LEP detectors in placing particular emphasis on hadron identification over a wide momentum range. For this purpose it uses a complex system of ring-imaging Cherenkov (RICH) counters covering most of the solid angle, supplemented by dE/dx information obtained from a TPC. In many other respects, the detector is rather similar to that of ALEPH. As can be seen in Fig. 20, it has a microvertex detector, an inner detector for triggering, a TPC, and an electromagnetic calorimeter, all lying inside a large superconducting solenoid that provides a magnetic field of 1.2 T. The RICH counters and an outer detector are also located inside the solenoid; the TPC is therefore made smaller than in ALEPH in order to accommodate them. The iron return yoke around the solenoid (see Fig. 21) is instrumented with streamer tubes for hadron calorimetry; muon detection is done with two planes of drift chambers, one inside the return yoke and the other on the outside. DELPHI has the special feature that the barrel sections of both the RICH counter and the electromagnetic calorimeter all use the same principle as the TPC to measure three-dimensional space points along charged tracks in these detectors. In this way it is hoped to ease the problems of track reconstruction in high-multiplicity events. Another feature that is unique to DELPHI is the provision of two sets of chambers in the forward region to improve both tracking and momentum measurements.

The RICH detectors make use of liquid and gas radiators, cleverly arranged so that a single readout system can be used for both. When combined with dE/dx measurements in the TPC, they provide π, K, and p identification for momenta from 0.3 to about 25 GeV/c with low error rate and good efficiency. The principle of the scheme (Fig. 22) is as follows. The inner cells are...
filled with a liquid Freon and are 1 cm thick in the radial direction. Cherenkov light, produced in a well-defined cone when a particle of sufficient velocity passes through a cell, enters a system of drift tubes filled with gas containing a small amount of TMAE. The TMAE converts the Cherenkov photons into electrons by photoionization, within a distance of less than 3 cm from the inner walls of the drift tubes. The electrons are then made to drift along the tubes, in a direction parallel to the magnetic field, to a set of anode wires backed by cathode strips, which together give the coordinates of the conversion point of each photon. The region outside the drift tubes is filled with a Freon gas. Cherenkov light produced in the gas is reflected, by a set of parabolic mirrors, back into the drift-tubes, where a ring image is formed. The photons are converted in the outer 3 cm of the drift-tube gas, i.e. well separated in the
radial direction from the conversion region of the inner-cell photons. Thus the two sets of
electron rings produced by Cherenkov light from the inner and outer radiators can be
distinguished, one from the other, in a simple way. A picture of one of the drift-tube modules
may be seen in Fig. 23.

DELPHI aims to do high-quality charm and beauty spectroscopy with mass resolutions $\sigma$
of 15–20 MeV/$c^2$ for D’s and 50 MeV/$c^2$ for $B \rightarrow D\pi$. Making use of the capability for good
hadron identification and of the information that will be available from the microvertex
detector, a signal-to-background ratio $\geq 1$ should be achieved for many charm and beauty
channels.

5.3 L3

L3 is the largest of the four detectors being built for LEP. The primary design goal is to
have a system capable of measuring the energies of muons, electrons, and photons with a
resolution $\Delta E/E \leq 1\%$ at 50 GeV. To achieve this, all the detector elements are placed within
an octagonally shaped aluminium coil, having an inner diameter of 11.86 m and producing a
magnetic field of 0.5 T. Pictures of the aluminium coil sections during manufacture are shown
in Figs. 24 and 25, and of the complete magnet, with its return yoke, in Fig. 26.

The detection of electrons and photons is done in a compact electromagnetic calorimeter
consisting of tapered crystals of bismuth germanium oxide (BGO), each viewed by two
photodiodes. There are 7680 crystals in the barrel section of the calorimeter. Each crystal
measures 2 cm $\times$ 2 cm at the front face and is 24 cm (22 radiation lengths) long. They are
slotted into a honeycomb structure made of thin carbon-fibre wafer to give the pointing
geometry required. Calibration of the first half-barrel is complete, and the second half-barrel is
well under way. Excellent results have been obtained for the energy resolution, namely
$\pm 0.5\%$ at 50 GeV, $\pm 1.5\%$ at 2 GeV, and $\pm 6\%$ at 100 MeV.

A hadron calorimeter surrounds the BGO array. It is built up of layers of wire chambers and
depleted uranium to provide a fine-grained system, 4λ thick. Layers of copper, instrumented
Fig. 24  Half sections of the aluminium coil for the L3 magnet

Fig. 25  A completed section of the aluminium coil for the L3 magnet

Fig. 26  The L3 magnet
with streamer tubes, are placed outside the uranium section to give a further $2\lambda$ of material to complete the absorption of hadrons.

Muons that filter through this system ($7\lambda$ including the BGO) are detected (still within the magnetic field) in a three-layer system of wire chambers, with a separation of approximately 1.5 m between successive layers to give a good measurement of the muon momentum. The system is made up of 16 modules assembled in two octagonal sections, which can be aligned to $\pm 40 \, \mu m$. Tests carried out on one of the modules have shown that muons can be detected with a spatial resolution of $\pm 150 \, \mu m$ in each layer, well within the design specification. One of the muon detector modules is shown in Fig. 27.

Charged-particle tracking is done in a time-expansion chamber (TEC) located at the centre of the detector inside the BGO array. The TEC is only 1 m in diameter in order to keep to a minimum the volume of BGO that surrounds it, but it aims to measure tracks to $\pm 40 \, \mu m$ with a two-track resolution of 500 $\mu m$ in the $r-\phi$ plane.

As can be seen in Fig. 28, the TEC, the BGO array, the hadron calorimeter, and the forward muon chambers are all located inside a massive support tube, 32 m long and 4.5 m in diameter. The tube weighs 300 t. It also supports the muon detectors on the outside, thus making all the detectors mechanically independent of the magnet.

Unlike the other three detectors, the L3 detector cannot be moved into and out of the beam. Most if its components have therefore to be in place at an early stage so as not to interfere with the installation of machine elements near the interaction point in this region. An important milestone will be reached when the support tube is lowered into the underground area in November 1988. Only after this can the final installation of detector components inside the magnet commence.

Fig. 27  One of the 16 muon-detector modules for L3
5.4 OPAL

The OPAL detector is illustrated in Fig. 29. It is often referred to as the safe, conventional detector at LEP because it is built up of elements based on well-proven techniques used mainly at PETRA but also at the SPS at CERN.
Three basic tracking devices are located inside the 0.4 T solenoidal magnetic field produced by a water-cooled aluminium coil, 4.5 m in diameter and 6.5 m long. First there is an inner detector used for triggering, rather like in ALEPH and DELPHI, but having a substantially better $r$-$\phi$ resolution of $\pm 30$ $\mu$m which enables it to be used as a vertex detector; it is thus an essential part of the tracking system. It is surrounded by a jet chamber which constitutes the main tracking device, and which is modelled on the one used for the JADE detector at PETRA. In essence, it is a large multicell drift chamber containing 3840 sense wires, and is filled with a mixture of argon (90%) + methane (8%) + isobutane (2%) at a pressure of 4 atm. Up to about 150 ionization measurements can be made along the track of each charged particle, giving good dE/dx resolution and $\pi$–e separation up to 7 GeV/c. The $r$–$\phi$ resolution has been checked in a full-scale prototype, and has confirmed the design figures of $\pm 110$–$150$ $\mu$m. One of the end-plates of the jet chamber is shown in Fig. 30. Although the jet chamber also gives information on the $z$-coordinates of the tracks, this is improved by a system of drift chambers immediately surrounding the jet chamber, which gives a resolution in $z$ of $\pm 300$ $\mu$m.

The electromagnetic calorimeter in OPAL is built up of 12,000 lead-glass blocks, each about 24$X_0$ long. Of these, 9600 make up the barrel section and are specially shaped to give pointing geometry. They are located outside the coil, which is 1.5$X_0$ thick, and are viewed by photomultipliers. The end-caps (Fig. 31), consisting of 1200 blocks each, are in the full (axial) field of the magnet and are viewed by specially developed, low-noise vacuum phototriodes.

Fig. 30 One of the end plates of the OPAL jet chamber

284
Fig. 31  An end-cap of the OPAL lead-glass electromagnetic calorimeter during assembly

Fig. 32  The OPAL solenoid and one half of the return yoke during installation
Immediately in front of the lead-glass there is a presampler to provide information on showers that develop in the magnet coil or in the end-plates of the jet chamber. There is also a set of time-of-flight counters in the same region. The electromagnetic calorimeter has been fully calibrated and checked for long-term stability; the energy resolution obtained is $\Delta E/E = 8\% / \sqrt{E}$. The magnet yoke is instrumented with limited streamer tubes to provide hadron calorimetry, and muon identification is done by means of a system of drift chambers arranged in four layers outside the return yoke. For the barrel section, the chambers employ a neatly devised cathode readout system with segmented diamond- and triangular-shaped pads on the top and bottom of each chamber; this enables the z-coordinate to be determined to \( \pm 4 \) mm.

A picture of the OPAL solenoid and half of the return yoke, taken during installation in the underground experimental hall, is shown in Fig. 32.

6. PHYSICS GOALS

The early work at LEP will be centred on studies at and around the Z pole. The field is rich and, with each experiment aiming to collect some $10^7$ events in this region, high-precision measurements in many decay channels will be possible, with good sensitivity to small effects. A key measurement will be the determination of the Z mass. For this, an accurate measurement of the LEP beam energy is required; the plan is to do this to an accuracy of \( \pm 1 \times 10^{-4} \) by measuring the spin precession frequency of the $e^+$ beams in LEP, assuming, as seems likely, that they develop an adequate degree of polarization. In this case the error on the measured mass of the Z could be as low as $\Delta m_Z = \pm 20$ MeV/c\(^2\), giving an error on $\sin^2 \theta_W$ of $\pm 0.0004$, at which level the uncertainty in the radiative corrections dominates. If the spin-precession method cannot be used, it should be possible to achieve $\Delta m_Z = \pm 50$ MeV/c\(^2\).

A measurement of the Z width to $\pm 20$ MeV/c\(^2\) also seems to be feasible. Remembering that the partial width $\Gamma_{\nu\bar{\nu}} \approx 170$ MeV/c\(^2\), a limit could be placed on the number of neutrino types, and evidence of new channels may emerge. An alternative, and probably better, method of determining the number of neutrino types is to study the reaction $e^+ e^- \rightarrow \nu \bar{\nu} \gamma$ just above the Z while scanning across the Z peak; the cross-section for this reaction is directly related to the number of neutrino types. Another obvious but important piece of physics will be to check that the partial widths $\Gamma_{e^+ e^-}, \Gamma_{\mu^+ \mu^-}, \text{ and } \Gamma_{\tau^+ \tau^-}$ are all equal, where an accuracy of $\pm 1\%$ could be achieved.

High on the list of priorities when LEP starts up will be the search for the Higgs boson H. Theory has little to offer as regards the value of the mass of the Higgs. If it does indeed exist, and if it has a mass below that of the Z, then decay channels such as

$$Z \rightarrow H \ell^+ \ell^- \text{ (BR} \approx 10^{-5} - 10^{-7} \text{ for } m_H = 20 - 70 \text{ GeV/c}^2)$$

and

$$Z \rightarrow H \gamma \text{ (very low BR)}$$

offer ways of discovering it. Rate limitations may restrict the range of detectable masses to $m_H < 40$ GeV/c\(^2\). For higher masses up to $m_H \approx 80$ GeV/c\(^2\), it will probably be preferable to wait for the energy of LEP to be increased, and to look for the reaction

$$e^+ e^- \rightarrow ZH$$

just below the $W^+ W^-$ threshold. Here the cross-section is estimated to lie in the range 1–10 pb, depending on the value of $m_H$ and on the incoming energy of the $e^+$ beams. The most
promising H decay channel should be $H \rightarrow b\bar{b}$, leading to two jets plus the decay products of the Z in the final state; for example:

$$e^+e^- \rightarrow ZH$$

\[
\begin{array}{c}
\rightarrow b\bar{b} \\
\rightarrow q\bar{q}
\end{array}
\] four jets (73%),

$$e^+e^- \rightarrow ZH$$

\[
\begin{array}{c}
\rightarrow b\bar{b} \\
\rightarrow \nu\bar{\nu}
\end{array}
\] two jets + missing mass (18%),

$$e^+e^- \rightarrow ZH$$

\[
\begin{array}{c}
\rightarrow b\bar{b} \\
\rightarrow e^+e^- \\
\rightarrow \mu^+\mu^- \\
\rightarrow \tau^+\tau^-
\end{array}
\] two jets + $\ell^+\ell^-$ (6%).

The search for the Higgs will be no easy matter. Rates will be low, backgrounds could be high, and there could be further complications if the top quark can be produced within the LEP energy range.

When the energy of LEP approaches its full design value of about 100 GeV per beam, operation above the $W^+W^-$ threshold becomes possible. This will provide a unique opportunity to study the three-boson couplings $\gamma WW$ and $ZWW$, which underpin the electroweak theory. The reaction $e^+e^- \rightarrow W^+W^-$ is dominated by the three diagrams shown in Fig. 33. Large cancellations between the amplitudes are expected, so measurements of this reaction provide sensitive tests of the theory.

It will be important to measure the mass of the W once the energy of LEP is high enough. There are various possible approaches to this: for example, measuring the excitation function for producing $W^+W^-$ pairs; looking at the two-jet invariant mass from $W \rightarrow q\bar{q}$ decays; or studying the $W \rightarrow \nu e$ invariant mass. Each of these could give $\Delta m_W \approx \pm 100$ MeV/c$^2$, so a final experimental uncertainty of $\Delta m_W \approx \pm 50$ MeV/c$^2$ seems achievable.

The higher LEP energies would also seem to be needed in the search for the t-quark, assuming that it will not already have been discovered in experiments at the p$p$ colliders at CERN and Fermilab. Predictions of its mass cover a wide range, but there are good reasons to expect it to lie in the range 55–250 GeV/c$^2$, the lower end of which will be accessible at LEP.

The above gives a selection of some of the physics aims of the LEP experiments. Clearly there are many other interesting studies to be made at LEP: for example, measurements of the longitudinal and transverse polarization of the W's; forward–backward asymmetries; search for supersymmetric particles; and so on. There can be few accelerators built to date where the richness of the physics return has been better assured from the start.

Fig. 33 Dominant diagrams for the reaction $e^+e^- \rightarrow W^+W^-$
7. EVOLUTION OF THE LEP PROGRAMME

The commissioning of LEP is due to start in July 1989. The first tasks will be to carry out the final checks of the machine hardware and to establish stable beams at intensity levels that are sufficient for their properties to be measured. This could take some months, since many of the machine components will only just have been installed. As already mentioned, all four experiments are expected to be in place by the time commissioning starts, although it is not planned to power up their solenoidal magnets until after the initial 'warming up' period of the machine.

During 1989, priority will be given to getting the machine understood and running reliably, although obviously every effort will be made to produce the first collisions as soon as possible. The goal is to provide the experiments with a working machine at least by early November 1989 so as to have, before the end of the year, a minimum of five or six weeks of steady running with enough luminosity to check out the detectors, to gain experience in operating them at LEP, and to have a first look at the physics. The hope is to reach an integrated luminosity of a few inverse picobarns at each interaction region during this first running period, and to record up to $10^5$ events per experiment at and around the Z peak.

Looking further ahead, it is planned to run LEP for 3000 to 4000 hours per year from 1990 onwards. Depending on the physics priorities, the running will probably be concentrated in the region of the Z peak until about $10^7$ events per experiment have been collected. It is hoped to reach the design luminosity during the course of 1990, and to obtain for each experiment some $10^6$ Z events in that year and perhaps $3 - 4 \times 10^6$ events in both 1991 and 1992. This running will be with the electron and positron beams unpolarized. There is, however, growing interest in operating LEP with longitudinally polarized beams, which would make possible a number of independent and stringent tests of the Standard Model. At present the feasibility of operating LEP with useful levels of longitudinal polarization is not known. However, work is in progress to examine the physics case and to study the relevant machine and experimental problems.

As already noted in this report, LEP has been designed to operate at energies above the $W^+W^-$ production threshold. During the early years of operation, the energy will be limited to 55 GeV per beam by the power obtainable from the copper RF cavities initially installed. The programme for operation at higher energies will depend on the progress made in developing superconducting RF cavities, which will be used to provide the additional power required. The work done to develop prototypes at CERN has given encouraging results, and efforts are now being made to transfer the technology to industry. Once delivered and tested, the cavities could be installed progressively during the normal planned shutdown periods for LEP. A possible schedule for upgrading the energy capability of LEP is shown in Fig. 34, assuming that

![Fig. 34 Evolution of the LEP programme](image-url)
the development and production of superconducting cavities proceeds at a reasonable pace. As can be seen, it may be possible to run LEP up to 77 GeV per beam by mid-1993. Other machine-related issues will need to be settled before going further than this: for example, whether to replace the copper cavities by superconducting cavities, and whether it will be necessary to extend the RF system beyond its present location in the straight sections around Points 2 and 6 to those near Points 4 and 8.

At the higher energies the peak luminosity should reach at least $2.5 \times 10^{31}$ cm$^{-2}$ s$^{-1}$, and an integrated luminosity of about 200 pb$^{-1}$ should be possible in a normal year's running. Of course, the precise schedule for the operation of LEP will depend on how the subject evolves during these years; whether the Higgs boson or the t-quark are discovered, or whether something quite unexpected emerges.

In any event, the plan is to complete the energy upgrade as soon as possible. It will then be possible for the experiments to start looking at the physics above the $W^+W^-$ threshold, where LEP offers unique opportunities, and to extend to higher energies the search for the top quark and the Higgs boson if they have not been found by then. It is clear that LEP will have a full programme, lasting for at least ten years.
DATA ACQUISITION AND RECORDING

C.N.P. Gee

Rutherford Appleton Laboratory, Chilton, Didcot, UK

Abstract

This series of lectures examines the latest techniques for Online data collection and recording, with emphasis on the needs of larger HEP experiments. Discussion of data collection includes typical schemes of parallel data readout to reach low deadtime and high processing power. The complex control systems for concurrent running, integration and testing are also mentioned. The presentation includes a detailed review of the new generation of data recording equipment. Conventional data storage devices are compared with commercial high density optical disks and magnetic tape cartridges which seem likely to become the new standard recording media.

1. Event Acquisition

1.1 Introduction

Use of computers for Online data acquisition and recording in HEP experiments extends back only to around 1970. This date coincides with first Cern Computing School. The earliest online systems allowed data recording on Magnetic tape, with perhaps a small number of histograms being accumulated and displayed. Use of CAMAC as a standard interface between computers and detectors started in 1972/1973.

A Standard at CERN was set by the EMC experiment [1], which ran in 1979 using four PDP-11 machines. The online processing power, even with 4 computers, was limited and far less than offline machines, so online work was concentrated on data reading and recording, with tools for some user-written checking of the data. At about the same time, 32 bit minicomputers appeared, and were swiftly adopted for online use in experiments. Use of CAMAC was widespread by this time, and could yield readout speeds of 700Kbytes/sec. A Cern variant known as "ROMULUS" could achieve readout speeds of 1.3 Mbytes/sec. It included provision for several controllers performing CAMAC readout in parallel, all data nevertheless being read into the minicomputer in a single DMA operation. ROMULUS generated a position-independant data structure so that variable length detector data could be supported.

The basic readout mechanism has been and is similar in all experiments. A Physics event occurs, and is manifested as analogue pulses emerging from the different particle detectors. Selected analogue signals are converted by discriminators to digital pulses, which are processed in simple electronic logic leading to an "Event Trigger". This is distributed to Digitisers (ADC, TDC, pattern units) which receive and convert all detector signals to digital binary values. When digitisation is complete, an Interrupt signal initiates fast readout into the online computer memory. When readout is complete, the experiment is restored to state where it will accept the next event.
The data read in response to one trigger constitute one "Event". Events are held in a memory buffer until copied to a storage medium (normally magnetic tape, sometimes others). Holding the data in memory gives an opportunity for other user programs to access events on a sample basis, and to check thereby that the experiment is working correctly.

1.2 The problem of Data Rates

This strategy applied to large experiments can lead to an impossibly large data volume, with all the interesting physics lost. For example, the OPAL experiment at LEP has a possible raw event rate of 45Khz, with events of 3.8Mbyte. Such problems are typically overcome using Multiple Triggers and Data Compression.

There is a simple relationship between the dead time per event D, the number of events read per second N, and the raw trigger rate R. The proportion of time dead is ND, so the live time is (1-ND). The number of events read is thus

\[ N = (1 - ND) \cdot R \]

or

\[ N = \frac{R}{1 + DR} \]

When DR is small, the accepted event rate N approaches the raw trigger rate R, and all physics event data can be read. When DR is large, the accepted rate approaches 1/D, and the experiment is said to be deadtime limited. The trick is to attempt to make both R and D as small as possible.

1.3 Multiple Triggers.

The function of the trigger in large experiments is to identify the small number of interesting events, and to generate an Interrupt signal only for these. The problem is to identify and reject unwanted events, arising possibly from unwanted copious physics or noise signals generated by the accelerator.

Many large experiments now use multi-level triggers where successively more complex decisions are made on successively lower data rates. The following description of the OPAL scheme is typical of the LEP Experiments.

1.3.1 Level 0 trigger

The bunches of electrons and positrons in LEP collide at 45Khz. Any collision may in principle lead to an interesting event. A gate generated from the bunch crossing signals is applied to all digitisers (ADCs, TDCs, FADCs), which start digitising just after each bunch.

1.3.2 Level 1 Trigger

To be effective, the level 1 trigger must reject most candidate triggers offered to it. But to minimise deadtime, these rejections must be fast. The critical time at LEP is the 22\(\mu\)sec interval between beam crossings. Decisions made within this time will
allow every bunch crossing to be examined as a candidate, so the trigger system itself will not impose any deadtime.

In OPAL, a complex decision is complete in 16μsec (Figure 1). The decision involves matching data patterns from several detectors (Tracking, Time-of-Flight, Electromagnetic and Hadronic Calorimeters, and Muon Detectors), in a 144-element θ-φ matrix. A negative decision is delivered at 16μsec, and clears all digitisers ready for the next bunch crossing. A positive decision commits the experiment to readout the detectors (5-10msec deadtime).

![Figure 1: OPAL Level-1 Trigger Decision Timing](image)

1.3.3 Level 2 Trigger

The level 2 trigger is applied to the reduced data rate of 10-20Hz accepted by trigger 1. The primary function is to repeat the algorithm of trigger 1, but using the full digitised precision of all detectors which is now available. Much tighter cuts can be applied to reject background events - for example to ensure that track vertices originate from the beam-beam region. Event data rates emerging from the level 2 trigger are around 1Hz.

Since the second level trigger is working with track and calorimetry data, it is able to flag an enriched sample of potentially interesting events for preferential processing by the third level trigger.

1.3.4 Level 3 Trigger.

All events accepted by the level 2 trigger are recorded. It is nevertheless useful to reconstruct and identify events of particular interest which should be given priority in further studies. This scanning is done by the level 3 trigger for as many events as possible, usually by running part or all of the offline reconstruction program in a farm of suitable processors. Identified events may be copied to a second data stream for fast processing.
Running the main analysis program as part of the online event stream in this way also reduces the offline analysis load (some events are already fully analysed), but requires care. Correct detector calibration constants must be known online. Also, the data volume may actually increase, so careful tuning is needed.

1.4 *Online Data Compression.*

The concept of multiple triggers (reducing the event rate) goes hand-in-hand with schemes for compressing the data (reducing the event size). Three methods are commonly used together.

1.4.1 Zero Suppression

Some readout systems include a sparse data scan in hardware, to omit empty channels and insert pointers to address the remaining occupied ones. Note that the readout using sparse data scan is not necessarily faster - the contents of each digitiser may still need to be read before suppression can be applied.

Some zero suppression algorithms require implementation in software - for example if a signal includes component channels only just above pedestal values. Zero suppression is best suited to very sparse data. Here a dramatic data reduction is possible. If many channels are full, however, the data can actually double its initial size.

1.4.2 Clustering

Many detectors produce a cluster of digitised values for one particle (one track may traverse many detector elements, or one detector signal may be sampled many times). In such cases, grouping correlated fired channels together allows most pointers to be removed, and can reduce data volume by factor of 2. This may involve re-ordering the data to collect physically adjacent detector elements together in one cluster, which is also useful for event reconstruction, since all channels contributing to one detector hit are processed together as one unit.

1.4.3 Fitting

Zero suppression and Clustering require only basic knowledge of the detector. Provided that the algorithm is correct, the raw data can be rebuilt if necessary. If complete calibration information is available, the clusters can be converted to a physical quantity - Energy, Position. As a tool to save space, this works best on data with a large cluster size, because several numbers may be needed to describe the fit results.

After fitting, the original data may be discarded, provided that calibration constants are correct. Early in the life of an experiment, it is seldom clear that calibration is in fact entirely accurate, so raw data may need to be retained. Furthermore, floating point numbers may be generated, which can cause data handling problems later. So overall, use of fitting as an online data reduction tool needs to be viewed with caution.
1.5 Parallel readout and Processing

In a large experiment, triggering and data compression calls for a substantial amount of online processing. Current technology helps by providing microprocessors of substantial power which are capable of being used in parallel architectures.

To give the shortest readout time, it is becoming common to use several independant parallel readout and processing channels, each tuned for one detector. The event interrupt must be distributed to all readout processors, which transfer data concurrently from all detectors. Further triggers must be inhibited until the slowest detector readout is complete. Events are then distributed and collected from successive processing layers. Since the different parallel streams may complete at different rates, software must decide if event order is to be maintained or not. Allowing events to flow through out of sequence gives the maximum throughput, but imposes difficulties on the subsequent analysis.

The routing algorithm can become very complex in practice - data from different trigger types may require routing to different processors loaded with corresponding processing algorithms, software has to survive with some of the parallel channels inoperative, and individual detectors may need private readout. This is the problem of partitioning which is discussed later.

1.6 Monitoring in a multi-level Parallel system.

It is desirable to transfer monitoring functions to lower level processors when possible, since this offloads the higher level, more expensive machines. Some data can only be sampled at particular points in the readout tree, for example raw data bit patterns (which should be checked for dropped bits and crosstalk), and intermediate results from clustering or fitting.

However, with multiple levels of trigger, the low-level programs monitor events before the trigger sample is pure, so physics monitoring has to be applied higher up the tree. This is reasonable, as it probably involves more complex calculations with a higher proportion of floating point operations. If intermediate values from the trigger decisions are available, monitoring can be applied selectively to different data streams. Dataflow statistics must also be accumulated, to check the performance of the readout system itself.

2. Data Acquisition Control

2.1 Characteristics of distributed acquisition systems

With few exceptions, experiments have in the past used a small number of computers. Control of the data acquisition system and the experiment in such cases is straightforward, as the systems have dedicated roles. The number of software modules is limited, reflecting the restricted processing power. With a limited user community, a restriction to one Online activity at once is usually acceptable (although the EMC software referred to previously could perform a limited set of concurrent runs). This simplicity of design encourages (hopefully) rapid fault isolation, and easy reconfiguration to circumvent isolated hardware problems.
Much greater control problems arise in the large experiments, because the experiment readout is spread over big distances and is so complex that reliability cannot be assumed.

2.1.1 What the users Expect

Workstations with windowing capability are now commonplace. Users expect easy access to the experiment, with all relevant information available on one screen. The possibility should exist to diagnose, test or calibrate a detector while data taking is in progress, with a degree of protection from other users doing the same thing. Users also want minimal interaction with system complexity. Simple overall control is needed to run the experiment, with as much automatic fault detection and recovery as possible. To provide for all of these, developments are needed in information management, central and distributed control, and Infrastructure including networks and databases.

2.2 Information Management

Tools are needed to control information flow (not events). In a distributed system, the information sources and sinks are neither obvious nor fixed.

2.2.1 Error Reporting

The previous generation of Vax-based Cern experiments already used a central message reporting system. Error messages were numbered, with supporting software to send messages to display screens and to allow individual messages to be suppressed.

A new Error Message Utility [2] has been developed by Cern for the LEP experiments. To use this, programs generate named messages, and a decoding step associates attributes with each named message. A message router uses the attributes and message names in conjunction with a route map to despatch the messages to consumers (messages are duplicated if necessary). The routing can be changed to reflect system or user activity, and system designers can write decision-making software to perform actions on specific errors. In this scheme, there is no fixed master system - the topology is determined by the route map.

2.2.2 Status Reporting

The majority of experiments generate a status display, to provide up-to-date information on activities within the online system. In the previous generation of Cern Vax software, status information was handled by the same mechanism as error messages. Useful items of information, such as event counts, were sent to the central message system, where a formatting file directed the text to a specific screen location.

In the modern large experiments, many programs generate status information. Some of this may be of interest for presentation on display screens, while some is used by other software. The flow of status information changes with time, as programs and people come and go. Handling this by status transmission to a central point implies a heavy network and processing load.
OPAL has developed an information management system somewhat like a network name server to address this problem. The information values (status items) are stored locally, and retrieved over the network only when required. Clients of the system retrieve information values by name. Detector groups develop their own suite of status display pages, using a common display tool.

2.2.3 Histogram serving and presentation

Online histograms are almost never drawn on a display screen by the program responsible for filling the histogram. Histogram filling can then be done in real time programs, while the complexities of plotting the bins are kept to the presenting program. Information was at one time transferred between the filling and drawing programs via constantly updated disk files, but the transfer is now commonly via shared memory or a network.

Development is underway within the Physics Analysis Workstation (PAW) [3] software to support histogram data retrieval via a server. This is an attractive option, since the viewing tools for Online and Offline analysis would be similar. Activities such as zooming, overlay and comparison processing are performed in the workstation. At the present time, many more tools are needed to complete this project, for example collection of histograms from many parallel processors, invisibly to the user, or retrieval of reference histograms for the same run type or conditions.

Several experiments have automatic histogram checking tools. For example, the Jade experiment at DESY did automatic checks on histogram filling rates. At LEP, the Delphi team have developed an expert system which correlates histogram contents with detector electronics grouping. All these new tools are in their infancy, and it will be interesting to observe if they are widely accepted.

2.3 Central and Distributed Control

2.3.1 Slow Control

Control of the voltages, gas mixtures, etc, required for correct detector operation has seldom been systematic in online systems. At Cern, such control was occasionally built into monitoring programs, and more often done by hand. There has seldom been effective use of any data transfer between the accelerators and event acquisition systems.

All the LEP experiments have control systems operating semi-independently of data taking (Figure 2). Detector gases can be checked even when no data-taking is in progress, detector voltages are lowered when the accelerator is filling or about to dump beam, and power supplies are monitored to detect failures. A connection to the run control system is still of course required to prevent operator voltage adjustment while data taking and to allow voltage manipulation during detector calibration.
2.3.2 Run Control

UA1 is probably the last large experiment to run without central run control in software. In consequence, new run control systems are still being developed. Several experiments have adopted the concept of partitions for independent runs. The notion of independent runs implies use of resources, e.g., a detector, a readout path, with conflict between different runs prevented by formal resource allocation. A complex trigger system is needed for complete run independence, and event and data routers must understand the partitioning system.

2.3.3 Program Control and Access

All experiments are using or developing a “Human Interface”, which is supposed to provide a comfortable working environment for an operator in dialogue with several programs. Initially the human interface software was devised to switch ownership of terminals between programs. This is not necessary on workstations, where the primary function is to supervise network access from terminals to programs executing on remote processors.

The human interface software should also provide a uniform mechanism for starting user programs in several types of processors, standard styles of dialogue for the operator and standard routines and structures for user programs.
2.4 Infrastructure - Networks and Databases

2.4.1 Networking

There is general consensus about the network functions needed in the large experiments - the functions include task-to-task messaging and/or remote procedure calling, remote file and record access, remote login, and mail and broadcast systems.

It may be seen from the list above that as many functions are needed for software development as for execution of the final running system. Lack of standard protocol implementations on some processors presents a major difficulty, and many ad-hoc solutions have been (are being) developed. Most experiments now use Ethernet to carry network traffic (Figure 3). Good passive access for WAN users is also necessary, to allow home based detector experts a reasonable level of participation in data taking while providing adequate protection to prevent very remote users from ending the run.

2.4.2 Databases

All LEP experiments have spent substantial efforts in evaluation of databases, and all now use database products of some sort. Not all experiments use Relation-
al Databases, and some have developed their own. Databases exist both for offline and online use, and several copies of the offline database may exist in different institutes. Information flows both to and from the experiment, so it is important to define which part of which database is the master. There is no discernable uniformity between the LEP experiments at the present time.

In the author’s view, a relational database is (probably) essential online. But bulky data (e.g., calibration data) is not necessarily best stored there. One should hesitate to build commercial database calls into the Online software. However, relational databases are highly suitable for run and tape administration and book-keeping.

3. Data Recording

3.1 Characteristics of HEP Data

Online HEP data recording needs are fortunately rather similar to those for fast disk backup devices (where there is a much larger consumer market). Data arrives at random rates, but frequently very fast, and consists of logical records (events) of random lengths. Data are read and written sequentially, normally in long files whose size is limited by capacity of the recording medium. Tapes are usually read completely only once or twice, although samples may be read many times during software testing.

Unlike backup handling, the data is repetitive, so loss of a few complete events is not serious. More difficult is data transfer between different types of computers, whose different number representations are a constant source of annoyance.

3.2 Remote Data Recording

At some high energy physics institutes (e.g., DESY), data from experiments is recorded in one central location. The event data is sent over fast links and stored on disk, and then copied to tape when disk is nearly full. This is extremely attractive from the experiment viewpoint, as there need normally be no tape handling at the experiment (where conditions are not ideal for computers anyway). Tape capacity at the experiments is needed only for local operations and to be able to continue data taking if the link breaks.

Remote recording is less attractive viewed from the recording centre, where tape mounts must be made in semi-real time, and disks and links are permanently busy. Moreover, storage ring experiments may generate their peak rates may arise concurrently (e.g., if beams are noisy).

The main overall benefit is clearly that the computer centre and the experiments get access to the latest recording technology, professionally operated in clean conditions, and it is (relatively) easy to change to a new recording medium.

3.3 Conventional Tape devices

Half-inch magnetic tapes have been a great success. Many good and a few bad points can be identified [4]:

299
1. Data Rates for reading and writing are generally similar to other computer peripherals.

2. There are good international standards which have been widely accepted by equipment manufacturers.

3. Tapes are cheap to buy and easy to copy.

4. Recorded data has a shelf life from 5-10 years, which is adequate for most users.

5. The capacity of a reel is reasonable.

6. Manufacturers have accepted tape as a device class requiring its own software support.

Some negative points are:

1. Tape handling is operator-intensive - one of the major costs in large computer centres.

2. Data on tape is not automatically catalogued - users must remember for themselves which tape contains which data.

3. Access to data is strictly sequential only.

4. If a tape is read or written more than a very few times, there is a substantial rise in error rate.

3.3.1 800 bpi NRZI tape

Data recording using the Non-Return to Zero (NRZ) method was patented in 1956 by Phelps [5]. His tape had separate YES and NO tracks. This ensured one transition on tape per recorded bit, to allow self-clocking (i.e. the timing information used to read the data bits is generated directly from the tape itself).

The NRZ technique was modified by IBM (NRZI) to record only YES signals, and add a track for odd parity. Each physical record starts with tracks making a flux transition to +1 if the data bit is set to zero, and a flux transition to -1 if the data bit is set to one. Subsequent set bits reverse the recording current, while clear bits do not. The parity track ensures at least one transition per stripe, so the system is self-clocking.

The OR of transition signals on read provides the strobe against which all tracks are read. Since the tape stretches, not all bits pass the head at identical times, so a signal sampling window is needed. This feature is known as SKEW, and variable skew times limit the density with which NRZI can be used. NRZI density started at 100 and then 200 characters/inch. By adding a cyclic redundancy check (CRC) stripe, a longitudinal check (LRC) stripe (ensuring an even number of bits is read in all stripes), and moving from 7 to 9 recorded tracks, the usable density was raised to 556 and finally 800 bpi.

At 800 bpi, the data is recorded in blocks separated by inter-record gaps of 0.6 inch - long enough for tape mechanics to stop and start again. A special data pattern, often referred to as a tape mark, with 1 row of only bits 2,3,8 set, followed by
crc of 0 (only parity bit set), and an LRC indicates end of file. This format is described in detail in [6].

3.3.2 1600 bpi PE Tape

A series of zero bits in one track on NRZI tape effectively yields one long magnet, and leads to technical problems with the head and amplifier which must have a wide frequency response. Phase Encoding [7] was introduced (1966) to overcome this problem.

The scheme uses Manchester Encoding in each track. Each bit is represented on tape by 1 or 2 magnetisation changes in a timing window. The direction of transition in the centre of the window gives the bit value, and a transition at the window start is made if needed to set the field level for the central transition. This implies that only two data frequencies are recorded on tape.

PE also addresses the issue of automatic skew compensation. Each physical block on tape has preamble, data, CRCs and postamble. The preamble and postamble consist of 40 bursts of 0 bits in all tracks and 1 burst of all 1 bits. The tapedrive reading circuits use the preamble to set up electronic delays to remove skew. The postamble allows deskew when reading tape backwards (sometimes used in error recovery). A tape mark is a series of 64 to 256 transitions at 3200 ftpi in tracks 2, 5 and 8.

To allow tape drives to distinguish automatically between 800 and 1600 bpi tapes, the PE standard includes an IDentification Burst of continuous flux transitions in track 4. This starts at least 4 cm before the BOT mark and continues into the BOT reflective marker. 800bpi NRZI tapes must be erased in this region.

This format is described in detail in [6].

3.3.3 Error handling and data Encryption techniques

A detailed discussion of error recovery is beyond the scope of this presentation, and for further details the reader is referred to [8]. Redundant information has been carried on all tapes from the 800bpi NRZI onwards. The most common source of error is loss of a small area of magnetic material on the tape leading to 1 dropped bit.

An error reading an 800 bpi NRZI tape is detected in the affected stripe which yields a parity error, and also in the CRC and LRC checkwords. The track in error can be deduced by comparing the read and calculated CRC rows (an algorithm involving small number of bit shifts is described in [6]). Then the bit in error can be repaired, and the calculated LRC compared (as a check) with the recorded LRC. The same technique can be applied in 1600 bpi tapedrives, and in some applies automatically.

These methods will repair one lost bit per block. Encryption techniques allow many lost bits per block. The following simple example from [8] (Figure 4) illustrates a system for encoding numbers 0-3 which is proof against single bit errors and detects some higher errors. Any of the received values with only a single bit error will map correctly onto to original value.
<table>
<thead>
<tr>
<th>Values to Transmit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted</td>
<td>11000</td>
<td>00110</td>
<td>10011</td>
<td>01101</td>
</tr>
<tr>
<td>Other</td>
<td>11001</td>
<td>00111</td>
<td>10010</td>
<td>01100</td>
</tr>
<tr>
<td>Received Words</td>
<td>11010</td>
<td>00100</td>
<td>10001</td>
<td>01111</td>
</tr>
<tr>
<td>Words</td>
<td>11100</td>
<td>00010</td>
<td>10111</td>
<td>01001</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>01110</td>
<td>11011</td>
<td>00101</td>
</tr>
<tr>
<td></td>
<td>01000</td>
<td>10110</td>
<td>00011</td>
<td>11101</td>
</tr>
<tr>
<td>Double errors</td>
<td>11110</td>
<td>00000</td>
<td>01011</td>
<td>10101</td>
</tr>
<tr>
<td></td>
<td>01010</td>
<td>10100</td>
<td>11111</td>
<td>00001</td>
</tr>
</tbody>
</table>

*Figure 4:* Encryption scheme for Automatic single-bit error Correction

3.3.4 6250 bpi GCR Tape

Data on 6250 bpi tapes is recorded in “storage groups” of 10 stripes of 9 tracks. Each group contains 7 data bytes + 1 ecc byte, encoded in a Hamming code (see [8]) to yield 9 stripes and followed by an ECC stripe. This encoding allows detection and correction any 1 or 2-bit errors per data group. Synchronisation groups are inserted after every 158 storage groups (0.25 inch) to ensure good deskew. At the end of the data, a CRC block is added to ensure residual error detection.

This encoded data is written on tape using the NRZI technique at 9042 flux reversals per inch. Interblock gaps are 0.3 inch, reflecting the improvements in tape handling technology.

A tape mark consists of a group of all 1s in all except tracks 3, 6, and 9. An ID burst is recorded before and up to the BOT mark in the PE frequency range on track 6. Following this is a Read Amplification Burst of all 1 in all tracks, allowing tape drive hardware to adjust its signal amplification levels.

The full GCR tape specification may be found in [9].

3.4 New Recording systems

It is fortunate that the bulk online data recording needs of HEP experiments are similar to those of system managers making fast backup copies of disks. Demand for fast and compact backup media is pushing large and small manufacturers to produce cheap fast archive devices.

3.4.1 Streamer Tapes.

Streaming tape is not really a new recording technique, but a change in tape drive technology. Streaming drives assume that data is always available, and so move the tape at constant speed through inter-record gaps. Data flow to tape is turned on and off at the appropriate times.

If the data flow into the tapedrive stops, the drive stops and backspaces. When more data is available, the drive starts and resynchronises itself with previously recorded blocks before starting output again. This means that a streaming tapedrive behaves poorly with intermittent dataflow. To ensure that the data really is avail-

302
able, many streaming tapes include internal cache data buffers, and some regulate the streaming speed according to cache occupancy.

3.4.2 The IBM 3480 Cartridge tape

Several manufacturers are producing compact cartridge tape systems, (e.g. DEC TK50, TK70), but it would seem to be the IBM 3480 which is gaining wide acceptance and is now being copied by smaller specialist companies.

The IBM 3480 uses an 18-track head and high-quality half-inch tape in a small cartridge. Data is recorded linearly down the tape at 972 flux changes per mm, yielding 38Kbytes of user data per inch of tape. The tape moves at 2 metres/sec, giving a 3Mbyte per second data rate. The drive includes a 512Kbyte buffer memory, with a controller with read-ahead and write-behind capacity. It appears to have streaming characteristics, since the reposition time is long (80msec). The tape capacity is approximately 200 Mbytes, although it seems likely that this will be increased soon. It is claimed [10] that linear densities in excess of 100,000 bpi have been achieved in the laboratory, while present head technology is capable of many times the current figure of 18 tracks. Cassette capacities of 1 Gbyte are clearly within reach.

The data encryption and Error recovery scheme is described in [11]. The 18 tracks are divided into 2 sets of 9, composed of 7 data and 2 checking tracks (The two sets are in fact interleaved on the tape). One of the check tracks on each side contains vertical parity for that side. The other check track contains a diagonal parity across the full tape. The 18-track record is partitioned into blocks, each containing 14 bytes plus 4 check bytes along the tape, and the bytes are further encoded into 9-bit patterns where the run of consecutive zeros is limited to the range 0-3. It is claimed that the write and read reliability of this system are 1 in $10^9$ and $10^{12}$ respectively.

The major advantage of this device derives, in the opinion of the author, not from its storage capacity and speed (which are at present only 30-50 percent greater than existing devices), but from the compact size and particularly from the existence of stacker systems. These allow sufficient cartridges to be loaded for substantial data handling operations to be completed without operator intervention.

A novel feature is that the positions of BOT and EOT are established not by markers on the tape, but by comparison of the rotation speeds of the two reels. In use, this means that EOT is not always found at the same position, and that worn tapes cannot be repaired by cutting away the first feet of tape.

3.5 Helical Scan Tape devices

Digital video recorders have been around for some time but have not been widely used. There has been a recent spate of helical devices aimed at smaller computers and with a wide range of interfaces (DEC Qbus, Unibus, SCSI, Apollo...). Several large computer centres are evaluating these devices because of their low device and media costs.

An example is the Exabyte tape system EXB-8200. This uses a digital quality 8-mm tape in a conventional domestic cartridge. A capacity of 2.3 Gbytes is achieved on 360 ft of tape. The head rotates at 1800 rpm, giving an effective linear speed of 150 ips with tracks 3 inches long. The tape moves at just over 1 cm/sec.
Each diagonal track contains 12720 bytes in 8 fixed-length blocks with 1024 user bytes, plus 4528 bytes of addressing and correction data.

It is too soon to judge the level of user acceptance of these devices.

3.5.1 Optical Devices

A few years ago when Write Once Read Many times ("WORM") Optical disks first appeared, it seemed possible that they would provide the next generation of bulk storage media. While there is now lively growth in this area, the expected revolution has not (yet) appeared. It is instructive to examine again the list in 3.3 on page 12 but now with reference to optical disks.

1. Data Rates for reading and writing are generally not very high, normally not more than 250K byte/sec.

2. Media and recording patterns are not standard. Disks exist at least in 5.25, 8, 12 and 14 inch formats. The major manufacturers appear to be not yet very interested.

3. Media costs are substantially above the prices of conventional tapes (currently 3 times conventional tapes, 6 times cartridge prices).

4. Estimated recorded data shelf lives vary from 30-100 years. This is comfortable but really useful only to a minority of users.

5. The capacity of 1 disk (typically 1-2Gbyte on a 12- or 14-inch platter) is disappointing in view of the large size.

6. Several different versions of software support are used to connect the disks into operating systems.

Moreover, only some of the negative points made earlier are improved:

1. Handling is still operator-intensive. Some "juke-box" systems exist, but they address primarily the problems of repeated access to a more or less static data library, not continuous exchange of disks.

2. Some optical disks with disk structure do contain internal catalogue information. Of course, the user must still remember which disk to use.

3. Random access to data is possible if the disk is appropriately structured.

4. A disk can be read a large number of times with no degradation in error rate.

At present, WORM devices still have to be treated as an emerging technology. Individual products from specialised companies no doubt work as advertised, but it seems clear that magnetic media will be used for the foreseeable future.
Bibliography


2. Burkimsher, P C. *The EMU user guide*, Cern DD/OC/CS


6. International Standards Organisation, ISO 1863-1976, *Information processing - 9 track, 12.7 mm (0.5 in) wide magnetic tape for information interchange recorded at 32 rpm (800 rpi)*

7. International Standards Organisation, ISO 3788-1967, *Information processing - 9 track, 12.7 mm (0.5 in) wide magnetic tape for information interchange recorded at 63 rpm (1600 rpi) Phase Encoded*

8. Wesley Petersen, W. *Error Correcting Codes, MIT Press (1961)*


NEW TECHNIQUES FOR DATA ANALYSIS
IN HIGH-ENERGY PHYSICS

Richard P. Mount
California Institute of Technology, Pasadena, California, USA

ABSTRACT
If Thomson and Rutherford were given a tour of CERN, SLAC or Fermilab, they would have little difficulty in comprehending the principles of the huge accelerators as awesome, but logical, extensions of their own experimental techniques. However, the path leading from the fundamental collisions produced by the accelerators to the publications in Physics Letters would amaze, confuse, and quite possibly distress them.

In an attempt to alleviate such confusion and distress I will describe the physical and logical foundations of data analysis techniques in high energy physics experiments. Particular stress will be placed on new techniques involving interaction, graphics, workstations and databases, while not ignoring older ideas which continue to be important.

1. INTRODUCTION

These lectures will concentrate on new techniques in HEP data analysis. However, since the intended audience includes students with little or no HEP background, I have provided a brief introduction to HEP itself, and a summary of the logic behind all physics analysis techniques, old and new. Unlike the less fortunate students at the school, the reader is invited to skip over the sections which insult his intelligence or his erudition.

1.1 Why HEP?

Our fundamental understanding of the universe has two frontiers:

1. High Energy Physics,
   very small space-time scales,
   high energy densities.

2. Cosmology,
   very large space-time scales (at least now).
We believe that HEP re-creates the conditions in the universe when it was only nanoseconds or picoseconds old. Thus HEP and Cosmology are not separate studies, since they merge in the early stages of the evolution of our universe.

HEP experiments have revealed the tantalisingly simple structure of matter and forces shown in Fig. 1. All matter is made up of half-integer spin particles known generically as fermions. Quarks carry 'baryon number' and leptons don't. All the forces are explained by the exchange of integer spin particles know generically as bosons. Figure 2 shows how the exchange of bosons can explain both scattering and new particle production using \( e^+ e^- \) collisions as an example. There is an appealing symmetry between the three 'generations' of quarks, and the three 'generations' of leptons. There is also an appealing similarity between the way in which all forces work, since they are all described by 'gauge theories' which require physical observables to be unaffected by a very general set of transformations. (For gravitation this is still only a conjecture).

---

<table>
<thead>
<tr>
<th>THE WORLD CONTAINS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATTER (FERMIONS)</td>
</tr>
<tr>
<td>quarks</td>
</tr>
<tr>
<td>u c t</td>
</tr>
<tr>
<td>d s b</td>
</tr>
<tr>
<td>and</td>
</tr>
<tr>
<td>leptons</td>
</tr>
<tr>
<td>e ( \mu ) ( \tau )</td>
</tr>
<tr>
<td>( \nu_e ) ( \nu_\mu ) ( \nu_\tau )</td>
</tr>
<tr>
<td>FORCES (GAUGE BOSONS)</td>
</tr>
<tr>
<td>Electroweak force</td>
</tr>
<tr>
<td>( \gamma, W^+, W^-, Z^0 )</td>
</tr>
<tr>
<td>Strong force</td>
</tr>
<tr>
<td>gluon</td>
</tr>
<tr>
<td>Gravitation</td>
</tr>
<tr>
<td>? graviton ?</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>The Higgs</td>
</tr>
<tr>
<td>( H^0 )</td>
</tr>
</tbody>
</table>

*Figure 1* The simple structure of matter and forces.
Figure 2  How $\gamma$ and $Z^0$ bosons make $e^+e^-$ interactions happen.

This structure is tantalising because, while it is fairly simple and consistent with everything we observe, it contains too many arbitrary numbers for intellectual comfort. The most offensive arbitrary numbers are the masses of the fermions. The natural mass scale of the ‘world’ shown in Fig. 1 is of the order $10^{15}$ to $10^{19}$ GeV, and the observed masses of the order $10^0$ GeV must either be due to outrageous coincidental cancellations, or something new. There is also no explanation of why the 6 quark ‘flavours’, or the apparently corresponding 6 types of lepton should exist at all.

The masses of the bosons are in marginally better shape. The photon and the gluons have zero mass which is a comfortable value giving theorists few problems. The $W$ and
$Z$ bosons are the heaviest particles known, and their theoretical co-existence with the photon is allowed by the, as yet undiscovered, Higgs boson, whose interactions with $W$s and $Z$s makes them massive. The Higgs itself has problems maintaining its respectability because, although it is an essential component of the dramatically successful Glashow-Weinberg-Salam electroweak theory, the theory places almost no constraint on its mass.

Fortunately for experimental HEP, it seems that enlightenment will be accessible. Most theorists firmly believe that both the Higgs, (perhaps in a more complicated form than one $H^0$ particle), and some new particles casting light on fermion masses, must be accessible to the current (LEP/SLC) or the next (LHC/SSC) generation of experimental facilities.

1.2 Making Measurements in the Quantum World

The fundamental interactions are not deterministic. For example, if an electron and a positron collide, we have no way of knowing in advance which of the processes shown in Fig. 2 will take place. Even if the physics of $e^+e^-$ collisions were totally understood, and we knew the initial conditions perfectly, we could not predict any more than the probability of producing each possible final state.

Since we believe that the perfect physical truths we seek would only allow us to calculate probabilities, we can reverse the argument and say that only by careful measurement of probabilities can we gain an understanding of the perfect physical truths. To measure probabilities accurately we must record and analyse many collisions. The high energy electron-positron and proton-proton collisions, which will be our windows on to new physics in the next decades, are themselves extremely complex events, and analysing them in large numbers will require massive data handling and computing resources.

There are further problems which complicate our measurements of the probabilities resulting from fundamental physics, and which make precise, high statistics, simulation an integral part of physics analysis.

Physics Smearing

Figure 3 shows a typical way in which physics itself makes it more difficult to measure fundamental processes. The figure shows three snapshots of an $e^+e^-$ collision in which the 'fundamental' process is the production of a $b\bar{b}$ quark-antiquark pair. Within a time during which we cannot possibly make any measurements at all, the $b$ and $\bar{b}$ quarks 'fragment' into jets of other particles. Even if we could calculate the complex physics of fragmentation with arbitrary precision, and measure all the particles produced, we could not identify jets from $b$ and $\bar{b}$ quarks on an event-by-event basis. On a statistical basis, $b$ and $\bar{b}$ jets can be distinguished from $u, d, s, c$ jets and $\tau$ decays, because, among
other features, they tend to be broader.

This example reflects a general problem: the new, high energy (short time-scale) physics we try to study will almost inevitably be smeared out by low energy processes acting on a somewhat longer time-scale. If we are lucky, we may understand these low energy processes rather well, but their effects are, of course, probabilistic, and the smearing cannot be removed on an event-by-event basis. Even worse, it is usually impossible to 'unsmear' the measurements without making rather rigid assumptions about the nature of the more fundamental process which was smeared. It is therefore not surprising that the normal approach to physics analysis is to perform several Monte-Carlo simulations of the smeared data, starting from various possible hypotheses about the underlying physics.

Detector Smearing
We have already seen that physics analysis is difficult even with perfect detectors. Detectors are far from perfect for many reasons:
1. Dead regions needed for mechanical supports, cables, the beam tube etc.

2. Dead regions due to lack of money. For example, the L3 detector has a very high resolution electromagnetic calorimeter made out of Bismuth Germanium Oxide (BGO) crystals. One third of the 12,000 crystals will not be installed for the first two years of running due to lack of money.

3. Limited Resolution. Every device has a finite resolution, and some, such as hadron calorimeters at moderate energy, are inherently imprecise (resolution $\leq 50\%/\sqrt{\text{Energy}}$).

4. Confusion. Detectors do not always measure what we want to know. For example:

   - a particle emerging from 1 metre of iron is probably a muon, but it may be a ‘punch through’ pion,
   - a combination of a charged track (measured in a wire chamber) and light in a BGO crystal may signal an electron, but could also be due to a photon and a pion,
   - it is normally impossible to disentangle (important) electrons embedded in (boring) jets,
   - etc.

Real physics detectors smear and lose information even more effectively than the physics smearing mentioned in the previous section. In most cases, the only effective physics analysis technique is to make assumptions about the underlying physics, and see if they result in simulated measurements which are consistent with the observations.

There is an important exception to the ‘no physics results without simulation’ rule. If new physics shows itself by producing mono-energetic particles or particle pairs, the first discovery can often be made purely on the basis of the observed events. Nevertheless, the first discovery of ‘something new’ must be followed by detailed studies involving simulation before the full implications can be understood.

1.3 Outline of Physics Analysis

Figure 4 shows the principal building blocks of physics analysis. The data are typically 100 kilobytes per event of highly compressed information gathered by a data acquisition system from up to $10^6$ sensitive devices. The simulation must include models of (what may be) the underlying physics, models of the physics smearing processes such as fragmentation, and a full treatment of the way particles traverse and interact in the detector, and how their passage finally produces electrical signals. In most studies it is also important to perform a detailed simulation of possible background signals.
Background can come from 'boring' physics processes, from beam-gas collisions, or even from electrical noise.

![Diagram showing data analysis process]

**Figure 4** An outline of physics analysis strategy.

The human mind is not equipped to decide whether $10^7$ real events, each 100 kilobytes long, are consistent with $3 \times 10^7$ simulated events. To overcome this frailty, the data are first 'reconstructed' reducing the 100 kilobytes to a few numbers for each event describing concisely what features of the event the detector has been able to measure. The term 'reconstruction' was first used in the days of bubble chambers. The charged particles emerging from few GeV collisions in a bubble chamber could be individually reconstructed with almost perfect efficiency. In today's high energy
detectors 'reconstruction' is often an optimistic term. Frequently the best that can be done is to measure unresolved jets or energy clusters. This is usually not a serious problem, since the unresolved particles in a jet carry mainly information about the 'boring' process of quark fragmentation.

Reconstruction reduces the data volume while attempting to preserve all significant measured information. The data volume is still enormous, so the next stage, often called 'Physics Analysis', reduces the data volume discarding information which may not be relevant to the particular effects under study. Both real and simulated data are reduced to a few simple distributions which can be overlayed to compare reality and hypothesis. Reducing the data volume in a way that preserves maximum sensitivity to new physics and maximum immunity to noise is the key to successful analysis. It requires intuition and intelligence, but also the computer tools to allow many wild ideas to be tried out quickly.

1.4 Two Physics Analysis Examples

Both the analyses described below may be ways to uncover important new physics.

**Higgs Search**  
One of the most eagerly sought reactions at LEP will be:

\[ e^+e^- \rightarrow Z^0 \rightarrow H^0 Z^0(\text{virtual}) \rightarrow jets + e^+e^- \]

The Higgs is expected to decay into jets of hadrons, so the measured final state will contain two hadron jets, an electron and a positron, all normally well separated in angle. If the Higgs is to be granted the status of 'particle' it must have a defined mass, and thus the \( e^+e^- \) pair produced together with it must always have a reconstructed mass equal to the centre-of-mass energy of LEP minus the Higgs mass.

The physics analysis technique is very simple:

1. Use 'cuts' to enhance the signal/background ratio. Some of the signal is, of course, lost.

2. Plot a histogram of the effective mass of the \( e^+e^- \) pair.

3. Simulate the signal to see how much was lost by the cuts.

4. Simulate all known backgrounds to see how much fake signal they contribute.

Although there may be a few tens of Higgs events hidden in a sample of millions of \( Z^0 \) decays, selection of events with an isolated electron and positron removes nearly all the background while preserving most of the signal. Figure 5 shows how a Higgs in the
Figure 5  How the Higgs might be discovered: the effects of Higgs masses of 20, 40 or 50 GeV.

mass range 20 to 40 GeV would stand out above the remaining background. If the mass is above 50 GeV, Higgs events would still be measured, but could not be identified.

Muon Charge Asymmetry

In addition to the ‘bump hunting’ technique, signs of new physics can also be found by a painstaking analysis of very common events. Figure 6 shows one such class of events, and their interpretation in terms of particle exchange. Interference between $\gamma$ and $Z^0$ exchange leads to an asymmetry in the angular distribution of muon pairs. If there are other $Z^0$-like particles they will distort this angular distribution, even though their mass is too high for direct discovery at LEP. Figure 7a shows what the data (points with error-bars) and simulation (histogram) might look like in the absence of new $Z^0$s, and Fig. 7b shows the small deviations from the simulation that might be produced by new high mass $Z^0$.

The effects I have fabricated in Fig. 7b are several times greater than the smallest effects we hope to detect at LEP. They are almost undetectable by casual inspection of the figure but show up clearly in a statistical analysis comparing its left and right halves. However, unless systematic losses of events due, for example, to dead regions of the detector, are understood almost perfectly, all the statistical precision is worthless and the potential discoveries may be lost. An even more terrifying spectre for an experimental physicist is the possibility of ‘discovering’ something which is not really there.
1.5 Physics Analysis and Accelerators

My examples so far, and my more detailed descriptions of techniques later in these lectures, are all drawn from $e^+e^-$ physics at the LEP accelerator. Although the principles of physics analysis at other machines are just the same, there can be large differences in data rates and experimental techniques. For completeness I now compare the LEP environment with some of the more extreme experimental environments of the past and the future.
Figure 7  Comparing the muon angular distribution for data and simulation.

Bubble Chamber Physics

Many physicists of my age worked for some years on reactions like

\[ K^- p \rightarrow XXX \]

using a hydrogen bubble chamber as both target and detector. These ‘good old days’ now seem very different from LEP physics. The key features which explain this difference are:

1. Bubble chambers have an excellent acceptance for charged particles.
2. In the ‘good old days’ there was no serious underlying theory (at least of hadronic physics).
3. As a result of 1) and 2), there was very little need for simulation.
Physics at LEP
For its first years of operation LEP and the LEP detectors will be used to study

\[ e^+e^- \rightarrow Z^0 \rightarrow X. \]

The main features of such physics are:

1. Electrons are ‘pointlike’ particles, that is they have no known substructure.

2. A few million events per year in which ‘matter is created’ will be observed. This rate is due to the combined effects of pointlike cross-sections which fall like the square of the energy, and an enhancement by a factor of over a thousand due to \( Z^0 \) production.

3. New pointlike (fundamental) particles are easily seen. The production rates of all pointlike particle-antiparticle pairs are similar once their production threshold is exceeded.

4. Precise predictions can be made for most features of the events. Deviations of less than 1% from the standard model predictions can be clear signs of new physics. This sort of physics analysis requires high statistics and great care in both data analysis and simulation.

LHC/SSC - Future Hadron Colliders
The process studied at hadron colliders can be written

\[ pp \rightarrow X. \]

It might be more correct to write this like

\[
(u + u + d + g + g + g + u + \bar{u} + d + \bar{d}) \\
+ (u + u + d + g + g + g + g + s + s + c + \bar{c} + u + \bar{u} + d + \bar{d}) \\
\rightarrow \text{XXXXX},
\]

which emphasises the complexity of protons and of their high energy collisions. Experimentation at tomorrow’s (or even today’s) hadron colliders is very different from life at LEP:

1. A few million events per \textit{second} in which matter is created will be observed.

2. Most events are too complex for precise theoretical understanding.
3. Nevertheless, millions of ‘hard’ events occur each year. In a ‘hard’ event a pointlike constituent of one proton, carrying a large fraction of its energy, hits a similarly energetic constituent of another proton. Hard events are, in principle, as valuable as pointlike scatters in $e^+e^-$ physics, but the absence of a known constituent energy, and the presence of millions of unwanted events, makes life more difficult for the experimenter.

CLIC/TLC - Future $e^+e^-$ Colliders
There are no firm proposals to build $e^+e^-$ colliders at $\sim 1\text{TeV}$, but studies are proceeding on both sides of the Atlantic. Physics at these machines would be a total reversal of the frantic LHC/SSC era. The pointlike cross-sections at very high energy and (probably) without any enhancement like that of the $Z^0$ at LEP would reduce rates to about ten thousand events per year. At such rates it would be easy to provide the experimenters with adequate computing facilities.

2. Physics Analysis and Computing for a Large HEP Experiment

Figure 8 shows the diagram of the analysis structure for the L3 experiment. If I changed the title to ‘HEP Analysis Structure’ the figure would be correct for almost all modern experiments. In particular, the emphasis on providing a database service, and an interactive graphics service, both used at all stages in the analysis, is typical of today’s experiments. The programs, such as the calibration processors, and the various components of the Monte Carlo and the reconstruction, may comprise up to 1,000,000 lines of code written specially for the experiment.

The physics analysis outlined in Fig. 8 requires many resources. The modern view is that software, data handling capacity, networks, and CPU power are all important tools for physics analysis. I would go even a little further, and say that I have written them in order of their importance, at least in so far as their need for manpower and money are concerned.

In an attempt to give some feeling for the resources required, table 1 shows some information about typical data volumes in a LEP experiment. The main message is that the data volumes are large, and making any fraction of these data available to the hundreds of physicists in a LEP collaboration necessarily requires expensive data handling and networking. Before going on to talk in detail about the latest software techniques and tools, I will describe a typical hardware and networking environment in which these tools are used.
2.1 The L3 Computing Environment

Figure 9 shows the overall structure of the L3 computing environment. The figure is now about four years old and only recently has begun to describe a reality rather than a goal. I will describe the L3 computing environment in some detail, because in most respects it is not at all specific to the L3 experiment and can be regarded as a good example of an environment for HEP data analysis.

Workstations were chosen several years ago as the best way to write and debug the hundreds of thousands of lines of code. Interactive graphics using workstations is the best available debugging tool, both for the software and for the detector itself.

Data handling capability for L3 is provided by the IBM mainframe component of LEPICS (the L3 Parallel Integrated Computing System) and by a share of CERN's
Table 1

Typical LEP data Volumes per Experiment

| Event rate ($Z^0 \rightarrow$ hadrons) | 0.2 per second |
| Event rate (junk)                       | 0.8 per second |
| Size of good events                     | 200 kilobytes  |
| Running time                            | 4,000 hours/year |
| Total 'raw' data                        | 10,000 tapes/year |
| Reconstructed Data                      | 10,000 tapes/year |
| Other data (eg. simulated)              | 20,000 tapes/year |

central facilities. Networking on the CERN site and worldwide is necessary to make the real progress on physics analysis approach the sum of individual efforts. Finally, the little boxes hanging off the bottom of 'LEPICS' reflect my statement about the relatively minor importance of CPU power. These boxes provide most of the CPU power for L3, and cost much less than the data handling equipment.

I will now describe in a little more detail the main elements of the L3 computing environment.

Graphics Workstations
I will give some examples of the importance of graphics in a later section. Here it is necessary to emphasise what is meant (in experimental HEP) by a workstation:

1. CPU equivalent to between 1 and 12 VAX 11/780 equivalents. (Tomorrow's workstations will, of course, be faster.)

2. Large, high-resolution bitmapped screen. Machines with $1280 \times 1024$ colour screens are now normal. The hardware and software of the workstation support graphics and multiple windows.

3. A good Fortran environment. This includes a Fortran compiler up to mainframe standards, the ability to run the largest HEP programs, and a good debugging system.

4. Good communications with the VAXes and IBMs used for data acquisition and data handling. An isolated workstation with the power of a CRAY-4 is not useful to an experimental high energy physicist.
LEPICS

The LEPICS configuration at the end of 1988 is shown in Fig. 10. Most of the figure shows an ‘ordinary’ IBM system with all the usual peripherals and network connections. The main role of the IBM part of LEPICS is to give jobs and physicists access to L3 data stored on tens of thousands of tapes complemented by many gigabytes of disk space. Although the 3090-180E processor is not a negligible CPU resource, it is far too small (perhaps by a factor 12) to meet L3’s CERN-site CPU needs. The first indications of how the computing power will be provided appear at the lower right corner of Fig. 10. The VICI is an interface between an IBM I/O channel and the industry standard VME bus. A string of 3081/E emulators is connected to the VME bus, and the IBM can copy data to or from the 3081/E memory at close to 3 megabytes/sec. The 3081/E is a processor built by CERN and SLAC which emulates the IBM system/370 instruction
set and uses the same data format as the IBM. The string of 3081/Es in Fig. 10 more than doubles the total CPU power of the configuration. Work is now in progress to connect more modern (and very cost-effective) processors such as the Apollo DN10000 to the IBM via VME to Channel interfaces.

![Diagram](image)

**Figure 10** The LEPICS computer system in December 1988.

This assembly of hardware and its interconnections can be the foundations of a powerful HEP computing system, if we can provide system and user software to exploit it. Fortunately nearly all the CPU cycles used by high energy physicists are employed to process ‘events’ — the real or simulated results of single collisions. Events are independent of each other, and so the 1000 events on a tape can be processed forward, backward, or in parallel. In practice, if we want the results of parallel processing (including histograms and statistical summaries) to be identical to those from a serial job, the software has to be rather well organised. However, adapting HEP software for par-
allel processing is relatively simple, and the resulting code may be even more intelligible than the original serial-only version.

Up to now, efforts to use parallel computing for HEP data analysis have stopped at the point where production programs had been made to run on a dedicated 'farm' of processors. The 'production team' would take control of the farm for a night or a week, and perform a long series of reconstruction or simulation jobs. Unfortunately, organised production is only part of the load, and in a collaboration such as L3, the other 395 physicists would also like to be able to run some of their jobs on cost-effective processors. It is necessary to take the large step from a 'processor farm' to a parallel computing centre. This step requires that the attached processors become managed dynamically in much the same way as the host mainframe CPU cycles are managed by its operating system. Figure 11 outlines how L3 will do this within the framework of the VM/CMS operating system.

![Diagram](image)

**Figure 11**  L3 plans for attached processor resource management under IBM VM/CMS.
The virtual machine (process) BMON in Fig. 11 is part of the SLAC Batch Monitor system used widely within HEP. BMON manages the queue of batch jobs, and when appropriate, starts a 'batch worker' (JOB VM) virtual machine and gives it a job to execute. The attached processor resources are managed by the additional virtual machine EMUMAN, which releases jobs in the BMON queue when AP resources are available to execute them. Dynamic management means that the APs allocated to a job may vary, and EMUMAN accomplishes this by telling low-level code in the JOB VMs how to map a fixed number of virtual processors on to a varying number of real allocated devices. The Interface VMs provide a stable software interface to heterogeneous attached hardware. EMUMAN sets up communication paths from the JOB VMs to the attached processors, but does not itself handle this main data flow.

LEP3NET

The L3 experiment involves collaborators from the USA, Western Europe, Eastern Europe, the USSR, China and India. The network connections which now link most of these collaborators were in the main set up through the efforts of L3 members. Figure 12 shows the current configuration of 'LEP3NET' together with the ESNET-X.25 network which was recently created using LEP3NET as a model. (ESNET-X.25 links SLAC, LBL, Fermilab, BNL and MIT and has a satellite link from Fermilab to CERN.) Most LEP3NET lines now run at speeds between 9.6 kilobits/sec. and 64 kilobits/sec. supporting remote log-on to computers, and the exchange of software. Later in these lectures I will describe how much higher speed networking could revolutionise the way in which universities are involved in physics analysis. L3 members are currently expending considerable (mainly political) efforts to achieve megabits/second as soon as possible.

3. Examples of Software Foundations

I will now describe some of the modern tools which support physics analysis. I will concentrate on the approach used by L3 simply because I can explain this approach better than the equally valid (and usually similar) approaches used by other experiments. I will give two complementary examples of data management, the ZEBRA system and the DBL3 database system. Then I will describe the GEANT3 system which takes most of the hard work out of describing the complex geometry of modern detectors. Finally I will give a sales talk about the importance of interactive graphics, and describe the PAW system which provides the foundations for interactive physics analysis on workstations.
3.1 ZEBRA: Data-Structure Management and I/O

High energy physicists continue to program in the ugly language called FORTRAN. Computer scientists could probably list 100 reasons why this is a stupid idea, but from the strictly practical viewpoint, FORTRAN brings two big problems:

1. It has no data structures, only fixed-dimension arrays.
   HEP data usually has a very complex structure. For example Fig. 13 shows a simplified version of part of the data structure used to describe L3 events.

2. It has no efficient machine independent I/O.
   L3 data will be acquired by a VAX, first processed on an IBM, and then looked at by physicists sitting at Apollo workstations. All these machines have different data formats.

These problems are augmented by one constraint:

3. Physicists want direct read/write access to data in memory; returning the wanted data as a subroutine argument is (supposed to be) far too inefficient. In other words, physicists insist on being able to overwrite any part of their data by mistake.

Neither of these problems is new, and many packages providing solutions have appeared in the last 15 years. I will describe the ZEBRA [1] system but many of the ideas are common to its predecessors/competitors such as HYDRA, ZBOOK, BOS etc.

ZEBRA manages chains and trees of data objects in memory, and allows the I/O characteristics of each object to be recorded, so that any part of a data structure can be
translated into a compact, machine independent, format and sent to another computer. Figure 14a shows the logical structure of a chain, which is normally used to relate objects which are all of the same type, such as the tracks emerging from a vertex. Figure 14b shows how chains and trees can be used to build more general structures including hierarchical relationships. To allow efficient navigation around complex structures, 'links' (addresses) must be stored in data objects pointing to other related objects. Figure 14c shows the links automatically created by ZEBRA to help both the user and the ZEBRA system itself.

ZEBRA manages data by creating 'banks' in large Fortran Common blocks known as ZEBRA Stores. The banks include space into which the user can put data, and also space in which system and user links are maintained. For example, the 'reference links', shown as the upward-right pointing arrows in Fig. 13, are maintained with the structural links within the bank. Figure 15 shows the format of a ZEBRA bank. In addition to the features already described, note the I/O control byte and the I/O descriptor words which support automatic data translation for machine independent I/O.

I will not replicate the entire ZEBRA manual, and will conclude this section on ZEBRA by giving a few commented examples of the code which you might write when using ZEBRA. Of course, the point I am trying to make is that it is 'not really all that unpleasant to use', but I will let you judge for yourselves.
Figure 14   Examples of ZEBRA data structures.

Initialise a ZEBRA Store

COMMON/name/IFENCE(10),LINK,ISTORE(100000)
DIMESION LQ(999),IQ(999),Q(999)
EQUIVALENCE (LINK,LQ(1)), (LQ(9),IQ(1),Q(1))
CALL MZEBRA(0)
CALL MZSTOR(IXSTOR,'/name/',', ',IFENCE,LINK,
The format of a ZEBRA bank.

\[ \text{LQ(L - NL - NIO -1)} \]
\[ \text{LQ(L - NL - NIO)} \]
\[ \text{...} \]
\[ \text{LQ(L - NL - 1)} \]
\[ \text{LQ(L - NL)} \]
\[ \text{...} \]
\[ \text{LQ(L - NS - 1)} \]
\[ \text{LQ(L - NS)} \]
\[ \text{...} \]
\[ \text{LQ(L - 1)} \]
\[ \text{LQ(L)} \Rightarrow \text{next-link} \]
\[ \text{LQ(L + 1)} \]
\[ \text{LQ(L + 2)} \]
\[ \text{IQ(L - 5)} \]
\[ \text{IQ(L - 4)} \]
\[ \text{IQ(L - 3)} \]
\[ \text{IQ(L - 2)} \]
\[ \text{IQ(L - 1)} \]
\[ \text{IQ(L)} \Rightarrow \text{status word} \]
\[ \text{IQ(L + 1)} \]
\[ \text{...} \]
\[ \text{IQ(L + ND)} \]

*ISTORE(1), ISTORE(1), ISTORE(20000),
*ISTORE(100000)

The nice little EQUIVALENCE trick allows integer and floating point data to co-exist in the data structures and ensures that LQ(bank_address - 5) is the address of the 5th structural link whereas IQ(bank_address + 5) is the address of the 5th data word. I gave a full set of arguments for MZSTOR to make my example realistic, but I will not burden the reader with an explanation of each one.

Create a Link Area

COMMON/GLINKS/LGEVNT,LGVRTX,LGTRAK,LGWIRE
CALL MZLINK(IXSTOR,'/GLINKS/','.LGEVNT,LGVRTX,LGWIRE)

When using complex data structures it is convenient to store the address of frequently used banks, rather than to re-navigate through the structure whenever the
address is required. However, ZEBRA may 'garbage collect' at almost any time. Any addresses you might want to maintain should be declared to ZEBRA, so that it can update the addresses for you if it moves any of your banks around. This example shows the declaration of one 'structural link', LGEVNT, followed by three 'reference links'.

Create a Bank

    CALL MZBOOK(IXSTOR, LGVRTX, LGEVNT, -1, 'VRTX', 6, 3, 100, 2, 0)

This shows the creation of a bank 'VRTX', returning its address LGVRTX. The new bank 'hangs off' link number 1 in the existing bank whose address is LGEVNT. The new bank has 6 user links, of which 3 are structural, implying that up to 3 banks may 'hang off' this new bank. The new bank has 100 data words, all of which are integers and all of which are cleared to zero when the bank is created.

Insert Data

    Q(LGVRTX+1) = X
    Q(LGVRTX+2) = Y
    Q(LGVRTX+3) = Z

Notice that this example involves no subroutine call. The example should be compared with the corresponding 'pure Fortran' (VRTX(1)=X etc.). Although I promised that you could judge for yourselves, I am sure that you can see that, in this example, ZEBRA is only marginally more CPU consumptive, and only marginally more difficult to code and understand, than 'pure Fortran'.

Use Data

    PRINT *, 'Vertex position', (Q(LGVRTX+I), I=1,3)

Once again, access to the data requires no subroutine call, and it is easy to make the code as readable (or unreadable) as normal Fortran.

Drop a Bank

    CALL MZDROP(IXSTOR, LGVRTX, ' ')

Unlike Fortran arrays, ZEBRA banks, or the data structures supported by banks, can be dropped when no longer needed. Dropped banks are marked, but the space is only recovered by compressing the data structure if a 'garbage collection' is provoked by some operation which is unable to find enough immediately available free space.

Output a Data-Structure

    CALL FZOUT(1, IXSTOR, LGVRTX, 1, ' ', 2, 10, IHEADR)
The data structure supported by the bank whose address is LGVRTX within store IXSTOR is written out to logical unit 1. The record is flagged as the start of a new event and is preceded by a 10 word integer header read from the array IHEADR. An initialization call to FZFILE will have already chosen machine independent binary, machine independent ASCII, or machine dependent ‘native’ output mode.

3.2 DBL3: HEP Database Management

I am not giving a lecture course on database theory so, in this section, I will allow myself to take an HEP (or even L3) viewpoint on database systems. From this viewpoint, I define a database system to be:

“A system for the storage and retrieval of data in which data are identified by the attributes which are most relevant for the user or application”

Here are two typical ‘queries’ to database systems conforming to my definition. The first example is formulated in the SQL query language:

```
SELECT NAME, PHONE FROM L3_COLLABORATORS
WHERE RESPONSIBILITY LIKE ‘%TEC%GAS_SYSTEM%’
ORDER BY DATE_LAST_CALLED IN THE_MIDDLE_OF_THE_NIGHT
```

The second example is typical of the functions required of a database for HEP calibration information:

```
CALL GET_VALID_CALIBRATION(‘BGO/BARREL’, ‘03:14:56 19-MAR-1990’)
```

When retrieving data from a database system it is not necessary to specify tape numbers, or device addresses, and in this respect, database systems are similar to modern file systems. In addition, a database system is normally expected to be efficient even when the size of the objects being stored and retrieved is very small.

There is an enormous range of systems which can be described as ‘database management systems’. At the most sophisticated end of the range are systems like ORACLE and SQL/DS which are widely marketed. Although no attempt has been made to optimise these systems for HEP, the main restriction on their use is cost rather than the quality and flexibility of the software. At the lower end of the range are systems like the ZEBRA RZ package, which computer scientists would probably call an ‘access method’ rather than a ‘database management system’.

Why should HEP be so interested in database systems? Let me answer this indirectly by pointing to Fig. 16 which shows the structure of the part of the L3 database refering to the alignment and calibration of the L3 muon chambers. The muon chamber system is just one of the high precision detector systems in L3. The intrinsic precision of these systems can only be realised by frequent calibrations of all their components.
The calibration information shown in Fig. 16 is not static; some information has a life of months, but many of the database objects are refreshed every few minutes. Unless these data are handled automatically it is quite impossible to exploit the detector we have built.

Figure 16  Structure of the L3 database for muon chamber alignment and calibration.

The L3 Collaboration recognised that a database system was required over four years ago. As a first step, the performance and features of existing products were assessed by using them to build a ‘model calibration database’ for L3. Table 2 shows the numerical results of tests using different underlying systems. Table 2 shows that the HEP-written packages KAPACK and ZEBRA-RZ were faster than commercial database systems. However, the HEP systems also lacked many of the useful features of the
commercial systems, and L3's choice of ZEBRA-RZ was finally based almost entirely on the near impossibility of persuading over 40 institutes to buy an expensive commercial system.

Table 2: Results of the L3 Model Database Tests

<table>
<thead>
<tr>
<th></th>
<th>ORACLE IBM 3090</th>
<th>ORACLE VAX 8600</th>
<th>SQL/DS IBM 3081 (night)</th>
<th>KAPACK IBM 3081 (day)</th>
<th>RZ IBM 3090</th>
<th>RZ VAX 8650</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Size</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Megabytes</td>
<td>120</td>
<td>6</td>
<td>120</td>
<td>420</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>'Rows'</td>
<td>7000</td>
<td>350</td>
<td>7000</td>
<td>20000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td><strong>Insert Performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elapsed Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>small row, msec</td>
<td>100-300</td>
<td>200-1000</td>
<td>1</td>
<td>&lt;10</td>
<td>&lt;10</td>
<td>&lt;10</td>
</tr>
<tr>
<td>large row, msec/kbyte</td>
<td>6-17</td>
<td>13-77</td>
<td>330</td>
<td>26</td>
<td>6.5</td>
<td>3-7</td>
</tr>
<tr>
<td>CPU Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>small row, msec</td>
<td>↓</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>large row, msec/kbyte</td>
<td>0.7</td>
<td>3.5</td>
<td>77</td>
<td>1</td>
<td>0.65</td>
<td>0.4-0.7</td>
</tr>
<tr>
<td><strong>Read Performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elapsed Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>small row, msec</td>
<td>50-80</td>
<td>100-150</td>
<td>55</td>
<td>20</td>
<td>59</td>
<td>78</td>
</tr>
<tr>
<td>large row, msec/kbyte</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>CPU Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>small row, msec</td>
<td>↓</td>
<td>25</td>
<td>↓</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>large row, msec/kbyte</td>
<td>1</td>
<td>2.7</td>
<td>0.7</td>
<td>0.13</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td><strong>Read model L3 recon-</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>struction environment:</td>
<td>6 secs</td>
<td>10 secs</td>
<td>6 secs</td>
<td>3 secs</td>
<td>6 secs</td>
<td>5 secs</td>
</tr>
<tr>
<td>~40 rows,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>~500 kbytes.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The RZ system supports a UNIX-like hierarchy of 'directories'. Directories can contain other directories and 'objects' where an object has a 'keys' and 'data'. The keys are used to identify objects, and are usually just a few words per object. The data part of an object can be retrieved once its keys are known. The DBL3 system, constructed on top of RZ, offers the following principal features:

1. Data objects are identified by pathname (e.g. //L3/BGO/BARREL/CALIB), and
by validity time. On insertion the range of validity times must be given. On retrieval, the system returns the most recently inserted object which is valid for the requested time.

2. Data objects can be compressed, either by applying compression algorithms to individual objects, or by comparing a new object with an earlier one and recording only the differences.

3. DBL3 manages a memory resident 'cache' of recently retrieved or entered data as shown in Fig. 17. The cache management is steered by two user callable subroutines:

   \[\text{CALL DBUSE('''/DBL3/MUCH/MCALB/ALIG/BEAC',}
   \text{Time-date, Other-Keys, Address, ....)}\]

   This means, 'I intend to use this database object. Please load it if it is not in memory (or if the copy in memory is no longer valid). Give me its address'.

   \[\text{CALL DFREE('''/DBL3/MUCH/MCALB/ALIG/BEAC')}\]

   This means, 'I won't be using this object for a while. You can overwrite it in memory if you need the space'.

4. Normally, if some DBL3 data are superseded, or even found to be wrong, they are not deleted. Figure 18 shows how old and new versions of calibration data can co-exist in the database. By default, the more recently inserted data will be retrieved by DBUSE, but it remains possible to make the database behave exactly as it would have done at some earlier time.

   In commercial database systems, such as airline reservation systems, a correct and high performance handling of concurrent and conflicting write requests is of great importance. In an HEP experiment, if two conflicting groups are given responsibility for a calibration task, then the problems cannot be solved by database technology alone. In other words, conflicting write access to a calibration database is not a problem. This opens up the possibility of creating many simultaneously valid copies of the database which keep each other up-to-date by sending all changes over network links. The L3 collaboration will use a pair of communicating 'server processes' to ensure the (almost) simultaneous validity of the database on the data-acquisition VAX cluster, and on the off-line LEPICS system. Generalisation of this idea, to ensure coherence of L3 databases worldwide, is an intriguing and challenging prospect.
3.3 GEANT3: Geometry and Tracking for Complex Detectors

High statistics physics using a complex detector needs a Monte Carlo simulation with several major components:

1. Generation of the particles emerging from the collision assuming some combination of known physics and more speculative hypotheses.

2. Precise representation of the detector geometry. Typical detectors have over 1,000,000 components with which particles interact.

3. Precise simulation of all secondary interactions and ‘showers’.

4. Precise simulation of the response of detectors (gas, plastic, glass, crystals, etc.) to the passage of the simulated particles.

The GEANT3 system [2] provides the core of a simulation program, leaving only the particle generation and description of the detector response to the user.

An economical but powerful description of detector geometry is supported by a hierarchy of ‘volumes’. A volume can be split up into sub-volumes by positioning other volumes inside it, or by slicing it up in a repetitive way. The hierarchy is represented by a (ZEBRA) tree data structure. The symmetry of HEP detectors makes it unnecessary that the tree end in 1,000,000 ‘leaves’ describing individual components, since most components are identical, apart from a spatial transformation, to many others.

For example, the L3 muon chamber system which consist of 176 chambers and tens of thousands of sensitive wires is described by some 60 calls to GEANT3 subroutines.

Graphics are vital in checking the correctness of this description of detector geometry. GEANT3 can produce a three dimensional display of the geometry that has been created and examples of these displays will be given in the next section.

Once GEANT3 ‘knows’ all about the detector geometry, it can track particles through the detector and record their paths and energy losses in the sensitive volumes. However, energy loss is only one possible fate of a particle travelling through material. Even in empty space, many particles decay spontaneously into others, and in solid or liquid matter, most particles interact after a short distance. GEANT3 includes a detailed simulation of the ‘electromagnetic showers’ which are the usual fate of electrons or photons, and a painstaking simulation of the much more complex and varied processes involved in hadronic showers. This latter simulation originated in the GHEISHA simulation system [3] but is now also available as the GEANH package within GEANT3.
3.4 Interactive Graphics

The outline of GEANT3 provides a good introduction to my sales pitch for interactive graphics. GEANT3 is itself a package of some 60,000 lines of code, not including the other CERN library routines it invokes. To make GEANT3 into a complete simulation for a detector like L3, a further 50,000 lines of code are required, mainly concerned with the details of detector response. Programs containing more than 100,000 lines of Fortran code are common in HEP simulation, reconstruction and physics analysis. There is no way that a simple inspection of printed output from such programs can convince anybody that reasonable things are being done by all 100,000 lines of code.

Electronic wizards cannot build equipment without oscilloscopes. Their software counterparts would be foolish to build even more complex programs without the equivalent instrumentation. I will illustrate the importance of graphics produced interactively by means of a few examples – many of them drawn from the L3 simulation program based on GEANT3.

![Figure 19](image)

*Figure 19* The upper two levels of the GEANT3 data structure representing the L3 geometry.

It is natural to think of graphical displays of physical objects such as real detectors. However, for the software builder, a graphical display of an abstract object, such as a data structure can be at least as useful. Figures 19 and 20 show the result of giving the interactive 'DTREE' command to GEANT3. This command displays the tree data-structure which has been built up to represent the L3 detector. To facilitate checking the logic, small pictures of each geometrical object described by the data structure are added to the picture. Figure 19 shows just the top levels of the representation of the L3 detector, whereas Fig. 20 shows the complete structure representing the muon chambers. It should be stressed that, having written the 60 lines of code that describe the muon...
chambers, no further information is needed to generate the graphical description of the chambers or of the tree data structure.

![Diagram of GEANT3 data structure for L3 muon chambers]

**Figure 20** The GEANT3 data structure representing the L3 muon chambers.

Having satisfied ourselves that the logical relationships between the components of our detector are correct, it may also be a good idea to check that the right dimensions were coded. The 'DSPEC' command draws a 'specification' sheet for any component of the geometrical structure, such as the L3 inner muon chamber MBI shown in Fig. 21 or the more complicated hadron calorimeter barrel structure shown in Fig. 22.

As a real example of debugging using graphics look at Fig. 23. This figure shows a 'zoom' in on one corner of one of the 144 trapezoidal modules which make up the L3 hadron calorimeter. The individual brass tubes and uranium or stainless steel plates
Figure 21  The GEANT3 ‘Specification Sheet’ for an L3 inner muon chamber.

Figure 22  The GEANT3 ‘Specification Sheet’ for the L3 hadron calorimeter barrel.
are clearly visible. When first coded, the module description included an error which made one of the stainless steel plates extend just outside the module’s case. The error was clearly visible on such a graphical display.

Figure 23  A corner of an L3 hadron calorimeter module.

The examples shown so far were mainly two dimensional projections. When probing the performance of the hardware or reconstruction software of a three dimensional detector, it is natural to try to use three dimensional displays. To get three dimensional information into the brain of a physicist it is usually sufficient to display an image on a high resolution screen (e.g. 1280 x 1024) and allow the physicist to rotate or wobble the three dimensional object. High performance workstations from several manufacturers have the hardware capabilities needed for these displays; acquiring the workstations only costs money. Much more challenging is the production of three dimensional displays which convey useful information. As an example of the problems compare Figs. 24a and 24b. Figure 24a shows GEANT3’s two dimensional view of an L3 muon chamber ‘octant’, one of 16 seven tonne structures each supporting 5 muon chambers. The display looks simple enough. Of course, GEANT3 knows all about the three dimensional structure of this octant, so why not rotate it and really understand the structure? Figure 24b shows a rotated view, but far from being clearer, an excess of (already existing) detailed information has made the picture unintelligible.
Figure 24 Views of an L3 muon chamber 'octant': a) end-on, b) rotated.

As a general principle, three dimensional graphics require a completely fresh approach; attempts to generalise the displays that work well in two dimensions are rarely satisfactory. Several experiments have found three dimensional displays a vital component of their physics analysis, and experience with prototype three dimensional displays for L3 has already shown their value. Unfortunately, reproducing 3D colour as a black and white drawing on paper is no way to convince you of this. I will limit myself to the single example of Fig. 25 which shows a composite display of an event in the L3 detector. The four quadrants of the display include three 3D views and one 2D presentation of muon chamber residuals. reconstructed

3.5 PAW: The Physics Analysis Workstation

In spite of its name, PAW [4] is purely a software project. The main aim of PAW is to provide a physics analysis environment which is portable, but can make effective use of the facilities offered by modern workstations. PAW can also run in a mainframe + dumb terminal environment, but the prospective users of such systems should be aware that almost all PAW development starts on workstations.

The components of the PAW system are shown in Fig. 26. The observant will notice that ZEBRA has already been mentioned in these lectures, and the old will notice that HBOOK and HPLOT have been around (in earlier versions) for many years. Thus PAW
Figure 25  A composite display of an event in the L3 detector.
Three of the four display windows show 3-dimensional views.

attempts to collect existing tools into a coherent environment, and to add to them extra utilities to make the environment complete.

The components in Fig. 26 merit individual descriptions:

- **KUIP**: The Kernel User Interface Package [5] is a new package providing the user interface to PAW. Like all the components of the PAW environment, it is designed to be used alone if appropriate. Thus interactive control of the L3 reconstruction program is now provided by KUIP, but we do not consider that the reconstruction is under the control of PAW. The reason why KUIP can be used outside PAW is that it allows complete flexibility in command definitions. Commands are organised in a hierarchical structure. For example

  HISTOGRAM/FIT/GAUSS
  performs a fit of a Gaussian curve to an HBOOK histogram whereas
Figure 26 The components of the PAW system.

L3/REL3/MUCH/MUTK displays numerical information about tracks in the L3 muon chambers found in the REL3 reconstruction program.

Commands are created by typing a ‘Command Definition File’ in a relatively intelligible format. The CDF is then turned into ‘gobbledygook’ Fortran by the KUIP compiler. Figures 27a and 27b show an extract from a CDF file and the corresponding output of the KUIP compiler. Frequently used KUIP commands can be stored in ‘macro files’, and it is even possible to make a macro file from the transcript of a KUIP session in which you have achieved something particularly clever.

- **HBOOK**: The HBOOK histogram package [6] is over ten years old. The original version supported booking, filling and printing (on a line-printer) of histograms and scatter plots. All HBOOK functions are invoked by calls to Fortran subroutines. The latest version of HBOOK (V4) supports a hierarchical directory of histograms and has many additional features of which the most important is
(a) KUIP Command Definition File

>N HISDEF
>Menu HISTOGRAM
>Guidance
Manipulation of histograms, ntuples.
Interface to the HBOOK package
>Command LIST
>Parameters
+
CHOPT ’Options’ C D=’ ’ R=’ ’ I’
>Guidance
List the histograms in the current directory (memory or disk).
Histograms are all HBOOK objects including ntuples.
If CHOPT=’I’ a verbose format is used (batch: HINDEX).
>Action PAHIST

(b) Output of the KUIP Compiler

SUBROUTINE HISDEF
PARAMETER (MGUIDL=99)
CHARACTER*80 GUID
COMMON /KCGUID/ GUID(MGUIDL)
EXTERNAL PAHIST
CALL KUNWG( 19)
CALL KUCMD( ’ ’,’HISTOGRAM’,’C’)
GUID( 1)=’Manipulation of histograms, ntuples.’
GUID( 2)=’Interface to the HBOOK package’
CALL KUGUID(’HISTOGRAM’,GUID, 2,’S’)
CALL KUCMD(’HISTOGRAM’,’ ’,’SW’)
CALL KUNWG( 46)
CALL KUCMD(’ ’,’LIST’,’C’)
CALL KUNDPV( 1, 1, 1, 1, 1)
CALL KUPAR(’LIST’,’CHOPT’,’Options’,’CO’,’S’)
CALL KUFVAL(’LIST’,’CHOPT’,0.0, ’ ’,’D’)
CALL KUFVAL(’LIST’,’CHOPT’,0.0, ’I’,’V’)
GUID( 1)=’List the histograms in the current direct’/
+’ory (memory or disk).’
GUID( 2)=’Histograms are all HBOOK objects includi’/
+’ng ntuples.’
GUID( 3)=’If CHOPT=’I’ a verbose format is used ’/
+’(batch: HINDEX).’
CALL KUGUID(’LIST’,GUID, 3,’S’)
CALL KUACT(’LIST’,PAHIST)

Figure 27 Source and compiled versions of a KUIP command definition file.

probably the support for ‘N-tuples’. An N-tuple can be regarded as a simple fixed-
format event-file, in which N words are used to store the key quantities describing
an event. Of course, this representation is only appropriate at the final stages of
physics analysis, and it is very important that a physicist can change his mind about what should be put into the N-tuples several times a day. Having created an N-tuple file, HBOOK and other components of the PAW system provide tools for making a wide variety of displays based on the N-tuple components.

- **HPlot**: The HPlot [7] package, allowing the graphical (as opposed to line-printer) display of HBOOK histograms, also has its origins in the mists of time. The latest version (V5) is fully integrated with the other components of the PAW environment.

- **HIGZ**: The High Level Interface to Graphics and Zebra [8] is an attempt to present users with a stable interface to the graphics jungle. In spite of (or because of) the many graphics standards (GKS-2D, GKS-3D, PHIGS, ....) the way in which graphical information can be stored and displayed still varies widely. HIGZ offers a GKS-like interface to higher level programs, together with some additional 'macroprimitives' to simplify the drawing of commonly needed objects such as axes and boxes. HIGZ can also store pictures in an RZ-based picture database. The graphics functions are then implemented on top of various graphics packages which may either be standards, or efficient manufacturer-specific products.

The original HIGZ within the PAW system only handled two dimensional graphics. Recent extensions to support three-dimensions have made it appropriate to use HIGZ as a general tool to ensure the portability of all programs using graphics.

- **COMIS**: The Compilation and Interpretation System [9] is a Fortran 77 interpreter which can be invoked from within PAW. Over the last 20 or 30 years, several attempts have been made to create 'standard' physics analysis software where a few simple instructions on data cards would produce the desired histograms. These systems usually became more and more complex before being abandoned in favour of the flexibility of making cuts and calculating quantities in pure Fortran. The problem with pure Fortran is that a simple change to the code requires re-Compilation of the code, and re-linking the program which can take a total of minutes or hours. In the PAW environment a command like

```
NTUPLE/PLT 30.X
```

will draw a histogram of variable 'X' in N-tuple file number 30.

```
NTUPLE/PLT 30.X SELECT.FOR
```

will produce the same plot but subject also to the selections made by subroutine SELECT.FOR which is interpreted in real time by COMIS. If SELECT.FOR is not doing the right thing, it can be edited from within PAW and the command can be re-issued.

- **ZEBRA**: I will not start to explain ZEBRA again. The only point I want to make here is that ZEBRA provides the data structure and file management for KUIP, and for the latest versions of HBOOK and HIGZ.
Finally I should show you an example of PAW running on a workstation. Figure 28 shows an Apollo workstation screen during a PAW session in which the performance of the L3 muon reconstruction was under study.

![Typical PAW session on an Apollo workstation.](image)

Figure 28  Typical PAW session on an Apollo workstation.

4. HEP Data Analysis in the 1990s

Up to this point I have examined the more forward-looking techniques currently employed in HEP data analysis. Since 1990 is in the far future, the title of this chapter entitles me to explore more idealistic ideas without the constraint of having it all working by next summer.

4.1 HEP Sociology

Figure 29 summarises the problem which we must try to solve. Experimental HEP is an intensely collaborative science. If collaboration also means that everybody has to move to the experimental site, then all intellectual exchanges with university colleagues and university students will be lost. Most people are sure that this loss would be fatal for HEP on both scientific and political grounds.
In the ‘good old days’ of bubble chamber physics, the problem was less severe. Analysis of bubble chamber data took years wherever it was done, and universities were not at a disadvantage. For example, I wrote my Ph.D. thesis on a bubble chamber experiment which was analysed, in its entirety, at Cambridge University by myself and one other graduate student.

In modern high statistics experiments, data can and should be analysed within hours of data-taking to ensure that all the money and effort is not being wasted on acquiring un-reconstructable junk. This immediate reconstruction, and the subsequent first look at the physics, can be done most effectively by people at the site where the data are acquired. Of course this is ‘unfair’ to physicists at a remote institute connected by a dial-up modem.

If nothing could be done about this problem, I would not bother to talk about it. However, in the last few years, the technology of network infrastructure has reached
the point at which it is technically feasible to give remote physicists the same access to an experiment's data as that enjoyed by their 'fortunate' colleagues who have to take shifts on the experiment.

4.2 The Hierarchy of Data Sets

Before looking at the possibilities of modern networking, we should understand the sort of data to which physicists need access. Most physics studies, or even technical software developments, do not start from the raw tapes written by data acquisition systems, but from tapes or data sets which are the output of a reconstruction program. The traditional name for these data sets, even if they are huge, is 'Data Summary Tapes' or 'DSTs'.

The names of DSTs are by no means standardised, but most large experiments have DST categories similar to those outlined below. In each case the data volumes I give are those expected after a few years of running a single LEP experiment.

Master DST
This is the output of the reconstruction program. The events will contain not only reconstructed 4-vectors (energy plus three components of momentum) for particles, clusters or jets, but also a more-or-less complete history of the reconstruction process. Many experiments add the original raw data for good measure so that the 'summary tape' is longer than the input tape. A Master DST may amount to 20,000 tapes. It can take minutes or hours to access a single event, and months to process all the events systematically. Consequently a Master DST may wait for months or years (or even for ever) before a new version is produced using better calibrations and reconstruction algorithms.

Clearly a Master DST is an unwieldy object. The data can be made more accessible either by selecting a subset of the events, or by selecting a subset of the data for each event.

Subset DSTs
Different physics studies need different event samples, and some physics studies are much more urgent (and perhaps more interesting) than others. There is no point in repeatedly reading through 10,000,000 events if the only events you look at are the 1,000 with three or more muons. Thus, in addition to writing a Master DST, most experiments write subsets of the events to separate files. Typical subsets for a LEP experiment would be:

- Events with one or more muons,
- Events with two or more muons,
- Events with two or more electrons,
- Events with missing energy transverse to the beam,
- Events with isolated photons,
- Events with many jets or broad jets.

The size a Subset DST could range from 1 to 1000 tapes. The smaller DSTs would be kept on disk - the larger DSTs perhaps in an automatic tape cartridge handler. The smaller, or more important, Subset DSTs may have quite a short life. For example as soon as a better calibration of the electromagnetic detector was available, the electron/photon Subset DSTs might be re-processed.

**Mini DSTs**

In my arbitrary nomenclature, a Mini DST contains less information for each event than the Master DST. Typically the raw data and the detailed reconstruction history are discarded, leaving only a summary of what was found. Normally it is impossible to correct a calibration error, or re-run a reconstruction algorithm using the data on the Mini DST. The Mini DST events may be around 5% of the size of those on the Master DST, and a Mini Subset DST will usually be something that you could carry in your luggage. Nevertheless, the Mini DST for the whole event sample could still be 1000 tapes, and could only be read infrequently.

**Micro DST**

Micro DSTs are likely to contain the N-tuples which can be manipulated by PAW. If a complete LEP data sample ($10^7$ events per experiment) were to be compressed to the size of one tape cartridge (200 megabytes), each event could be represented by only 5 words. Micro DSTs at this level of compression become very specialised. A particular selection of variables might be adequate for one afternoon's study by one or two physicists, after which a new selection would have to be made from the Mini DST. Part of the challenge of high statistics physics analysis is to achieve the best balance between a large DST (containing everything that might be needed) and a compact DST (which can be read in seconds rather than hours).

### 4.3 Network Access to Data

It would be extremely costly and manpower intensive to ship all Master DSTs, Subset DSTs, Mini DSTs and Micro DSTs by freight to 40 remote institutes. Furthermore, except in the case of the Master DST, the tapes would often be out of date by the time they had cleared customs. Thus if a remote physicist is to be at no disadvantage, he must have network access to much of these data. To give an idea of what the network requirements are, Fig. 30 shows how long a physicist sitting at a remote workstation
would have to wait for a 200 kilobyte event if he had access to various levels of network performance. The hypothetical network links range from the 9.6 kilobits/sec. 'phone line', which many physicists would, even now, consider a luxury, to the 100 (or more) megabits/sec which can easily be carried by a 10 micron thick glass fibre. With 2 megabits/sec to himself, or with a share of a higher speed network, the remote physicist ceases to care where the data are located.

<table>
<thead>
<tr>
<th>Line Speed</th>
<th>Time to Transmit one 200 kbyte event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dedicated</td>
</tr>
<tr>
<td>9.6 k bits/sec.</td>
<td>3 mins.</td>
</tr>
<tr>
<td>64 k bits/sec.</td>
<td>30 secs.</td>
</tr>
<tr>
<td>2 M bits/sec.</td>
<td>1 sec.</td>
</tr>
<tr>
<td>10 M bits/sec.</td>
<td>200 msec.</td>
</tr>
<tr>
<td>100 M bits/sec.</td>
<td>20 msec.</td>
</tr>
</tbody>
</table>

Remote Access becomes more appropriate than file-transfer or air-freight.

Figure 30 Access times for a 200 kilobyte event at different levels of network performance.

These technically possible network speeds bring other benefits. In a collaboration linked by multi-megabit lines it becomes possible to use computing and data handling resources wherever they are located without compromising the speed and flexibility of the physics analysis.
4.4 Optimising Remote Access to Data

Figure 31 shows a simple approach to remote data access. A physicist runs a program on his workstation using data obtained from a remote data handling system which is probably located at the experimental site. If the task involves several minutes of computing for each event, then this approach is optimal. On the contrary, if the 'event selection' rejects 99% of the events after looking at one bit in the event header, then this approach is very inefficient.

![Diagram of Workstation and IBM Mainframe](image)

*Figure 31* The simple approach to remote data access.

A remote access strategy which optimises performance given finite resources must also include the more difficult approaches shown in Figs. 32a and 32b. The physicist's application must be split into local and remote parts so that the communication load is minimised and the performance is consequently maximised. This sort of splitting is best thought of from the very beginning, so that physics analysis programs are constructed as a set of communicating modules and any module can run at either end of the network.

Figures 31 and 32 are still over-simplifications. In reality, there may be several data handling centres accessible over the network, and overall performance will be optimised if copies of frequently accessed data migrate automatically to a center which is close to the remote physicist. It is hard to say which will be most difficult to achieve, the technically possible but largely unfunded network infrastructure, or the software for distributed data management which is a technical challenge and will also require close cooperation between autonomous HEP computing centres. Those of us who believe that large HEP experiments will be exciting and rewarding in the next decades are committed to a solution of these problems.
Figure 32 More difficult approaches to remote data access involving split application programs.

REFERENCES


VECTORIZATION OF HEP PROGRAMS

Michael Metcalf

CERN, Geneva, Switzerland

Abstract

The introduction of vector and parallel computers provides an important opportunity for increasing the efficiency of HEP programs. However, the exploitation of these architectures has undesirable consequences as well as offering tempting rewards. This paper examines some of the problems, progress and potential involved.

1. Early Pessimism

The first attempts to test the value of using vector processors\(^1\) in high-energy physics (HEP) took place in the early 1980s, on the Cray I then installed at Daresbury in the UK. The results were discouraging, showing only the improvement with respect to the CDC7600 which is expected from the ratio of their respective clock periods. This result was not unexpected and could be easily attributed to a known feature of HEP programs: their lack of heavily-used, simply structured DO-loops with many iterations. More typical of HEP programs was the presence of loops with relatively few iterations, determined by the smaller scale of the detectors of that time, and of a complicated internal structure in those loops, particularly IF statements and external references, needed to take account of all the various conditions which might and do arise in real data in real detectors.

It seemed immediately obvious that the inherently scalar nature of HEP programs was an insuperable barrier to optimizing them to take advantage of the vector functional units, whether by the use of the automatic vectorization capability of the compiler, or by hand tuning of the code. The problem was exacerbated by the generally flat time-distribution of the programs: there were few or no 'hot-spots' — areas of the code in which a large fraction of the CPU-time was spent, and whose tuning could rapidly bring quick returns. The wide-spread use of memory managers, which rely heavily on the obfuscating EQUIVALENCE statement, did not improve matters.

These early results engendered a feeling of pessimism about the utility of these machines in our environment, which has still not been totally dispelled. Work by Bollini and others [1] in 1985, and by Basile and others in 1986, [2], did nothing to improve the situation, as they demonstrated that by investing some sensible of amount of time into optimizing some 'typical' HEP programs, of the order of some man-months, improvements of only 20% and 34% respectively could be obtained. However, on the other side of the Atlantic, some conflicting evidence was being produced by Levinthal and co-workers, and this will be described below. The important message from these early years is that little gain is to be had from these computers

---

\(^1\) This paper assumes some familiarity with the hardware and software features of supercomputers. For an introduction see, for example, the report of Metcalf and Hendrickson [3].
by depending on automatic or hand optimization of existing code — the only route
to programs well adapted to vector architecture is the design of new algorithms
which properly exploit the vector functional units.

2. Portability in the Supercomputer Era

Before examining the work currently being undertaken along these lines, it is as well
to consider some of the other consequences of the use of supercomputers in HEP.
For some years now, we have well understood how to write FORTRAN 77 pro-
grams of high quality and which are easily portable [4], (always assuming the
presence of the CERN Program Library, where the problem is less tractable). In an
environment involving the use of a range of machines of mixed scalar and vector
architectures, the situation is changed by the potential conflict between scalar and
vector optimization. Taken to extremes, vector optimization using, for instance, large
arrays of temporary variables or explicit vector notation, can have a destructive
effect on code structure and hence maintainability, as well as presenting obvious dif-
ficulties in porting the code back to scalar machines with more limited memories and
without the ability, in the compiler, to recognise the vector notation. This can lead
to some difficult choices: is it better, and in which circumstances, to have one slow
but portable version of a program, or to have two or more optimized ones? To what
extent is the manpower effort required to optimize code for given machines worth-
while? The dangers of maintaining two separate versions of the same program are
well known; whereas it is possible using a source code manager to maintain reason-
ably efficiently versions which represent well isolated differences, for instance in the
area of input/output, maintaining radically different versions at the algorithmic and
data structure level is a different kettle of fish.

As far as libraries are concerned, the situation is somewhat better, insofar as the
effort and difficulties can be centralized, and any benefits made available to all.
Thus, it might be possible to produce vectorized versions for the Cray and the IBM
Vector Facilities (VF), and make them generally available to the whole HEP com-
munity. This contrasts to the situation in optimizing a data analysis program for an
individual experiment, where a large investment would have only short-term benefits
over the life cycle of that single experiment. Similar work has to be undertaken by
each and every experimental group wishing to take full advantage of the supercom-
puters, and it is not clear that the demanding task of optimizing, rather than simply
converting, data analysis codes will be regarded by experimental groups as having a
high priority.

For commercial libraries, vectorized versions already adapted to vector architec-
tures can be purchased, and at CERN we have installed, for instance, the Cray ver-
sions of the NAG library and the finite-element magnet design program TOSCA.

Further obstacles to portability are placed in our path by the actual differences
among the hardware of supercomputers, for instance the presence or absence of a
hardware scatter/gather facility, and by the different ways in which compilers opti-
mize code automatically. Thus, for instance, it can be advantageous on the IBM
VF's to unroll DO loops, whereas this practice is not recommended when using
CFT77 on the Cray.

Yet another set of difficulties is posed by the various extensions to standard
FORTRAN 77 with which the vendors tempt their users. The extensions fall into
three classes:
• Syntax extensions such as those shown in this single statement taken from a program for a CYBER 205:

\[ \text{NSTATUS}(1; \text{NTOT}) = 0 \]

Here we note a name of more than six characters, a use of the non-standard character ",", a definition of the length of an array section by the second part of the subscript, and finally an array assignment.

• Compiler directives as in:

```
CDIR$ IVDEP
C$VDIR
```

which both direct the compiler to ignore vector dependencies in a loop, the former for CFT77 and the latter for VS FORTRAN.

• The increasing use of "Fortran 8x" syntax, for instance to specify operations on array sections as in:

\[ A(1:50) = B(1:50) + C(51:100) \]

Unfortunately, Fortran 8x is not yet a standard, and it is dangerous to introduce features based on draft texts, as the final syntax may change. In addition, some extensions of this type are claimed to be Fortran 8x, but in fact only resemble it semantically but not syntactically.

We are thus faced with an increasing number of dialects, a situation we have encountered several times now on scalar machines over the last two decades, and only the final introduction of a new Fortran 8x standard [5] can halt this tendency. In the meantime, we see the use of pre-processors once again creeping in, examples being VAST [6] which translates FORTRAN 77 DO loops into Fortran 8x array syntax, and AFTRAN [7] which does the reverse. This latter pre-processor could be a useful tool in facilitating program portability across a range of machines, but a new standard would be better.

### 2.1 Multitasking

In addition to the vector capability of supercomputers, most large systems offer the possibility of adapting programs to make simultaneous use of two or more CPUs. The utility of such multitasking in the off-line (as opposed to the on-line) environment is still debatable, (see for instance the discussion by Mount [8]). At CERN, the Cray X-MP/48 has a fixed memory of 8Mwords shared by four processors, and it can immediately be seen that once the average program size exceeds 2Mwords, CPUs start to become idle, unless at least one program is using multitasking. On this particular machine, there are two levels of multitasking: micro-tasking, which essentially performs the individual iterations of loops in parallel, and macro-tasking, which involves a more coarse-grained program restructuring. On the IBM 3090-600, there is usually no overall system advantage to be gained by the use of multitasking, as the virtual memory architecture prevents CPUs being locked out, and multitasking can, in fact, involve a system penalty because of the overhead involved in

---

2 This happened with the PARAMETER statement in FORTRAN 77.
switching tasks. However, an individual programmer working on a large interactive problem might find considerable gains in real-time in using multiple CPUs on one problem, even at the cost of a delay and overhead incurred by other users of the system.

However and whenever multitasking is used, it is clear that the various machine implementations are once again incompatible. Some noble spirits have already tried to take some initiatives to design a portable description of parallelism for use in a FORTRAN context, the latest being a design by Kuck et al. [9] which is possibly intended to become a collateral standard to either FORTRAN 77 or Fortran 8x. If successful, this would certainly be a good starting point for any attempt to introduce parallel constructs into a future Fortran standard.

3. The Challenge

None of the foregoing restraints have prevented some sections of the HEP community from regarding the availability of supercomputers as a challenge to be accepted! The real pioneers have been the theoretical physicists, and this section deals with their work before turning to other successes.

3.1 QCD Calculations

Traditionally, theorists have made relatively light demands on computers, employing them either for numerical calculations, for algebraic symbolic manipulation, or for interactive display and manipulation of functions. Over the last five years, however, there has been a significant change in their needs. The current theory of the strong force, quantum chromodynamics (QCD), describes hadrons in terms of their constituent quarks and of the colour forces between the quarks. However, the description of, for instance, a proton as consisting of two up quarks and one down quark, should not be taken as a static one. It is a description of the average state of a proton, and in fact there is a rapid creation and annihilation of virtual quark-antiquark pairs. This process is governed by Heisenberg's Uncertainty Principle, which allows such virtual particles to exist only fleetingly.

One of the big unanswered questions in physics is why a particle, such as a proton, has the actual mass and other properties which we observe. Is it possible, starting from first principles, to predict and not simply to measure these properties? Developments which brought K. Wilson a Nobel Prize have suggested an approach known as QCD lattice gauge theory; the computational load which can potentially be generated by the application of this method is estimated at many years of time on any existing supercomputers! This sudden surge in theoreticians' demands dwarfs the traditional, already large requirements of the experimentalists. It will only be finally satisfied by the completion of projects such as the GF11 at IBM and APE in Italy, which are designed to provide an architecture tailored to this one problem.

The technique involved is to place a four-dimensional grid, or lattice, over the space-time volume occupied by the quarks of which the hadron is composed. The colour forces between the constituent quarks can be represented as fields, and the distribution of energy within the lattice is defined by a set of unitary 3 x 3 matrices of complex (in the mathematical sense) elements for each node of the lattice. The quantum fluctuations are simulated by a Monte-Carlo algorithm [10]. However, the ideal calculation should be performed on a lattice whose spacing becomes van-
ishingly small and hence contains a very large number of points, whereas in a feasible calculation on a real computer one is limited to, perhaps, a lattice of 16x16x16x32. In an actual program, the declarations of the fields might appear in the form

\[
\text{COMPLEX X(3, 3, 16, 16, 16, 32)}
\]

and an immediate difficulty in using large vector processors is then apparent, namely that the vector lengths involved are short, whereas these supercomputers are at their most efficient only when operating on long vectors. Some considerable efforts have been invested in adapting QCD lattice gauge programs, originally developed on slow, serial computers, to run on vector machines. Not only must methods be developed to overcome the short vector problem, but handling effectively the transfer of data between memory and the vector registers also presents a major problem inherent in the geometry of the four-dimensional lattice. Experience has shown that, in the limit, only the use of assembly code can make full use of the capabilities of a vector processor to tackle a problem ill matched to this type of computer architecture. A further obstacle to using existing machines is that their memory sizes are too small, and the transfer of data between backing store and memory then requires efficient management in the programs.

Since serial machines are too slow, and vector processors too expensive and not well adapted to this problem, strenuous efforts have been made by Wilson and his colleagues to persuade industry to develop more appropriate hardware. One such development, mentioned above, is the GF11 supercomputer, the result of a collaboration by Beetem, Denneau and Weingarten [11] at IBM. This machine will consist of 576 processors connected via a network capable of realizing any permutation of the processors and instantaneous reconfiguration. Operating in parallel, these processors should deliver 10Gflops of computing power, allowing a job estimated to require \(3 \times 10^{17}\) arithmetic operations to be performed in one year, rather than a century on an existing vector processor. In the meantime, work on supercomputers around the world has continued, especially on Crays, but also for instance on the ICL DAP, and many results have been reported, for example Toussaint [12]. Typically, these results confirm the theories qualitatively but the numerical results obtained are still inaccurate. Future work will concentrate on large-scale variational calculations in QCD, on the evaluation of the weak coupling or of the strong coupling to high orders, as well as on as yet uninvented methods. All of these would require even greater computing power.

### 3.2 Pattern Recognition

The pattern recognition program for a large, modern experiment is very complex. Unlike the relatively simple detectors at earlier generations of accelerators (with notable exceptions like the SFM at the ISR), detectors nowadays are to a large extent composed of a substantial number of subdetectors. Each subdetector requires its own pattern recognition module, with a lot of additional code for associating the various tracks, track components and showers into a single event, and for performing fits of tracks and vertices inside magnetic fields, (see, for example, Bock et al. [13]). Bearing in mind that such a program is in a continuous state of change, adapting it well to a vector architecture is a difficult undertaking, perhaps to be attempted only if one part of the program dominates in terms of the CPU time required to process the events. The only work so far reported is based on algorithms developed for classical spectrometer or cylindrical configurations, and these are discussed in this section.
3.2.1 Fermilab E711

Much of the pioneering, but sometimes controversial, work in vectorizing HEP programs has been carried by the enthusiastic team led by D. Levinthal at Florida State University in Tallahassee. One example of this was the attempt to convert a running pattern recognition program for the Fermilab experiment E711 from the VAX to a highly optimized version on a CYBER 205. No effort was spared to achieve the maximum possible benefit from the architecture of this machine and from the extensions provided by the FTN200 compiler. To this end, the algorithms were recoded a number of times, using finally the APL-like features provided by the compiler and its run-time library. The resulting code was totally non-portable, and relied so heavily on *explicit* vectorization, that it was compiled with the automatic optimization disabled. Since this paper is less concerned with difficult and contentious timing results, than with drawing attention to the need to modify the way we approach the design of algorithms and data structures, for detailed timing figures the papers by Levinthal *et al.* [14] and Levinthal [15] should be consulted.

The experimental configuration is shown in Figure 1. The significant feature is the array of four drift chambers which follow the spectrometer magnets, shown here in the vertical projection (no bending). A naive *scalar* method to find individual tracks in such chambers would be to combine each hit on each plane with each hit on every other plane, and test the resulting combination against a \(\chi^2\)-hypothesis. This gives an \(O(n^4)\) combinatorial explosion. The addition of a fifth chamber in 1987 would clearly have worsened the situation, although simpler tests on two and three point combinations can obviously be incorporated to reduce the worst-case limit. *Vector* methods, on the other hand, start with a completely different approach which depends basically on storing, once and for all, a list of all possible allowed track combinations, based variously on either so-called *cells* or on wires, and then testing these against the actual hits recorded in the event.

![Diagram of the E711 experiment](image)

*Figure 1:* The E711 experiment.
3.2.1.1 Vector method I.

The first algorithm tried depends on dividing each chamber in each projection into cells of several wires, where a cell is defined such that if a ‘seed’ track with cell numbers say 1, 2, 2, 2 exists, then each new valid track can be obtained by incrementing these cell values by one on each chamber. In other words, the combinations 2, 3, 3, 3 and 3, 4, 4, 4 etc. can be derived from the given initial seed. These allowed combinations are stored, and then for each such combination and each plane, vector instructions are used to increment the counter of each combination where the corresponding cell hit is present in the actual event, and a second vector instruction is used to signal that a track is actually present if three or more hits are finally counted. The drawbacks of this approach are that a slow clean-up algorithm is required to sort the found tracks, and that it is basically too coarse-grained to be satisfactory.

3.2.1.2 Vector method II.

The second method tried was based on overlapping cells. Once again, each possible track combination, based on Monte Carlo simulations, is stored. In a loop over each plane, vector instructions use the track combinations on a plane to compress the actual hits, Figure 2. Finally, a track is deemed to be found if three or more hits are present. This algorithm suffers from the disadvantage that the scatter/gather instructions on which it is based are relatively slow, and that three-hit tracks are found several times so the duplicate ones must be eliminated by a sort.

<table>
<thead>
<tr>
<th>Wire no. on plane 1</th>
<th>Value</th>
<th>Plane no.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>2</td>
<td>-- &gt;0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>3</td>
<td>-- &gt;0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td>6</td>
<td>-- &gt;1</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td>9</td>
<td>-- &gt;1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Compression of hits.

3.2.1.3 Vector method III

The final algorithm, and the one actually used for data analysis, once again stored each track combination based on cells, but this time in a bit array. (Bits are addressable on a CYBER 205 and a form of BIT data type is available as an extension.) The bit array is triply dimensioned: number of tracks by number of cells by number of planes. A bit is set if it corresponds to an allowed hit on that plane for that track. Thus,

\[ \text{BITX}(3251, 57, 1) \]

has the binary value 1 if track 3251 has cell 57 on its first plane. This form of data structure requires more storage, but the algorithm will run faster, as 128 bits may be logically combined per cycle. This is a typical trade-off between space and time.
Once these patterns are stored, to process an individual event a double loop over each actual hit and each plane can be made, in which the set bits are ORed into a two-dimensional array of tracks by planes. This is effectively a projection of the stored data masked by the actual hit cells. A track is deemed to be present if three or more bits are set on a row corresponding to a track, as shown in Figure 3.

<table>
<thead>
<tr>
<th>Track no.</th>
<th>Plane no.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>2000</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2001</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
| 2002      | 1 1 1 1   | ← track 2002 is in these data
| 2003      | 0 0 0 1   |
| 2004      | 0 1 0 0   |
| 2005      | 0 0 0 0   |
| 2006      | 1 0 0 0   |

Figure 3: Plane hits.

The methods outlined here all require additional logic for duplicate and short tracks, but retain their important characteristic of being linear in time with the amount of data to be handled. The fact that they depend on explicit vector instructions mapping onto the CYBER 205 hardware makes them fast, but are a lesson in algorithm design which initially ignored any consideration of portability. However, they are now shown to be readily convertible to the Cray.

3.2.2 MARK III

Among other work in pattern recognition is that carried out by Hauser [16] for the MARK III detector at SLAC. A sketch of the basic topology is shown in Figure 4;

Figure 4: The MARK III experiment.
for our purpose it is sufficient to know that it consists of eight concentric chambers which detect the passage of tracks emerging from a common vertex as they pass through a magnetic field.

The pattern recognition can be arranged to proceed in six steps, each of which can be vectorized. They are:

1. The data are unpacked into linked lists with pointers between the layers, cells, wires and hits. This is carried out in parallel for each layer.

2. The tracks are found using a stored track 'dictionary' of 12832 combinations. This is done in parallel by \( r \), range.

3. When a particle passes near a sense-wire in a wire chamber, its passage is detected by the arrival of drift electrons at the wire. It is possible to measure the time taken for the electrons to drift to the wire, but not to know whether they come from the left or the right. This is known as the left-right ambiguity, and typically has to be resolved by the pattern recognition program: the hits will fit better to a trajectory when coming from one side rather than the other. In this case the ambiguity is resolved by solving linear equations in parallel over distinct segments of the detector.

4. A further problem with real data from a detector with a finite measuring resolution is to distinguish between two separate tracks which are so close that they share one or more signals, but in a way which produces ambiguities in the pattern recognition. Such ambiguities can be resolved by pseudo-\( \chi^2 \) fits, performed in this case in parallel by bundle of close tracks.

5. The dip angle of each track is calculated from the stereo information, in parallel for each track.

6. Finally, a helix fit of the trajectories in the magnetic field is performed, in parallel for each track.

Methods such as these have been adopted by other experiments, and serve as a model for a parallel approach to pattern recognition in a single major detector component. The difficulty is that, for many experiments, this phase of the pattern recognition is just part of a more complex program, and each part would have to be vectorized in order to gain an overall large gain. To the extent that that is not done, Amdahl's Second Law limits the benefits obtained.

### 3.3 Event Simulation

By way of contrast to the pattern recognition programs mentioned earlier, which are specific to a given experiment, there is an increasing trend in experimental high-energy physics to use a small number of general-purpose programs to perform the various steps required in the generation of simulated event data. Such data are required in very large numbers in the design of physics detectors and pattern recognition programs. They are also used in the evaluation of the physics data. For instance, new physics effects can be detected by comparing distributions obtained from actual data with those obtained from data simulating all known physics effects in the same detector.

Over the years, a few programs have come to dominate simulation activities: for the generation of the basic physics interaction and the production of the long-lived
decay particles, the LUND program (from the University of the same name) is in wide use; for the generation of electromagnetic showers the EGS program from SLAC; for the generation of hadronic showers the GHEISHA program (mainly work by H. Fesefeld of Aachen); and for full detector and event simulation, with some of the others as callable modules, the GEANT program of Brun et al. [17].

The basic steps in any full simulation are the following:

- generate the final state of the physics event, with a program such as LUND;
- track all particles through the various sub-detectors, taking account of the magnetic field where present;
- consider at each step in the tracking all appropriate physical processes — decays, bremsstrahlung, absorption, etc.;
- generate showers (cascades) in the calorimeters, a slow process due to the very large number of particles generated and followed through to decay or absorption (see Figure 5), and which involves, for EGS, the consideration of a wide range of physics processes — annihilation, Bhabha scattering, bremsstrahlung, Moller scattering, multiple scattering, Compton scattering, pair production and the photo-electric effect (with even more processes to consider for hadronic showers);
- calculate, for each particle, digitized information such as would be generated by the sensitive elements of a real detector;
- record these data in a format identical to that used for the real events.

![Figure 5: The development of an electromagnetic shower.](image)

The production of tens of millions of simulated events, numbers required to explore in a statistically significant manner all the physics channels of an experiment, demands huge amounts of computing time, comparable to that needed to process the real data. In addition, the simulated data must themselves also be processed through the chain used for the real data. In these circumstances, there is a clear advantage to be gained by investing, centrally, some considerable effort in speeding up these programs. The investment becomes available to all, and allows the processing of larger numbers of simulated events, leading to better physics results for many experiments. That is a current goal of those involved.
3.3.1 Fermilab E711

Once again, Levinthal and co-workers have carried out some early investigations [14], [15]. In the calorimeters of the experiment already mentioned, their approach to shower generation was to use a parametric description of the shower in a vectorizable algorithm. Using a formula to describe the energy loss in terms of its hadronic and electromagnetic components, each with exponential terms and involving a total of four random variables, two 3-D grids are notionally placed over the detector, and energy generated in each grid point. The energy over the whole grid is summed. The digitizing is performed by associating each grid point with an actual active element, and finally summing the energy seen by each such element.

3.3.2 EGS4V

A quite different approach was used by Miura [18] in an experimental attempt to vectorize EGS. In the scalar version, each particle in the shower is tracked separately through the detector. Any newly generated particle is placed on a stack, and tracks are taken from the stack for further processing until the stack is empty. A feature of this program is that it contains no loops at all, and is therefore totally unvectorizable in that form. Miura’s method is to keep each particle in separate lepton and photon queues, along with a code for each particle determining which physics process it is about to undergo. The program then processes, in parallel, all the particles corresponding to the process code with the highest sum. In this way, the program has loops, and the longest possible loop (in terms of the number of iterations) is the one chosen preferentially. Finally, the queues of particles are updated, and garbage collection is carried out on the heap storage area.

This work has been carried out only for the ideal case of a semi-infinite block of homogeneous material, thus ignoring the extensive computations required in a real detector to determine its geometry, and of an advantageous initial beam energy. The large speed-up obtained must be considered against this fact, and also in the light of the very large expansion in the size of the program caused by the introduction of arrays in the place of scalars. This growth would become intolerable at very high energy.

This is also an appropriate place at which to point out that the introduction of loops into any program where there was none before is an excellent way of improving code even on a scalar machine. The presence of loops enables a scalar compiler to perform a whole series of optimizations [20] which would otherwise be impossible. Thus, in making timing comparisons between scalar and vector versions of code, it is vital to compare like with like, and the vectorizable algorithms should be run on the scalar machine.

3.3.3 GEANT3V

A considerable effort is being devoted to the vectorization of the most widely used simulation program in HEP, GEANT. This work is currently being done by Dekeyser and Georgiopoulos [19], and concerns the physics and geometry as well as just the tracking. The change being made to the tracking algorithm is illustrated in Figure 6. The scalar version of GEANT has an algorithm which tracks each particle through each volume of a detector in an order determined by the generation and decay or annihilation of the particles. Each track is followed through all the volumes through which it passes, before dealing with the next track on the stack. In the vectorized version, the numbering of the tracks is such as to confine a given sequence of

363
successive numbers to a single volume. This allows similar tracks in a volume to be treated together in a loop which follows them all to the boundary of the volume. Here they are transferred from a local to a global stack, and the procedure repeated for all volumes. A simulation of this algorithm, in the context of the OPAL detector, has shown that the 170,000 scalar steps required reduce to 5500 vector steps. In an ideal case in which all media could be treated simultaneously, this would even reduce to 550 steps.

The vectorization of the GEANT physics routines is similar to that described for EGS4V: separate arrays of particles requiring physics processing are kept according to the type of processing to be undertaken, gamma decay, bremsstrahlung, δ-ray, etc.

In the case of the geometry routines, those which provide information on the location of a given point according to the part of a sub-detector of a uniform medium in which that point is contained, vectorization is possible by the creation of separate arrays for each set of particles in a different volume. Here, the definition of a volume is not necessarily of a physically distinct entity, but includes all volumes of the same shape and material, even if the size and position are different. This arrangement leads to the possibility of being able to make vectorized calls to routines such as GNEXT (what is the path length to the next volume?), and GINVOL (is this point still in the current volume?).

The work on GEANT is of the utmost importance in determining whether the introduction of vector processors into HEP in general and at CERN in particular
can ever be regarded as worthwhile if judged in terms of cost/benefit at list prices, particularly as it takes much longer to generate a simulated event than to analyse a real one. It is made difficult by the need to bear in mind the often different requirements of experiments as diverse as those at LEAR and the planned SSC, and by the nine orders of magnitude of particle energies which have to be considered. It is at least encouraging that the simulation shows that, if a minimum useful vector length is set at 20 elements, then 90% of all subroutine calls in this version are vector rather than scalar in nature. The improvements so far are reflected back into the versions running on scalar machines too.

3.3.4 Particle Transport

As explained at the beginning, the earlier attitudes to vector processors were fairly pessimistic. The first hint that Monte Carlo codes might, in fact, be successfully implemented on them came from outside HEP, namely from the field of nuclear reactors. Here too there are problems of tracking (or transporting) neutrons through shielding or photons through plasma. An example of the adaptation of such programs to vector, parallel and vector-parallel architectures has been described, for instance, by Martin et al. [21]. Work in this field has been an inspiration to progress in high-energy physics.

The basic problem is similar to the one of a shower as described for EGS, although with fewer processes to consider. The adaption of the Monte Carlo program to a vector architecture is similar to the one described for GEANT tracking. In addition, the code has been adapted using several different schemes to run on parallel architectures, in the example given below the IBM 3090/400. This has four processors, and the relevant software to enable a program to make use of two or more of them simultaneously.

The first algorithm is to divide the problem into physical volumes, each of which is assigned to a processor. This method leads to poor load balancing of the processors, as the particle distribution is unequal between the volumes. Also, interprocessor communication is a problem, as particles can cross volume boundaries. A second version assigns a given energy band to a processor, but this suffers from the drawbacks of having to load the whole geometry description into each processor and of posing a problem when the energy of a particle changes to another energy band. A third algorithm assigns each zone in which a photon originates to a processor; this still has less than optimal load balancing, but is better. The problems of making effective use of multiprocessor architectures are thus evident.

A final algorithm used was a parallel-vector version designed to run on a Cray X-MP, which may be run with its vector CPUs working in parallel. The basic idea is to make a parallel loop over each sub-sample of particles, with a further vector loop over the particles themselves. Work in this field is still progressing, and close contact is maintained between the two communities.

4. Conclusion

This paper represents a qualitative assessment of the difficulties and achievementsin preparing HEP programs to use, in an effective fashion, the supercomputers whichare now increasingly available to our community. Automatic vectorization of littlevalue, and fiddling with existing scalar code by hand is just a blind alley. There is
an unresolved problem of program portability. Keen, inventive minds have to find
algorithmic solutions, but this can be costly in terms of manpower, as well as of
physical memory. We have to begin exploring the potential use of the multitasking
features of supercomputers, in order to gain experience we may absolutely need lat-
er. Unfortunately, it will be the generation of experiments after LEP that will first
benefit substantially from any advances.

In all, a lot of work and research are necessary. The outcome will depend also
on developments in hardware and software over which we have no control, and
where there is no consensus as to the best methods. That said, this is an exciting
and promising field of research, with great rewards for the winners.

ACKNOWLEDGEMENTS

I wish to thank those colleagues who read early drafts of this paper, and particularly
R. Doran for his painstaking efforts to improve its readability.

*   *   *

Bibliography

1. D. Bollini, C. Chiccoli and P. Pasini, *Simulation software for DELPHI on

2. M. Basile, M.L. Luvisetto and E. Ugolini, *High energy physics event process-

3. M. Metcalf and R. Hendrickson, *Coding for the Cray X-MP using CFT77: An


5. M. Metcalf and J. Reid, *Fortran 8x Explained*. Oxford University Press,


7. J.L. Dekeyser, C. Georgiopoulos, F. Hannedouche, G. Riccardi, J. Vagi and
   S. Youssef, *The AFTRAN Vector Preprocessor Project*. FSU-SCRI-88-43,

8. R. Mount, *Present and Future Computer Architectures*. CERN/88/03, Gene-


    T.J. Watson Research Center, Yorktown Heights (1985).

    (1987) 111—120.


VECTORIZATION OF TRANSPORT MONTE CARLO CODES
Kenichi Miura
Mainframe Division, Fujitsu Ltd., Kawasaki, Japan

Abstract

This lecture note reviews the computational techniques, the
coding methodology and the performance for the transport Monte
Carlo simulation on the vector supercomputers, taking a
cascade shower simulation code EGS4 as an example. It is
shown that a reasonable vector performance can be obtained by
treating the problem in a different manner from the
conventional sequential processing in such a way as to exploit
the vector architecture of current supercomputers. This note
also discusses computational techniques and issues in parallel
processing the transport Monte Carlo codes on the
shared-memory parallel systems, and compares vector approach
with parallel approach.

1. INTRODUCTION
1.1 Parallelism in scientific computations

In recent years, the demand for solving large scale scientific and
engineering problems has grown enormously. Since many programs for solving
these problems inherently contain a very high degree of parallelism, they
can be processed very efficiently if algorithms employed therein expose
parallelism to the architecture of a supercomputer. Parallelism in such
computations is either in the data (that is, the identical operations can be
performed on the elements of large array of data), or in the control flow
(that is, multiple streams of instructions can be executed in parallel).

Pipelining and parallel processing are the commonly used techniques in
today's supercomputers [1-2]. In pipelining, array data (to be called
vector hereafter) can be processed as one entity with single instruction so
that subsequent elements of data can begin execution before previous
elements have completed execution; an assembly line approach. This approach
is commonly called vector processing.

In parallel processing, on the other hand, processors are connected
together via interconnection networks or via memory, where each processor

Mailing address: Computational Research Dept., Fujitsu America, Inc.
3055 orchard Drive, M/S B2-7, San Jose CA 95134, U.S.A.

368
works either on a segment of the array data synchronously, or on a different part of the program asynchronously.

Today's supercomputers such as CRAY X-MP, CRAY 2, ETA-10, AMDahl/ FUJITSU/SIEMENS Vector Processor Systems, Hitachi S-820 System and Honeywell/NEC SX System, mainly depend on the vector processing approach to boost their performances, with parallel processing capabilities besides vector processing in some cases. It should also be noted that even within the uni-processor configuration of supercomputers, the parallelism is also incorporated in the hardware structure in several ways: replication of identical arithmetic pipeline circuits, multiple functional pipeline units, concurrent operation of the scalar unit and the vector unit. One such example is shown in Fig. 1 [3].

In order to fully utilize the architecture of a supercomputer, development (or rediscovery) of suitable algorithms is very important. This lecture note will discuss the algorithm issues as well as the software engineering issues involved in the high-performance computation of the transport Monte Carlo simulation, particularly how the parallelism in the application programs are to be matched with a given supercomputer architecture. The techniques discussed here are not solely for the Monte Carlo applications, but should be applicable to other applications as well.

2. GENERAL REMARKS ON VECTOR CODING AND VECTOR ALGORITHM DEVELOPMENT

2.1 What is "vectorization"?

Vectorization is the act of tuning an application code to take advantage of the vector architecture [4]. Vectorization may be done by the compiler alone, with the aid of the so-called the compiler directives, with explicit function calls to some special library, or through code restructuring, depending on the capabilities of the compiler as well as on the complexity of an application code.

2.2 New discipline, or new way of thinking

It is worth mentioning, at this moment, the fundamental differences in the conventional sequential (to be called scalar hereafter) and vector algorithms based on the author's own observations. In fact, a non-negligible number of existing programs developed for scalar machines are structured in such a way as to obscure the parallelism which the physical phenomena or the computational models thereof inherently possess. Some of the differences may be summarized as follows:
i) Tradeoffs between Floating point arithmetic operation counts and complexity of the codes

One of the major concerns in the scalar algorithms is to minimize the number of floating point operations; the complexity of the program control flow and the irregular access pattern to array data are comparatively insignificant. On the other hand, efficient vector algorithms require simple program control flow and carefully organized data access pattern to array data: the constraint on the number of floating point operations is relatively eased. Therefore, various numerical algorithms must be re-evaluated with new criteria.

ii) Cost of memory referencing operations

In the supercomputers, moving the data to and from memory (that is, the load/store operations) is as expensive as the arithmetic operation. Furthermore, even when the algorithms themselves are readily vectorizable, the performance of supercomputers is quite sensitive to the access patterns to the array data in the main memory, due to the highly interleaved memory structure.

Especially, close attentions should be paid to the load/store operations when array data are accessed with indices which have the power-of-the-two strides or with arbitrary indices (that is, the indirect addressing or the random gather-scatter operations) for possible severe memory access conflicts. A typical example is the algorithms based on the table-lookup technique, which is optimal for scalar processing, but may not be necessarily optimal for vector processing.

iii) Importance of storage size in exposing parallelism in array data

Because of the storage size constraint in the conventional machines, programmers tend to utilize the same storage location as much as possible, hence the recursion. In order to resolve the recursion, or to expose a higher degree of parallelism by code restructuring, vector algorithms in many cases require a larger size of data storage than in the scalar algorithms. In practice, a careful organization of array data structure is the key factor in successful vector coding.

iv) Psychology of programming

Programmers tend to localize the most complex part of the program logic in the innermost loops for ease of thinking and debugging. Vectorization of such programs requires semantic restructuring rather
than simple syntactic transformation. One typical example is the transport Monte Carlo simulation programs as will be discussed later on.

v) Architecture-dependent features
The optimal vector algorithms depend on architectural characteristics of supercomputers, such as the available main storage and vector register sizes, definition of vectors, and type of available arithmetic concurrency. Especially, the vector data handling capabilities such as compressing and/or expanding vectors (Fig. 2) and random gather/scatter operations (Fig. 3) are very important for wider applications of vector supercomputers.

3. CHARACTERISTICS OF TRANSPORT MONTE CARLO SIMULATION CODES
3.1 Scalar nature of code structure
Supercomputers have successfully exhibited very high performance for applications like PDE or signal processing. These machines, however, usually give only the scalar performance for the existing Monte Carlo simulation codes such as neutron and radiation transport calculations in nuclear engineering, phase-space Monte Carlo simulation and cascade shower Monte Carlo simulation in high energy physics [5-9]. The Ray tracing technique in computer graphics may also be included in this category.

Although the transport Monte Carlo calculations consume vast amount of computation time and are expensive, they constitute the only feasible means of solving many problems involving complicated interactions and arbitrary geometrical structures. A typical control flow of the transport Monte Carlo code is depicted in Fig. 4. In this simple model, each particle is transported through the media, boundary crossing is checked in a given geometrical structure, then it encounters some interactions. This process is continued until the particle escapes from the structure, or until there is no further interest in this particle. Processing of each particle involves many data-dependent IF tests, due to the stochastic nature of the computational model of physical interactions, hence leaving very little parallelism within the particle loop.

It should be noted, however, that a very high degree of parallelism exists at the particle level, since there could be thousands of particles to be simulated, each of which can be treated independently of others. Due to the above-mentioned reasons, the transport Monte Carlo simulation has generally been regarded as a perfect application for the parallel processing rather than for vector processing.
3.2 Basic strategy for vector processing

As the vector supercomputers become widely available, quite a few reports have been recently made regarding the efforts in vectorizing the transport Monte Carlo simulation [7-13], most of which strongly indicate that vector approach is indeed worthwhile.

The basic strategy for vectorization is similar in all the reported works, namely to pool particles in a common data stack, and to form vectors with particles possessing identical characteristics by gathering them from the stack. In this way, many particles can be processed in one pass. In other words, the loop structure in Fig. 4 is inverted so that the particle loop becomes the innermost in each block of the code. In order for this strategy to be successful, it is very important to carefully design the data structure so that the vectorized algorithms can exploit the parallelism contained in the problem; the scalar Monte Carlo codes in many cases adopt inherently sequential data structure (such as a push-down stack).

The efficiency of vector processing also depends on the varieties of interaction patterns and/or complexities of geometry, both of which strongly influence the complexity of the codes as well as the effective vector length at each step of simulation. In the actual vectorized codes, most of the loops are heavily populated with the nested IF-THEN-ELSE structures, and the compiler's capability to vectorize such complex loops is essential in obtaining a good vector performance. In practice, code-restructuring involves a deep understanding of the code itself and takes considerable amount of effort, since the Monte Carlo codes are usually very large in size. More detailed discussions on these features will be made in the sections 4 and 5.

The reported vector performances over the scalar range anywhere from 1.4 to to 85, most of them falling between 5 and 10 [13].

4. VECTORIZATION TECHNIQUES FOR MONTE CARLO SIMULATIONS

Some of the common techniques in vectorizing the Monte Carlo Simulation codes are described in this section.

4.1 Vectorization of DO loops containing IF Tests

One of the most important issues in vectorizing a Monte Carlo simulation code is how to vectorize the DO loops containing IF tests. In a typical Monte Carlo simulation code, there are two types of IF structure, namely the feed-forward type IF test and the feed-backward type IF test [10]. These two types of IF structure must be treated separately in vector coding.
The first type is encountered when different computations are to be performed depending on the result of an IF test (Fig. 5a). For example, positrons may be treated differently from electrons, or the particles in high energy range may be treated differently from those in low energy range, etc. The Feed-forward type IF test can usually be vectorized by the vectorizing compilers. Fig. 5b depicts a simple case when the mask bits are used to process the two branches separately. Note that the feedback path has been eliminated in the vectorized code.

The second type is usually encountered in the rejection sampling routines [14-15], where the data dependent values of the trial (rejection) functions are compared with the random numbers and the trials are repeated until the sampling is accepted (Fig. 6a). The vectorization of the Feed-backward type IF requires some semantic modification of the original scalar code. The common approach is to define two temporary buffers, the Accept Buffer (Buffer 2 in Fig. 6b) and the Reject Buffer (Buffer 1 in Fig. 6b). The accepted samples are compressed into Buffer 2, while the rejected samples are compressed into Buffer 1 at the end of each iteration. In the subsequent iterations, Buffer 2 becomes the input to the loop. This process is repeated until Buffer 2 becomes empty. In the vectorized code, the feedback path still exists, but it has been moved to the outside of the loop, hence reducing the IF test to the Feed-forward type. As is clear in the above description, the vector data handling capabilities such as the compressing vectors and/or the indirect addressing are essential for this operation.

In many practical cases, these two types of IF tests are mixedly used and heavily nested to construct a very complex IF structure.

4.2 More on vectorization methods for rejection sampling routines

It should be noted that the vector performance of a loop with the Feed-backward type IF tests, if vectorized in the way described in 4.1, is sensitive to the rejection probability. When the rejection probability is high, the vector length of the Reject Buffer decreases very slowly, and many iterations are required before it becomes empty. In practice, there are three different ways to vectorize the Feed-backward type IF tests, which should be used selectively depending on the rejection probability, the complexity of the IF structure and the frequency of usage of such a loop.

Method 1: Partial vectorization

The portion of a DO loop which contains the feedback path is separated from the rest of the loop body, and the rejection sampling for each element of the vector is processed individually in the scalar mode. Although not
fully vectorized, this approach is adequate for the cases where such computations are performed very infrequently. Coding effort is minimal in this method.

Method 2: Local Vectorization

This method has been described in 4.1; the buffers are locally defined, and the feedback path is moved to the outside of the loop, but still the semantic modification is confined within a subroutine which contains this loop. This is the approach most commonly used in practice.

Method 3: Global Vectorization

The buffers are globally defined, and the feedback path is moved even to the outside of the boundary of the subroutine which contains the loop. This approach is effective when the rejection rate is high; subsequent calls to this routine may be deferred until a sufficient number of particles have been accumulated in the Reject Buffer. On the other hand, this method requires global code modification, and the code structure tends to become complex.

4.3 Random number generator

The most commonly used technique for random number generation in the transport Monte Carlo codes is the congruential method due to its simplicity [14–15]. In EGS4, for example, the following multiplicative congruential method is used:

\[
\text{Loop over } i \\
\text{Iseed } = A \times \text{Iseed modulo } 2^{32} \quad \text{(Random seed in integer format)} \\
\text{Ran } = \text{Iseed } \times 2^{-32} \quad \text{(Normalized floating-point random number)},
\]

where \( A = 663608941 \).

Although this algorithm may seem recursive, it can be easily vectorized if the multiplicative coefficients \( (A, A^2, A^3, \ldots, A^N) \) modulo \( 2^{32} \) are pre-calculated and stored in an array. At the time of random number generation, each element of this array (up to the desired number of random numbers not exceeding \( N \)) is multiplied by the current value of Iseed and the resulting integer random numbers are normalized to obtain the floating-point random numbers, all in the vector mode. Only the last integer random number needs to be stored as the seed for future use. Similar technique is also applicable to the linear congruential method [16].

The vectorized random number generator generates the identical random number sequence as the original scalar algorithm. It should be noted, however, that the vectorized Monte Carlo code and the scalar Monte Carlo
code do not necessarily produce the identical simulation results since the order in which the random numbers are used may be different in two cases.

4.4 Vector algorithms for scoring computations

In the Monte Carlo Simulations, the simulation results are usually represented by accumulating the physical quantities in the regions of interest or, equivalently, in the form of the histograms. Typical examples are the energy deposition in the detectors, or the angular distribution of the radiation. This type of computation may be generically called scoring. A simple model of the scoring computation may be depicted as follows:

\[
\text{Loop over } i \\
\quad k = \text{Ireg}(i) \\
\quad \text{Esum}(k) = \text{Esum}(k) + \text{Edep}(i),
\]

where \( i \) is the particle index \((1 \leq i \leq N)\), \( \text{Edep}(i) \) is the energy to be deposited by the \( i \)-th particle, \( k = \text{Ireg}(i) \) is the index of the region where the \( i \)-th particle is located \((1 \leq k \leq K_{\text{max}})\), and \( \text{Esum}(k) \) is the accumulated energy in the \( k \)-th region.

This type of computation is "inherently sequential" and not vectorizable in its present form, since more than one particle may deposit energy in the same region (to be called recursion). If the CPU time for the scoring computation is insignificant, it may be left scalar. If the CPU time is significant, on the other hand, there are several vectorizable algorithms which can avoid the recursion. Some of such algorithms are briefly described in the following.

Algorithm 1: Running Sum Method

(1) Sort \( k = \text{Ireg}(i) \) in the ascending order.
(2) Rearrange \( \text{Edep}(i) \) accordingly to obtain \( \text{Edep}(i') \).
(3) Count the runs for all distinguishable \( k \)'s in (1) (say, \( R(k') \)).
(4) For each \( k' \), take the summation of \( R(k') \) elements of \( \text{Edep}(i') \) and store it in \( \text{Esum}(k') \).

Algorithm 2: Sort and Stride Method (due to B. Parady [17])

(1) Sort \( k = \text{Ireg}(i) \) in the ascending order.
(2) Rearrange \( \text{Edep}(i) \) accordingly to obtain \( \text{Edep}(i') \).
(3) Find the maximum run of \( k \) in (1) (say, \( R_{\text{max}} \)).
(4) Accumulate \( \text{Edep}(i') \) in the corresponding \( \text{Esum} \) with the stride of \( R_{\text{max}} \).

Algorithm 3: Two-dimensional Work Buffer Method (due to S. Orii [18])

(1) Define and clear a two-dimensional work buffer \( \text{Wbuf}(K_{\text{max}}, L) \).
(2) Accumulate \( \text{Edep}(i) \) in \( \text{Wbuf}(\text{Ireg}(i), i) \), \( L \) particles at a time.
(3) After all \( \text{Edep}(i)'s \) have been accumulated in \( \text{Wbuf} \), accumulate
Wbuf(k,*k) in Esum(k) for each k.

Each algorithm has its own merits and demerits, and should be
selectively used, depending on the N, Kmax and available work buffer area in
the memory. Typical vector vs. scalar speedup factor is from 5 to 10, if an
appropriate vector algorithm is used.

5. VECTORIZATION OF ELECTROMAGNETIC CASCADE SHOWER SIMULATION CODE EGS4
   --- A CASE STUDY ---

This section describes the approach for vectorizing the electromagnetic
cascade shower Monte Carlo code EGS4, and shows that the vector
supercomputer with powerful vector data handling capabilities can achieve a
good vector performance. All the works which have already been reported on
the experiences in vectorizing the transport Monte Carlo simulation codes
are concerned with the transport of the neutral particles such as neutrons
and photons [7-9,12], but no research work has been reported in the area of
the charged particle transport, especially the cascade shower simulations.
Therefore, this section is devoted to this new subject, taking the author's
own experience in vectorizing the EGS4 as an example.

5.1 briefly describes the concept and the structure of the original
EGS4 code. 5.2 describes the general methodology for vectorizing EGS4. 5.3
describes the timing results of the vectorized version of the code with
respect to the original scalar code. The vector supercomputer used for this
research is AMDAHL 1200 Vector Processor System with FORTRAN77/VP
Vectorizing Compiler [19].

5.1 Overview of EGS4 Code

EGS4 is the latest version of the EGS (Electron-Gamma Shower) Code
System which has been developed at the Stanford Linear Accelerator Center by
W.R. Nelson, et al., over many years [6]. This code system is a general
purpose package for the Monte Carlo simulation of the coupled transport of
electrons and photons on an arbitrary geometry. EGS4 is widely used in high
energy physics (simulation of electromagnetic cascade showers) and in
medical physics.

An electromagnetic cascade shower starts with one particle with very
high energy (say, above 1 Gev) which subsequently creates many particles
through radiation and collision (Fig. 7). The physical processes
incorporated in EGS4 are listed in Table 1, and to be collectively called
interactions hereafter. Particles in a shower are transported through the
media, and are eventually discarded as they lose their energy below a
prescribed threshold through collision and radiation processes or as they
escape from the geometrical structure. The analog Monte Carlo approach has been adopted in EGS4, and all the multiplicative processes are simulated.

The original EGS4 is coded in MORTRAN3, a FORTRAN precompiler language developed at SLAC, which is expanded into FORTRAN77. For this study, however, a FORTRAN version of the EGS4 was used. The size of the code is over 1900 lines in FORTRAN77 excluding comments. Some characteristics of the code structure of EGS4 may be summarized as follows:

5.1.1 General code structure
The general control flow of the major subroutine SHOWER is illustrated in Fig. 8. In EGS4, only one particle is processed at a time, and there are no explicit DO loops at all in the main body of the code.

5.1.2 Data structure
The push-down stack is used for storing the particle data with a pointer which points to the top of the stack. At the start of a simulation, the stack is loaded with one incident particle (usually an electron). Since only one particle can be processed at a time in the scalar processing, there is no need to develop a shower to its full extent. Instead, as the simulation proceeds, the newly created particles are placed in the stack, the particle with the lowest energy always being on the top. This is equivalent to tracing the shower tree in Fig. 7 toward the shortest branch, thus keeping the stack depth to the minimum. The push-down stack is obviously the optimal choice for scalar processing from the viewpoint of the memory size requirement.

5.1.3 Control scheme
The control scheme employed in EGS4 is very simple; the particle on the top of the stack is always to be processed in the next simulation step. When the stack is empty, simulation is completed.

5.2 Vectorization of cascade shower simulation code EGS4
As described in 5.1, the code structure of EGS4 is highly sequential and seems unvectorizable at first sight. The following subsections will describe how EGS4 code has been restructured to yield a high vector performance on the AMDAHL 1200 Vector Processor System [19].

5.2.1 Independence of particles and degree of parallelism
In a cascade shower simulation, once a particle is created, it is completely independent of other existing particles and can be processed in
any order. Therefore, if the shower is fully developed at the earliest possible stage, a very high degree of parallelism is expected. Furthermore, if a sufficient number of particles have been accumulated for one type of interaction, they can be efficiently processed in one pass in the vector mode. This observation leads to entirely different control scheme and data structure from those in the original scalar code as described in 5.1.

An experimental vector version, named EGS4-V, was developed along this line. Neither the physics models nor the sampling algorithms have been modified; only the order in which the particles are to be processed are different from the EGS4.

5.2.2 Global code structure of EGS4-V

Figure 9 illustrates the code structure of the subroutine SHOWER, the main part of EGS4-V. It consists of the dataflow control section and a multi-way jump to the slave subroutines. The dataflow control section monitors the particle data at each simulation step and initiates the execution of the next subroutine. The slave subroutines include the particle transport, the interactions and the garbage collections. Most of the slave subroutines are the vectorized version of the original subroutines, but some are newly defined for this vector version (Table 2).

5.2.3 Data structure

The EGS4-V uses the queues instead of the push-down stack in order to fully expose the parallelism in the particle data. Here, a queue means a collection of particles which are ready for the next step of computations. Queues are defined in the main memory as one-dimensional arrays, but the order of the elements is unimportant for this application. It is obvious that a shower can be most quickly developed by traversing the shower tree in Fig. 7 in the horizontal directions; the newly created particles are immediately stored in the queue.

There are two separate queues in EGS4-V, namely, E-Queue for electrons/positrons, and P-Queue for photons. This is not the only choice; one common queue may be used for all types of particles. A more detailed analysis of these two approaches is yet to be conducted. The event status ID is also assigned to each particle in the queue besides the physical variables. The said ID can take one of the 14 values as shown in the first column of Table 2. 14 event status counters are provided, each keeping track of the number of particles in the corresponding status.
5.2.4 Control scheme

The dataflow control section in subroutine SHOWER serves as a global event monitor by constantly scanning the event status counters, so that the subroutine with the highest particle count is always to be executed in the next simulation step for the maximum vector efficiency.

5.2.5 Garbage collection

In any vectorized shower simulation code, many particles are created and/or discarded in each simulation step, and the queue can easily overflow. Therefore, it is necessary to reclaim the unused portion of the queues. This process is commonly called garbage collection. There are two methods for implementing the garbage collection. The first method is to compress the queue whenever the number of discarded particles in the queue exceeds a certain threshold. In this method, the newly created particles can be stored in the contiguous locations of the queue. The second method is to use a so-called source buffer which holds indices pointing to all the available locations in the queue. When particles are discarded, their indices are added to the source buffer, while when particles are newly created, their locations are provided from the source buffer. In this method, the queue is always accessed via indirect addressing.

In EGS4-V, the first method has been adopted for ease of debugging. A more detailed analysis of the two methods is yet to be performed. The dataflow control section checks the amount of garbage after executing the subroutines which involve discarded particles (i.e., ANNIH, ECUT, COMPT, PHOTO and PCUT).

5.3 Timing measurement of a sample problem

A sample problem has been run on AMDAHL 1200 VP System to measure the vector performance of EGS4-V against the original scalar EGS4. In this problem, 1 Gev electrons are injected into a lead block of infinite size, so that no boundary crossing takes place. The number of the cases (that is, the number of the incident electrons) has been varied from 10 to 200. The results of the timing measurement are shown in Fig. 10. Since the publication of the early results [11], the code has been improved, and an asymptotic vector vs. scalar performance ratio of 11.6 has been obtained for this measurement.

Figure 11 depicts an interesting dynamic behavior of the active vector length. The number of cases for this run is 100. The vector lengths have been averaged over 10 simulation steps in order to produce a more legible graph. In Fig. 11, the vector length varies considerably in spite of the
above-mentioned smoothing process. The vector length starts from 100, reaches the instantaneous peak of 2,113 (smoothed out and not shown in Fig. 11), and decreases with frequent spikes. The total number of simulation steps was 2720.

6. GEOMETRY HANDLING

When the geometrical structure is complex, a significant portion of the computation time may be spent in the geometrical computations, and the vectorization of the geometry routines becomes important. In a typical transport Monte Carlo simulation code, the geometry routines are called within the transport section of the code in order to determine the shortest distance of the two: the distance to the next interaction point and the distance to the nearest geometrical boundary (Fig. 4). It should also be noted that the "ray-tracing" problem in computer graphics is very similar to the Monte Carlo problem.

Some code systems such as MORSE-CG [20] and GEANT3 [21] are provided with the geometry packages with wide repertories of geometrical objects, which are the building blocks for constructing complex geometrical structures. In the case of EGS4, the geometry is handled by a subroutine named HOWFAR, which is to be defined by the users for specific applications. The users of EGS4 have compiled several frequently used macro geometry routines, such as parallel planes, concentric cylinders, spheres, cones etc. In EGS4, the MORSE-CG geometry package can also be embedded in HOWFAR routine.

The basic idea for vectorization, again, is to form a vector with particles which are located within the identical geometrical object. One of the problem with vectorizing the geometry routines is that the branching destinations (i.e. number of cases) may be arbitrarily large, depending on the complexity of the user-defined geometry. (In the case of the physics routines, on the other hand, the number of the interaction types are fixed to a relatively small number, say, 10 or less.) Furthermore, if the geometrical structure is highly nested, the tree data structure which describes the hierarchy of geometrical objects has to be traversed for "neighbor-search", in order to identify the regions where the particles are to be transported after boundary crossing.

Efforts to vectorize the geometry routines have been reported by several researchers [22-27]. Brown reported a speedup factor in excess of 10 with his RACER3D general geometry reactor analysis code run on Cyber 205 [22]. Cloth and Filges reported a speedup factor of 3.2 to 4.5 with their vectorized geometry program run on CRAY X-MP for the radiation transport
applications [23]. Only the geometry with the rotational symmetric surfaces of the second order were considered in this work. Youssef proposed a "generalized ray tracing" algorithm [24], and obtained a speedup factor of 20 on Cyber 205 for a test program which comprises $10 \times 10 \times 10$ packed cubes [25]. Dekeyser and Georgiopoulos have been developing a vectorized geometry code for GEANT3, in which the particles in one volume (region) are to be processed in the vector mode [26–27]. The author of this lecture note is also investigating the vectorized geometry routines for the EGS4–V code.

7. PARALLELIZATION OF TRANSPORT MONTE CARLO SIMULATION CODES

7.1 Issues in parallelizing transport Monte Carlo codes

Although the parallelization process may seem conceptually more natural and straightforward than the vectorization process for the transport Monte Carlo simulations, there are new issues in parallel programming at the same time. These issues are not solely confined to the Monte Carlo simulation, but of a more general nature. Some of them are addressed in the following.

7.1.1 Shared memory architecture vs. distributed memory architecture

When an application program is explicitly partitionable and the workload can be equally distributed across the processors, parallel processing on the distributed memory architecture is the obvious choice because of the predictable performance improvement and scalability with the increased number of processors. On the other hand, it is a real benefit to have a shared memory for any application where the workload has stochastic nature and cannot be predicted prior to the actual computations, since the common database (such as a stack) can be accessed by all the processors thus allowing runtime load-balancing. Also, large data base for the cross section tables and for the geometrical structures, which are common features in any production Monte Carlo codes, do not have to be replicated in the shared-memory architecture. Of course, there is always a shared resource conflicts such as the bus contention or the exclusive accesses to the memory, and the performance is not scalable as the number of processors is increased. Still yet, whether problems like the cascade shower simulation can be efficiently processed on any distributed memory architecture or not, is not clear at this moment.

7.1.2 Identification of global and private variables

In a shared-memory architecture, all the variables in the COMMON blocks must be carefully examined whether they should be shared among the processors (global variables), or privately copied for each processor
(private variables). Each COMMON block may contain both types of variables, in which case it has to be split into two blocks. This is a very time-consuming task, if done manually. Definitely, good software tools are needed in this area. Another issue to be noted is that the notion of the COMMON block for parallel processing is not well established, and some systems do not support both types of COMMON blocks, hence the portability problem.

7.1.3 Machine-dependent library functions or synchronization primitives for parallel programming

Even within the category of the parallel processing systems with the shared-memory architecture, each system has its own library functions, compiler directives or FORTRAN extensions to describe and/or control parallel processing. There is no standard in this area. This, again makes the porting of the parallel codes very difficult.

7.1.4 Parallel random number generation

Unless great care is taken that each particle uses the same sequence of random numbers in the parallel code as in the scalar code, results are not guaranteed to be the same. Worse yet, it is quite possible to construct a parallel code which does not produce the same results from run to run due to the effect of race conditions in obtaining random numbers. Frederickson et al. proposed the concept called Lehmer-tree, based on two sets of the linear congruential random number generators [28]. By adopting this concept and by storing a random seed with each particle, the same simulation results can be obtained regardless of the order in which the particles are processed and with any number of processors. This is a new area, and further research will be needed to establish algorithms for generating good parallel random numbers.

7.2 Reported research works

Several research works have been reported in the literature for parallel processing of the transport Monte Carlo simulations. Frederickson et al. have parallelized a transport Monte Carlo code on HEP-1 shared-memory parallel processing system, using the Lehmer-tree to guarantee the reproducibility [29]. Martin et al. have parallelized a photon transport code on various the shared-memory and the distributed-memory parallel processing systems (CRAY X-MP, IBM3090/400, NCUBE) [30]. The author's own experiences in parallelizing EGS4 on Sequent B21000 Parallel Processing System will be described in 7.3.
7.3 Parallelization of EGS4 code [31]

There are two basic approaches in parallelizing the EGS4 code: one is to parallelize the original scalar code in such a way as to process many independent particles in parallel (to be called fine-grain approach), and the other approach is to start with the vectorized version and either to process each loop in parallel (so called microtasking), or to process the independently executable vectorized subroutines in parallel (so called macrotasking, or large-grain approach).

7.3.1 Fine-grain parallel processing

In this approach, each processor fetches a particle from a shared stack and executes the scalar simulation code. The synchronization is done by locking and unlocking the stack pointer to the shared particle stack, thus allowing dynamic load balancing. The fine-grain parallel version of EGS4 has already been completed. The lehmer-tree technique is incorporated in this code. Fig. 12 illustrates the control flow of this code. A sample problem was run with one 50 Gev electron injected into a lead block of infinite size, and a parallel speedup factor of more than 25 was obtained with 29 Sequent B21000 processors. (Fig. 13)

7.3.2 Microtasking and Large-grain approaches

Two approaches are possible for parallelization of the vectorized EGS4 code. So far, the microtasking approach did not turn out to be attractive due to the complexity of the DO loop structure and a lack of a software tool at the time of this study. On the other hand, the large-grain approach is more promising since the code structure of the vectorized version of the EGS4 already incorporates independently executable subroutines, which can readily be exploited in the large-grain approach (Fig. 9). With the advent of the vector multi-processor systems, this approach seems to be the right one, and deserves further research.

8. CONCLUSIONS

This lecture note reviewed vectorization and parallelization techniques for the transport Monte Carlo simulations. The fundamental differences between scalar coding and vector coding, and also between the scalar coding and parallel coding have been addressed, based on the author's own experiences. It has been pointed out that the transport Monte Carlo codes inherently contain a very high degree of parallelism, and that they can be either vectorized or parallelized efficiently.
As for the vectorization of the transport Monte Carlo codes, typical speedup factors of 5-10 have been reported in the literature. It should also be noted that parallelism at the higher level becomes visible through the vectorization process, which is quite suitable for vector-parallel processing. There are active research works going on at CERN, Florida State University, University of Michigan and KFA as well as the author's own efforts, to vectorize the transport Monte Carlo codes for high energy physics applications. The vectorization techniques described in this note is not solely confined to the transport Monte Carlo simulation, but should be applicable to other seemingly unvectorizable problems. The vector data handling capabilities which are accessible from FORTRAN language, are the key factors for implementing vector codes.

Although parallelization of the Monte Carlo codes may be more straight-forward than vectorization, a lot more research and development should be made in programming environment in general, especially in compiler technology, in debugging tools, and in parallel software development tools which can provide useful information for efficient parallel programming. The necessity for a global scanning capability on the part of the compiler should be emphasized as the architectural trend moves toward various forms of parallel architecture, where the detection of so-called large granularity parallelism is required. Development of a fully automatic compiler may become impractical for such systems. Rather, user-friendly interactive tools incorporating the graphical representation of program flow seem to be the right approach. The non-determinism in parallel processing (such as the parallel random number generator) could be a real hazard in developing parallel programs.

Since supercomputing is an application-driven area, very close interactions between researchers in various applications and manufacturers of supercomputers will be crucial in order to cope with the ever-increasing demands for large scale scientific and engineering computations.

ACKNOWLEDGEMENT
The author would like to thank Dr. W. R. Nelson of Stanford Linear Accelerator Center for providing EGS4 code and for valuable discussions, Dr. R. G. Babb II for valuable discussions on the fine-grain and the large-grain parallel processing techniques, Amdahl Corporation for providing the computational resources for the development and timing measurements of the vector version of EGS4, and Sequent Computer Systems, Inc. for providing computational resources for the timing measurements of the fine-grain version of EGS4.

384
REFERENCES


Table 1. Type of Interactions Treated in EGS4

<table>
<thead>
<tr>
<th>Subroutine Name</th>
<th>Type of Interaction</th>
<th>Incident Particle</th>
<th>Secondary Particles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNIH</td>
<td>Annihilation</td>
<td>Positron</td>
<td>2 Photons</td>
</tr>
<tr>
<td>BHABHA</td>
<td>Bhabha Scattering</td>
<td>Positron</td>
<td>1 Electron</td>
</tr>
<tr>
<td>BREMS</td>
<td>Bremsstrahlung</td>
<td>Electron or Positron</td>
<td>1 Electron (Positron)</td>
</tr>
<tr>
<td>MOLLER</td>
<td>Moller Scattering</td>
<td>Electron</td>
<td>2 Electrons</td>
</tr>
<tr>
<td>MSCAT</td>
<td>Multiple Scattering</td>
<td>Electron or Positron</td>
<td>1 Electron (Positron)</td>
</tr>
<tr>
<td>COMPT</td>
<td>Compton Scattering</td>
<td>Photon</td>
<td>1 Photon</td>
</tr>
<tr>
<td>PAIR</td>
<td>Pair Production</td>
<td>Photon</td>
<td>1 Electron</td>
</tr>
<tr>
<td>PHOTO</td>
<td>Photo-electric Effect</td>
<td>Photon</td>
<td>1 Electron</td>
</tr>
</tbody>
</table>

Table 2 Subroutines in EGS4-V

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Subroutine Name</th>
<th>Description</th>
<th>Discarded Particle?</th>
<th>Energy Deposit?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ELECTR</td>
<td>Electron Transport &amp; Multiple Scattering</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>EGARB</td>
<td>E-Queue Garb. Coll.</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>ANNIH</td>
<td>Positron Annihilation</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td>BHABHA</td>
<td>Bhabha Scattering</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>5</td>
<td>BREMS</td>
<td>Bremsstrahlung</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>MOLLER</td>
<td>Moller Scattering</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>7</td>
<td>INTRCT</td>
<td>Interaction Sel.</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>8</td>
<td>ECUT</td>
<td>Electron Cut-off</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>9</td>
<td>PHOTON</td>
<td>Photon Transport &amp; Interaction Sel.</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>PGARB</td>
<td>P-Queue Garb. Coll.</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>11</td>
<td>COMPT</td>
<td>Compton Scattering</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>12</td>
<td>PAIR</td>
<td>Pair Production</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>13</td>
<td>PHOTO</td>
<td>Photo-electric Effect</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>14</td>
<td>PCUT</td>
<td>Photon Cut-off</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Fig. 1 An Example of Vector Architecture
(FUJITSU Vector Processor System VP-200E)

MASK VECTOR

M

0 1 0 0 1 1 0 1

VR1

VR2

VCP VR2, VR1, M (Compress)

VEX VR3, VR2, M (Expand)

VR3

B1 A2 B3 B4 A5 A6 B7 A8

Fig. 2 Vector Compress and Expand Operations
Fig. 3 Vector Gather and Scatter operations

Fig. 4 A simplified control flow of transport Monte Carlo code
Fig. 5 Feed-forward type IF test

Fig. 6 Feed-backward type IF test
Fig. 7 An electromagnetic cascade shower

Fig. 8 Control flow of scalar EGS4
(Subroutine SHOWER)
Fig. 9 Control flow of vectorized EGS4
(Subroutine SHOWER)
Fig. 10 Timing Measurement of vectorized EGS4 on AMDAHL 1200VP. Computation time vs. number of cases (1 Gev electrons into Pb).

Fig. 11 Typical dynamic behavior of vector length. Average of 10 vector lengths vs. simulation steps. (1 Gev electrons into Pb, 100 cases)
Fig.12 Control flow of fine-grain parallelized EGS4
(Subroutine SHOWER)
Fig. 13 Performance measurement of fine-grain parallelized EGS4 on Sequent B21000 Parallel Processing System. Speedup factor vs. number of CPU's (A 50 Gev electron into Pb).
ABSTRACT
Since 1979, the Fortran standardization committee, X3J3, has been labouring over a draft for the next version of the standard. Its initial intention of publishing this draft in 1982 was hopelessly optimistic, and in fact it was published only in 1987. A number of fundamental issues have been thrown up over the past three years, such that it is now clear that unanimity can never be achieved: efficiency versus functionality; safety versus obsolescence; small and simple or big and powerful? This paper reviews the current state and content of Fortran 8x, and attempts to bring out some of the controversial issues surrounding its development.

_FORTRAN is not a flower, but a weed. It is hardy, occasionally blooms, and grows in every computer._
_A. Perlis_

1. LANGUAGE EVOLUTION
The hardiness of the Fortran 'weed' is now an accepted, if remarkable, aspect of scientific and numerical data processing. Various reasons, or excuses, for its vigour have been put forward—the magnitude of the existing investment, its ease of use, and the efficiency of its implementations—but one which is sometimes forgotten is the fact that the language itself has evolved since it was first introduced in 1957. This oversight is most evident in the writings of its detractors, who often unfairly compare their own favourite language with code written in abysmal style in Fortran 66, conveniently ignoring the success of Fortran 77 which, given some self-imposed discipline [1], can be employed to write programs of a very high standard.

This evolution has its formal side. The language has twice been standardized in the framework of ANSI and ISO, in 1966 and 1978. Following the publication of the second standard, the technical committee responsible for the work, X3J3, re-formed to begin work on a third standard, with the intention of having it ready in 1982. This time scale was hopelessly optimistic, the draft being published only in 1987.

What are the justifications for continuing to revise the definition of the Fortran language? One is to modernize it in response to the developments in language design which have been exploited in new languages, particularly PASCAL and ADA. Here X3J3 has the advantage of hindsight, and can avoid the pitfalls of adding trendy features like the already outmoded DO...WHILE, whilst drawing on the obvious benefits of concepts like data hiding. In the same vein is the need to eliminate the dangers of storage association, to abolish the rigidity of the outdated source form, and to improve further on the regularity and portability of the language.

However, taken together, these would constitute only a ragbag of items perhaps better obtained by switching to other languages. The real strength of the new standard will be its incorporation of powerful array processing features and of derived-data types, allowing users to access vector processor hardware using a convenient notation, and to define and manipulate objects of their own design. Unfortunately for those of us who would like to see a new standard published by the end of
this decade, there are still many possibly insuperable obstacles to be overcome. Formally, X3J3 is currently at milestone eight of the eighteen milestones which have to be passed before a new standard receives full international recognition, and there are powerful voices stating that the advances it has made have gone too far, and that there should be some further retrenchment before it is acceptable to them. Such retrenchment would, of course, be unacceptable to others. A ballot of X3J3 held in April 1986, whose result was 16 to 19 against the draft of that date, resulted in a first slimming down, including the loss of BIT data type, in an attempt to reach a compromise between radicals who want a large range of new features and conservatives with more modest goals and who emphasize the importance of long-term backwards compatibility. A second ballot, 29 to 7, showed that better agreement had been reached, but that no true consensus had been achieved, given that two of the dissenting votes are those of IBM and DEC claiming to act as representatives of their users.

This dissension within the committee could have been regarded as healthy as long as it led to a constructive debate on its objectives resulting in a final resolution of the difficulties. However, some of the issues are so difficult, that reconciliation is now seen as unobtainable:
- Should Fortran 8x be innovative, or merely standardize existing practice?
- Should the language be small and simple, or big and powerful? Is the present proposal too big, making it impossible to fit into the small machines of 1990, impossible to be implemented by small software houses, and impossible to be understood by non-professionals (the bulk of Fortran' users)?
- Are users prepared to drop existing features, given 10 to 20 years' notice, or must all existing code work for ever?
- Are subsets of the language useful, or an impediment to portability?
- Do users want a safe language, or one which permits them to write the tricky programs more often associated with assembly languages?
- Are the existing proposals difficult and inefficient to implement? Does that matter if users thereby have an easier life?
- Will the heap storage mechanism necessary for dynamic array allocation impair efficiency, as happened with PL/1?
- Will the presence of new features cause existing ones to be implemented less efficiently?

In spite of all these points of disagreement, X3J3 and its parent committee X3 finally decided in the summer of 1987 that the draft of that time should be submitted to a formal period of public comment, at both the international and American levels. This period ended in February 1988, and showed that the public was on the whole unreceptive to many of the novel ideas contained in the document. By a margin of about two to one (interpretations vary) the 400 letters received seemed to want a simpler, less innovative standard, truer to the 'spirit of Fortran'. X3J3 has examined various proposals to simplify the language as a response to this negative reaction but, hamstrung by a two-thirds voting rule, is currently unable to agree on any single proposal. It might require direction from ISO to resolve this impasse, which could conceivably result in the work on Fortran actually being removed from ANSI and hence X3J3, and being taken over by another body*).

The development of the published document has not followed a straight path, and some items (for instance BIT data type and the significance of blanks) have been in and out several times. This has meant that the public has had some difficulty in following the committee's work. This present paper is an attempt to give a feel for Fortran 8x; fuller details are given in [2]. It reflects one possible outcome of the present debate, but without describing the form in which pointers are likely to be added.

*) At its meeting in September 1988, the ISO body (JTC1/SC22/WG5) defined its own version of Fortran 8x, which it renamed Fortran 88, and gave X3J3 five months to produce that as a new draft standard, or else!
2. BACKWARDS COMPATIBILITY

The procedures under which X3J3 works require that a period of notice be given before any existing feature is removed from the language. This means, in practice, a minimum period of one revision cycle, which for Fortran means a decade or so. The need to remove features is evident: if the only action of the Standards Committee is to add new features, the language will become grotesquely large, with many overlapping and redundant items. The solution likely to be adopted by X3J3 is to publish as an Appendix to the standard a set of two lists showing which items have been removed or are candidates for eventual removal.

The first list contains the Deleted Features, those which in the previous standard were listed as Obsolescent Features, and have now been removed. The list of Deleted Features is empty for Fortran 8x which contains the whole of Fortran 77.

The second list contains the Obsolescent Features, those considered to be redundant and little used, and which should be removed in the next revision (although that is not binding on a future committee). The Obsolescent Features are

- Arithmetic-IF
- Shared DO termination
- Alternate RETURN
- PAUSE
- Real and double precision DO-variables
- DO termination not on CONTINUE (or ENDDO)
- Branch to ENDIF from outside block
- ASSIGN and assigned GOTO

3. MAIN NEW FEATURES OF FORTRAN 8x

3.1 Source form

The new source form allows free form source input, without regard to columns. Comments may be in-line, following an exclamation mark (!), and lines which are to be continued bear a trailing ampersand (&). The character set is extended to include the full ASCII set, including lower-case letters (which in Fortran syntax are interpreted as upper case). The underscore character is accepted as part of a name, which may contain up to 31 characters. Thus, one may write a snippet of code such as

```fortran
SUBROUTINE CROSS_PRODUCT (x, y, z) ! Z = X*Y : 
z1 = x(2) * y(3) - & x(3) * y(2)
```

3.2 Alternative style of relational operators

In response to user demand, the deliberate redundancy of an alternative set of relational operators is introduced. They are

- `<` for `.LT.`
- `<=` for `.LE.`
- `=` for `.EQ.`
- `>` for `.GT.`
- `>=` for `.GE.`
- `<>` for `.NE.`

enabling statements such as

```fortran
IF (x < y .AND. z**2 >= radius_squared) THEN
```

to be written.

398
3.3 Specification of variables

The existing form of type declaration, as shown by

\[
\text{REAL } a(5,5), b(5,5)
\]

is extended to allow all the attributes of the variables concerned to be declared in a single statement. For instance, the statement

\[
\text{REAL, ARRAY}(25), \text{PARAMETER} :: a = [25*0.], b = [25*1.]
\]

declares the two objects \( a \) and \( b \) to have the attributes of being arrays of named constants, whose values are specified by the array constructors (see Subsection 3.9) following the equals signs. Many other attributes may also be specified, in particular real and complex variables may be specified to have a defined minimum precision and/or exponent range, as shown in

\[
\text{COMPLEX(PRECISION = 12, EXPONENT\_RANGE = 100), ARRAY}(10) &
\]

\[
:: \text{current}
\]

where the variable \( \text{current} \) is specified to have a minimum precision of 12 decimal digits and an exponent range of at least \( 10^{\pm 100} \). This facility is of great benefit in writing portable numerical software. Where constants corresponding to a given range and precision are required, a corresponding exponent letter must be chosen, for example

\[
\text{EXPONENT LETTER (12,100) C}
\]

\[
:: \text{current (3) = 15.6C57}
\]

At the time of writing, it seems likely that this facility will be simplified to correspond more closely to the actual, underlying hardware precision.

For those who wish to define explicitly the type of all variables, the statement

\[
\text{IMPLICIT NONE}
\]

turns off the usual implicit typing rules.

3.4 CASE construct

The CASE construct allows the execution of one block of code, selected from several, depending on the value of an integer, logical, or character expression. An example is

\[
\text{SELECT CASE}(3*1-j)
\]

\[
\text{CASE}(0) \quad ! \text{for 0}
\]

\[
:: \quad ! \text{executable code}
\]

\[
\text{CASE}(2,4:8) \quad ! \text{for 2, 4 to 8}
\]

\[
:: \quad ! \text{executable code}
\]

\[
\text{CASE DEFAULT} \quad ! \text{for all other values}
\]

\[
:: \quad ! \text{executable code}
\]

\[
\text{END SELECT}
\]

The default clause is optional; one of the clauses \textit{must} be executed.
3.5 Loop construct

A new loop construct is introduced whose syntax is combined with that of the old form of the DO-loop. In a simplified form it is

\[
\text{[name:]} \text{ DO ([control])} \\
\text{block of statements} \\
\text{END DO [name]}
\]

(where square brackets indicate optional items). The control parameter, if omitted, implies an endless loop; if present, it may have one of two forms:

\[
i = \text{intexp1, intexp2 [, intexp3]}
\]

or

\[
\text{intexp4 TIMES}
\]

The optional name may be used in conjunction with CYCLE and EXIT statements to specify which loop in a set of nested loops is to begin a new iteration or which is to be terminated, respectively.

3.6 Program units

An enhanced form of the call to a procedure allows keyword and optional arguments, with \textit{intent} attributes. For instance, a subroutine beginning

\[
\text{SUBROUTINE solve (a, b, n)} \\
\text{OPTIONAL, INTENT (IN) :: b}
\]

might be called as

\[
\text{CALL solve (n = i, a = x)}
\]

where two arguments are specified in keyword (rather than positional) form, and the third, which may not be redefined within the scope of the subroutine, is not given in this call. The mechanism underlying this form of call requires an \textit{interface block} with the relevant argument information to be specified.

Procedures may be specified to be recursive, as in

\[
\text{RECURSIVE FUNCTION factorial(x)}
\]

The old form of the statement function is generalized to an \textit{internal procedure} which allows more than one statement of code, permits variables to be shared with the host procedure, and contains a mechanism for overloading operators and assignment for derived-data types. This we shall return to in Subsection 3.14.
3.7 Extensions to CHARACTER data type

A number of extensions to the existing CHARACTER data type permit the use of strings of zero length:

\[ a = "" \]

and the assignment of overlapping substrings:

\[ a(5) = a(3:7) \]

and introduce new intrinsic functions, such as TRIM, to remove the trailing blanks in a string. Some intrinsics, including INDEX, may operate, optionally, in the reverse sense.

3.8 Input/Output

The work in the area of input/output has been mainly on extensions to support the new data types, an increase in the number of attributes which an OPEN statement may specify, for instance to position a file or to specify the actions allowed on it, and to add a NAMELIST feature, illustrated by

\[
\text{NAMELIST/list/a, i, x} \\
\text{READ(unit, NML = list)}
\]

which would expect an input record of the form

\[
&\text{LIST} \quad x = 4.3, \quad A = 1.E20, \quad I = -4
\]

3.9 Array processing

The introduction of array processing features is one of the most important new aspects of the language. The reasons are threefold: an array notation defined in the language simplifies the syntax; new features extend the power of the language to manipulate arrays; and the concise syntax makes the presence of array processing obvious to compilers which, especially on vector processors, are able to optimize object code better.

An array is defined to have a shape given by its number of dimensions, or rank, and the extent of each one. Two arrays are conformable if they have the same shape. The operations, assignments and intrinsic functions are extended to apply to whole arrays on an element-by-element basis, provided that when more than one array is involved they are all conformable. When one of the variables involved is a scalar rather than an array, its value is distributed as necessary. Thus we may write

\[
\text{REAL, ARRAY(5, 20) :: x, y} \\
\text{REAL, ARRAY(-2:2, 20) :: z} \\
\text{z = 4.0 * y * SQRT(x)}
\]

In this example we may wish to include a protection against an attempt to extract a negative square root. This facility is provided by the WHERE construct:
WHERE (x > = 0.)
    z = 4.0 * y * SQRT(x)
ELSEWHERE
    z = 0.
END WHERE

which tests x on an element-by-element basis.

A means is provided to select sections through arrays. Such sections are themselves array-valued objects, and may thus be used wherever an array may be used, in particular as an actual argument in a procedure call. Array sections are selected using a triplet notation. For an array

\[
\text{REAL, ARRAY}(-4:0, 7) :: a
\]

\(a(-3,:)\) selects the whole of the second row, and \(a(0:-4:-2, 1:7:2)\) selects in reverse order every second element of every second column.

Just as variables may be array-valued, so may constants. It is possible to define a rank-one array-valued constant as in

\[
[1, 1, 2, 3, 5, 8]
\]

and to reshape it to any desired form:

\[
\text{REAL, ARRAY}(2, 3) :: a
\]
\[a = \text{RESHAPE}([2, 3], [1:6])\]

where the first argument to the intrinsic function defines the shape of the result, and the second defines an array of the first six natural numbers.

3.10 Dynamic storage

Fortran 8x provides four separate mechanisms for accessing storage dynamically. The first is unlikely to survive the present carnage, and is not described here.

The second mechanism is via the ALLOCATE and DEALLOCATE statements which, as their names imply, are used to obtain and return the actual storage required for an array whose type, rank, name, and allocatable attribute have been previously declared in the procedure:

\[
\text{REAL, ARRAY}(:), \text{ALLOCATABLE} :: x
\]

\[\text{ALLOCATE}(x(n, m))! n \text{ and } m \text{ are integer expressions}\]

\[x(j) = q\]
\[\text{CALL sub(x)}\]

\[\text{DEALLOCATE} (x)\]

Deallocation occurs by default, unless the array has the SAVE attribute, whenever a RETURN or END statement in the same procedure is executed. The fact that allocation and deallocation occur in random order implies an underlying heap storage mechanism.
The third mechanism, useful for local arrays with variable dimensions, is the *automatic* array:

```
SUBROUTINE sub(i, j, k)
REAL, ARRAY(i, j, k) :: x ! bounds from dummy arguments
```

whose actual storage space is provided (on a stack) when the procedure is called.

Finally, we have the *assumed-shape* array, whose storage is defined in a calling procedure, and for which only a type, rank, and name are supplied:

```
SUBROUTINE sub(a)
REAL, ARRAY(:,:,:) :: a
```

Various enquiry functions may be used to determine the actual bounds of the array:

```
DO (i = LBOUND(a, 1), UBOUND(a, 1))
  DO (j = LBOUND(a, 2), UBOUND(a, 2))
    DO (k = LBOUND(a, 3), UBOUND(a, 3))
```

where LBOUND and UBOUND give the lower and upper bounds of a specified dimension, respectively.

### 3.11 Intrinsic procedures

Fortran 8x defines about 100 intrinsic procedures. Many of these are intended for use in conjunction with arrays for the purposes of reduction (e.g. SUM), inquiry (e.g. RANK), construction (e.g. SPREAD), manipulation (e.g. TRANSPOSE), and location (e.g. MAXLOC). Others allow the attributes of the working environment to be determined (e.g. the smallest and largest positive real and integer values), and access to the system and real-time clocks is provided. A random number subroutine provides a portable interface to a machine-dependent sequence, and a transfer function allows the contents of a defined area of physical storage to be transferred to another area without type conversion occurring.

The MIL-STD 1753 bit intrinsic procedures are likely to be included in a final version [3].

### 3.12 Derived-data types

Fortran has hitherto lacked the possibility of building user-defined data types. This will be possible in Fortran 8x, using a syntax illustrated by the example

```
TYPE staff_member
  CHARACTER(LEN = 20)::first_name, last_name
  INTEGER::id, department
END TYPE
```

which defines a structure which may be used to describe an employee in a company. An aggregate can be defined as

```
TYPE(staff_member), ARRAY(1000)::staff
```

defining 1000 such structures to represent the whole staff. Individual staff members may be
referenced as, for example, staff(no), and a given field of a structure as staff(no)%first_name, for the first name of a particular staff member. More elaborate data types may be constructed using the ability to nest definitions as in

```fortran
TYPE company
    CHARACTER(LEN = 20)::name
    TYPE(staff_member), ARRAY(1000)::staff
END TYPE

: TYPE(company), ARRAY(20)::companies
```

to build a structure to define companies.

### 3.13 Data abstraction

It is possible to define a derived-data type, and operations on that data type may be defined in an internal procedure. These two features may be combined into a module which can be propagated through a whole program to provide a new level of data abstraction. As an example we may take an extension to Fortran’s intrinsic CHARACTER data type whose definition is of a fixed and pre-determined length. A user-defined derived-data type, on the other hand, may define a set of modules to provide the functionality of a variable length character type, which we shall call *string*. The module for the type definition might be

```fortran
MODULE string_type
    TYPE string(maxlen)
        INTEGER::length
        CHARACTER(LEN = maxlen)::string_data
    END TYPE String
END MODULE String_type
```

With

```fortran
USE string_type
:
    TYPE(string(60)), ARRAY(10)::cord
```

we define an array of 10 elements of maximum length 60. An actual element can be set by

```fortran
cord(3) = 'ABCD'
```

but this implies a re-definition, or *overloading*, of the assignment operator to define correctly both fields of the element. This can be achieved by the internal procedure

```fortran
SUBROUTINE c_to_s_assign(s,c) ASSIGNMENT
    TYPE (string(*)) :: s
    CHARACTER(LEN = *)::c
    s%string_data = c
    s%length = LEN(c)
END SUBROUTINE c_to_s_assign
```
which can be included in the module, together with other valid functions such as concatenation, length extraction, etc., to allow the user-defined string-data type to be imported into any program unit where it may be required, in a uniformly consistent fashion.

4. CONCLUSION

This paper has tried to set out the background to the current revision of the Fortran language standard, and to give an overview of its principal new features.

Any standard is a compromise between conflicting points of view, as indicated in the first section. On the other hand, the presence of a widely accepted and implemented standard provides the only practical means to write and to use portable programs, and the evolution of that standard is the only means to escape from the temptation to use vendors’ extensions or private dialects and languages. It is my opinion that although the current proposal does not contain everything which we would like to see, nevertheless, its final acceptance and implementation are the only way to avoid a future of anarchy and chaos. It is to be hoped that reason will prevail, and that the present serious disagreements can finally be resolved to the satisfaction of the largest possible section of the user community.

*   *   *

REFERENCES

ACCELERATOR CONTROLS

Fabien Perriollat

CERN, Geneva, Switzerland

Abstract

Controls systems are getting more and more important for the running of High Energy Physics Laboratories, due to the ever increasing complexity of the different accelerators used. The problems of accelerator control are explained from the computer science point of view. The major domains of the control systems are studied with the various software techniques used. Some fields of the current development and research will be reviewed. Moreover, similarities exist between the new generation of large High Energy Physics experiments and accelerator control. Also the vast experience in software for accelerator control is more and more useful for the physics experimentalists.

1. Introduction

1.1 The accelerator controls problem

Various types of accelerators exist. One of the simplest is the Electrostatic Van der Graaff and one of the most complex is the CERN cascade of 12 accelerators running in multi-user mode. The classes of accelerators are: DC machines (electrostatic, pseudo DC (cyclotron), pulsed machines (Linac/Synchrotron), mixed (circular collider)).

1.2 The components of accelerators

The general (machine) components are the magnets, cryogenic supplies, RF plants, vacuum, instrumentation and beam diagnostic, ions or leptons sources, safety and services (cooling, ventilation, etc). Other components, from the controls point of view, are power converters (power supplies), timing, synchronization, sequencing, mechanical movement, measurement instruments, and simple ON-OFF elements.

1.3 The function (or the duty) of controls

The main function is to maintain the running condition of the equipment. Controls are also interfacing with the operators and users of the accelerator. The data acquisition and data recording is done by controls. The different types of actions are the setting up, running down, modifications of working conditions, help in survey and trouble-shooting, studies and improvement of accelerators, recovery after fault repair, adjustment and optimization of working points and production.
1.4 The domain of the controls

The controls domain reaches from the operators to the accelerator equipment through various stages, where the separation line between the equipment and the controls is not always very well defined. Commercial equipment including its own control is used as well as locally embedded data treatment and the merging of analog and digital electronics.

1.5 The scope of the lecture

Not included in this lecture is the control theory about close loop control and optimum control nor the finite state machine theory or the signal treatment.

On the other hand, the controls problems, with various view points are treated as well as the computer science techniques used to solve controls problems, the trends of evolution of the control techniques and future possibilities. There will also be a short explanation of the hardware interface with equipment. The lecture was mainly architecture and software oriented.

2. Accelerators and their controls

2.1 Classes of equipment

The classes of equipment are the simple ON/OFF acquisition or the acquisition and control, (e.g. state of relay; the complex digital equipment, the analog (or quasi analog) systems, the multi—states and hybrid (e.g. power converter); the timing system and instrumentation (a mixture of timing, analog and digital control with sequencing).

2.2 Actions to be performed

These actions are the setting up and running in, the running down and stopping, the adjustment and optimization of the working condition, the survey of the system, recovery after correction of abnormal state, the saving and restoring of the working point, preparation of future runs and operation, studies of accelerator physics, trouble-shooting of hardware and software, modifications and adaptation of existing systems and the introduction of new controls facilities with qualifications and reception tests.

2.3 Users of control systems

The control system is the necessary path between any user and the accelerator equipment. The operation of an accelerator complex requires many specialists in many different disciplines. The specialist wants to see the full details of his operation and the rest must be hidden.
2.3.1 The Equipment Specialist

He constructs and tests independent parts of the equipment and also assembles and integrates these parts. For this, he needs a stand-alone operation, full control of all details of his equipment, an easy way of accessing his control tool set, and widely distributed access points (close to the equipment).

2.3.2 The Accelerator Physicist

He takes the various hardware parts and connects them to form an operation. He is mainly concerned with the interaction of the various facilities and is an intensive user of the instrumentation and beam diagnostic facilities. He explores frontiers of the working condition, requiring very flexible and powerful control facilities. He also requires new controls facilities, as on-line modelling, expert systems, off-line simulation and operation preparation.

2.3.3 The Accelerator Operator

His task is to keep the accelerator complex running for long periods of time. He must also execute the production program as scheduled. The operator requests from the control system convenient ways of monitoring the overall operation. He must be able to diagnose and correct faults and to change parameters and operations as required. His must log information on the current working condition of the accelerator.

The control room and the human interface of the controls system are the working environment of the operators. The ergonomics of the operator desk and the control room is very important.
2.3.4 The Physics Experimentalist

The physics experimentalist requires information about the state of the accelerators and their beams, and if there is no conflict with other users or with the accelerator itself, he must to be able to control and correct the elements in the beam lines.

2.3.5 The Maintenance and Repair Team

Generally, this team is a mixture of equipment specialists and controls specialists. They want to run powerful tests quickly and use diagnostic facilities. They also use on-line documentation and help procedures. They must have a good overview of the controls system and also good relations with the operators.

2.3.6 The Control Developer

He uses tools for introducing new controls facilities and also for tests, debugging, and validation without disturbing the accelerators' running conditions. The controls developer wants good management facilities for the software components.

3. General control architecture

The use of a controls system model is an efficient way of reducing the complexity of the system. It also gives a good overview of the problem concerned and it is a starting point for a formal description. Starting discussions between partners (clients and developers) can also be done very effectively with this model.

Two models are shown here: the model of the 3 functional layers and the data flow model.

3.1 The model of 3 function layers

The model of the 3 function layers provides the connecting path between the operator and the equipment, and also the environment for running autonomous activities. It offers connection to the other digital data processing facilities, e.g. office workstations, central D.P.s, etc.

3.1.1 The top level: operator/human interface

The operator is the master of the control activities. He needs good information on the states and actions taken by automatic process. The operators has multiple view points of the process for correlation of process events and for better understanding of process phenomena.

The properties of the layer of operator interface are:

* pure asynchronous processing (interaction of operators);
• user interface must be very attractive, user-friendly and easy to learn;
• high resilience to operator mistakes and random actions;
• capable to present very large quantities of data in a wide diversity of formats (texts, graphics, synoptics, etc.).

3.1.2 The service layer

It provides services not dedicated to a special process and offers the general data processing service. This layer is the environment for autonomous processing and tasks.

The properties of the service layer are:
• fast response time to a query or request;
• high level reliability and availability;
• transparency for the user (operators) as much as possible;
• easy to update, maintain and service by control operation and developer teams.
3.1.3 The equipment layer

This layer provides the connectivity to the process equipment and reacts to equipment stimulus. It does controls according to the equipment time constrains and coordinates the various functions which take part in the controls of the equipment.

The properties of the equipment layer are:

- hard real-time constraints especially for pulsed accelerators;
- the request of very high level of parallelism and distribution in data processing;
- provision of data reduction and data formatting (according to the equipment specification).

3.2 The data flow model

This model consists of three elements, the data sources, the data sink and the data lifetime.

3.2.1 Data sources

The data sources are the equipment acquisition (medium rate), the beam instrumentation (intensive data source), the operators (slow rate), and in the central service: the main coordination (medium rate) and the data base and storage (medium to high flux).
3.2.2 Data sink

The data sink is composed of equipment with medium rate range: PPM equipment (pulse-to-pulse modulation), and other ones with slow rate: for DC machine or none PPM equipment. The operator presentation has a very high flux of data. The central service is a very intensive data sink during some special tasks.

3.2.3 Data lifetime

The data lifetime of the equipment is medium to long (up to 1 run). The operator presentation needs a very short time and is always volatile, while the central service has a very long lifetime (up to the lifetime of control systems). The central service always makes a hard copy and backup—up copies.

4. Equipment access (front-end processing)

The complexity of this layer depends very much on the “dynamic” of the accelerator.

There are very drastic differences between

- the DC or quasi DC machines;
- the accelerators with intensive PPM mode;
- the multi-path beam transfer line.

One of the functions of this layer is to relax the real-time constraints. Furthermore, the merging between synchronous and asynchronous actions, maintaining the running conditions (especially in PPM mode), synchronizing the control actions as well as running the highly repetitive process are some of the more important functions. The morphology can be described as a high level of parallelism in processing with a very time-sensitive process.

4.1 Visibility from the higher layer

The goal is to provide a uniform access method for the equipment. The benefits are a uniform and homogeneous documentation, easy writing of data-driven general programs, simplification of the network function needed, and a well defined and stable interface between the layers.

4.1.1 Possible solutions

One of the solutions is the data base approach, which will have data base access to save and retrieve data records. It will also have functions acting on these records to execute predefined actions on the equipment. Another solution is the object-oriented approach, giving direct access to the equipment by its name with an action key (selector).
4.1.1.1 Data base approach

This approach can easily be provided, especially for small and simple systems. All data of an equipment will be directly accessible and special cases can easily be implemented. The process for archiving/retrieving and starting-up after an emergency is simple and efficient.

There are also drawbacks in this approach. The structure of the data record is imported in application programs and there is a large number of access functions to the various record elements. This approach also allows any half-baked and "spaghetti" type of application programs. The long-term maintenance of application software can be very difficult and access to the data base can be a bottleneck of the system.

4.1.1.2 Object-oriented approach

The benefits of this approach are the very well controlled access to the equipment and the very uniform general services provided by inheritance mechanism. This approach allows any number (in principle) of abstractions with data hiding. The application programs are not sensitive to the implementation of the equipment access method (easier maintenance). The approach is very well adapted to distributed systems.

The drawbacks are the difficulty for general services like archives, retrieving of data, etc. A special property for a special case must be implemented in the object. This approach requires a new way of thinking, especially for the "Fortran" programmers.

4.2 Real-time processing

The goal is to handle the Real-time problem of controls. The benefits are that this real-time processing releases other program layers from real-time constraints, and the time sensitive programs are concentrated around the equipment itself. It also allows reduction of the number of various mechanisms to solve the RT problem.

The method to build this processing consists of reducing the complexity of the problems by analysis and modification of external specifications and also minimizes the number of significant events issued by the equipment. The method requests also to define standard behaviour of the various classes of equipment (standard control protocol of equipment) and to use a unique repetitive running mode.

4.2.1 Possible solutions

One solution consists of the model of the 3 synchronous actors, for the preparation task, the control action (send data to the equipment), the acquisition process (retrieve results of measurements and status of equipment). These 3 actors are executed in a synchronous and recursive manner (the basic period concept).

4.2.1.1 Preparation task

The preparation task prepares all the data structures and values for the coming controls and acquisitions tasks. At the time of the significant event, which triggers the action, all the external conditions and parameter values for the basic period must be defined and no more changes can be executed.
later on (for the basic period). For this task, the RT’ constraint is normally not too strong (time window for this activity is not too tight).

4.2.1.2 Control task

The control task is sending the raw data (built by the preparation task) to the equipment. The time window for this action can be very narrow. Errors or problems to access the equipment must be reported. In special cases, more than one task can do the job, especially when more than one time window is needed to execute control of the equipment during the basic period.

4.2.1.3 Acquisition task

It reads the equipment data from the equipment according to data structure provided by the preparation task. This task also transforms raw acquired data into equipment data with possible data reduction and sends this data to the equipment object. When all data are available, it sends events to data consumer tasks. For instrumentation, the quantity of data to be acquired can be large.

4.3 Data source processor

The goal is to provide a regular data source mechanism (during the schedule of the basic period) to various end users in the upper layers. This processor uses an automatic mechanism for acquiring data and application programs driven by pure data flow.

4.3.1 Possible solution

One solution is to send systematically all acquired data. This is a very simple mechanism but only feasible for small or even very small control systems. Another solution is the subscription mechanism, i.e. end user programs (data consumer programs) take out a subscription to a defined data acquisition list. This solution uses difficult list management mechanisms and a tricky process in case of abnormal ending of user programs. It needs an evaluation of the peak condition case and requests dynamic routing and broadcasting process from the medium layer.

4.3.2 Difficulties of the equipment layer

These difficulties are the result of the large distribution (large numbers) of processors (keeping the hardware cost to a reasonable limit and minimizing the hardware overhead). There is also a high degree of parallelism in processing and stringent time dependence (small time window) as well as a very high level of reliability requested. The reliability of control systems depends very much on this layer. As a consequence, a good selection of building blocks is necessary for both hardware and software as well as a good and efficient RT environment. A powerful software and system debugging is requested with RT correlations, parallelism and remote debugging.

Some maintenance difficulties exist. The modification and update of running software request a software quality and certification (need of simulation environment for software qualification) and trouble-shooting tools, which can be very tricky. Trouble shooting and error reporting must be taken seriously right from the beginning of the design.
5. Service layer

The functions of this layer can be well distributed around the other layers according to the computer network properties. The evolution for the implementation of this layer may be very drastic in future, with the new computer technology and networking facilities.

The two main classes of functions are (i) general services to the control activities, (ii) functions which are related to the topology of the control system.

5.1 General services

These are services which can be requested by any part of the control system. They consist of a database management service, archive and history service, paper output (printer and plotter services) logging, network bridge to other LAN of the laboratory or to WAN, number crunching facility, program librarian and files server.
5.2 Functions of the in-between layer

These are services or functions needed on the path of data between the front-end and the operator interface. They are alarms and survey processing, error reporting and tracing, processing coordination, general manager of controls activity, data concentration and data distribution/broadcasting, network management and data server.

5.3 Major components of the service layer

5.3.1 Data base

Ideally, it will supply data to all data driven programs, in conjunction with the 2 other sources of data: equipment and operators. Its properties are to fulfill the RT constraints, to respond in a predefined and "short" time, and to provide uniform and efficient access methods. Data must be secured and data integrity guaranteed for the reliability of the control system.

The main classes of data structures are the dictionary tables (name to record), the lists (sequence of records with links), the indexes (to flat files or flat collections of data objects).

5.3.1.1 Possible implementation

The RT data base service is used for implementation. The RT data base can easily be derived from one central data base (off-line data base) for the stable data. The updating (plus pretty printing, backup...) is well provided for by the central DBM. There is a general service for multi-viewpoint of RT data base with a uniform access method. All data structure is easy to manage and maintain.

The ad-hoc solution for specific domains gives more flexibility for the specific problems of the domain. The implementation for the first domain is fast but the data is difficult to manage and maintain. No general service exists. The access method to the data is domain dependent.

Experience has shown that the RT data base has decisive advantages for large control systems.

5.3.2 Archive and history service

This is an intensive user of the RT data base. It must provide service for archiving and retrieving typical accelerator working status, for deferred (off-line) data analysis, setting-up of the machine from the power-off state, and for off-line preparation of the future machine state.

5.3.3 Alarm system and survey processors

These systems do an autonomous monitoring of the equipment. They inform the operators of abnormal status, and issue warning information and global beam monitoring data. Major properties requested by the users are reliability of alarms information, very high availability of the alarm system, fast response time (from an error in the equipment to the message in front of the operator), simple interaction facility, significant messages (not encoded). Through these properties the operators will reach a high level of confidence in working with the alarm system. If this high level of confidence is not reached, the system will never be used afterwards.
The alarm system is a large user of the data base service. The data base provides easy modification according to equipment evolution, good maintenance and updating tools, attached correcting procedure, standard and well-defined, and homogeneous and significant messages.

5.3.3.1 Major components of the alarm system

The equipment monitoring process surveys accelerator equipment and controls system components and detects abnormal states. It also reports the detected faults to the alarm collecting central service. The alarm collecting service receives alarm messages from the monitoring process and possible other application packages. It maintains the equipment status list and prepares the display list for the operator consoles. It also acts according to the operator interactions, on the equipment or on the list.

The console presentation and interaction process displays the current accelerators status in a condensed form or in a detailed form with various levels of details. This process interacts with the operator for access to various levels of detailed information and for modification of the central equipment list (masking an equipment not in use, releasing a surveillance constraint).

5.3.3.2 Possible implementation

This can be distributed survey processing as well as centralized alarm collecting. Also used is distributed (on the operator console) presentation and interaction processing. Equipment access must provide elaborate and standard status information.

5.3.3.3 Problems

The end user (operators) must get confident very quickly with the system. This confidence must be maintained during the entire lifetime of the controls system. There will always be difficulties in detecting the abnormal state. This is not too difficult in finite state equipment but can be very difficult for analog values, for very dynamic or wave form signals. Other difficulties are how to define (and by whom) a reference state of equipment complex and how to cope with the unavoidable "very special equipment behaviour". Moreover, it is certainly not easy to work with the very large number of equipment and equipment classes and to provide enough standard, as well as simple and well-defined recovery mechanism.

5.3.3.4 Actions to be taken

It is important to integrate the problem of the alarm system from the beginning of the project, and to strengthen the equipment as much as possible (hardware and associated software) to follow standard behaviour and meaning for status information and simple recovery procedures. It is also necessary to use the finite state machine model for the equipment and to define early in the project the alarm messages, their real meaning, and standard procedure for recovery.
6. The operator interface layer

This layer provides the human interface to the accelerators through the control system. Its quality and its user-friendliness will largely contribute to the success of the control system. The office and home computers, easy to use, low in cost and to be learned quickly, represent a big challenge to control systems and their developers. The fast evolution of the technology (hardware and software) in the fields of graphic workstations, with graphic and interaction software tools and standards, are a real opportunity to have an efficient and attractive operator interface. The user does not want to be cut down by the controls system (= operator interface) in his phantasy for running the accelerators.

6.1 Function of this layer

In order to provide an efficient human interface, the layer must have several functions. One of these functions is to reduce the complexity of the accelerator process, and present various levels of abstractions in the presentation and to structure this presentation. It also must give easy and uniform access to any of the nucleus equipment elements of the accelerator complex. Another function is to provide a very abstract view of the process (beam physics view point) and global operation. Also important is the uniform error reporting with not ambiguous message presentation.
6.2 Properties of the implementation

They must be attractive for the user (operators and developers), and easy to maintain, fast for adaptations and modifications according to equipment and accelerator processing evolution. There must be a simple “programming” environment, easy to learn and to use by the various partners who contribute in developing applications. The operator interface will benefit from the rapid and enormous progress in the domain of computer—human interface. Specifications have to be very well understood and must give a powerful context of discussion between the partners (user and developers). On-line documentation must be provided automatically to the maintenance team for trouble-shooting.

6.3 Technical solutions

The current trend is to use graphical workstations or first class office computers (no more aggregates of computer peripherals), and to rely more on industrial software and international standards.

Reduction of complexity can be obtained by structured presentation, i.e. through global to detailed views (tree structure). The synoptic presentation is very attractive and user—friendly. There is a request of many graphic resources (data base with graphic data structure, and powerful graphic editor). The “virtual accelerator” for multi—pulse machines (PPM mode) gives an independent view for every kind of beam and reduces the coupling constraint of the equipment.

Simple equipment access can be done by general tools: knobs and panels of action buttons. Other tools for more complex cases are the data viewer facility, 2 dimensional or 3 dimensional representation running in repetitive mode (driven by the data flow), and synoptic “animation” with status, data, simple graphic, etc.

Another need is the good and easy programming environment with multi—processing, multi—windows, with a process manager (including resource manager) for console supervision, simplified graphics and interaction environment tuned to process. Simple programming is achieved, with interpreter including network facility (ex—NODAI), network compiler for compiled language, simple and uniform access to equipment data (object—oriented, data server).

6.4 Dedicated tools for maintenance

For the maintenance team and debugging phase, access to hidden data and main internal data flow is required. This is obtained by interactive access through the interpreter language, visibility of the list of equipment in use, validation procedure suite for reception and trouble—shooting, and a facility for data path tracing.

7. General problems of control systems

Certain general problems must be solved. This will be either explicitly chosen after evaluation, implicitly worked out by the facts and the history.
7.1 Security and access control

Here two problems must be solved. First, the users filtering for anti-hacker protection and the access to equipment or processes, only to people authorized. The solution is done by classical methods: absolutely no external access to the system, a non-popular environment (not UNIX or VMS), user identification and recognition. Secondly, the mutually exclusive access to equipment. Normally, the access to the accelerator process (equipment) can be made by one person only at a time (for controls). Consequently, access to controls must be mutually exclusive between the various simultaneous users. This protection can be done by offering only one access point to one equipment; by discussion between the operators in the control room; by reservation of nucleus equipment (semaphore mechanism); by exclusive split of the equipment between mutually exclusive working sets and access controlled by the general manager.

7.2 Analog observation system

This is needed to look at the raw waveform of power supplies, beam observation instruments and timing pulses. It must provide time correlation between various signals. It is very dependent on the hardware and must be very reliable as it is the last facility available for the operators when the control system is down.

7.3 Applications development policy

The applications are the main domain of evolution, modification and innovation during the lifetime of the controls system.

The two extreme types of applications are (i) a well-defined and stable control procedure, and (ii) the 5 minute program for a quick experiment with the accelerator beam. These extreme cases can imply well managed applications development with education, SASD, validation and documentation or a “do it yourself” application using building blocks and tools kit.

The applications can be developed either by a dedicated, well educated and indoctrinated team, where applications are well manageable, and the administration is very bureaucratic. In this case, the end-user can be unhappy, as he does not receive what he has expected. The applications can also be developed by the end-users themselves (operation, machine physicist teams) or the equipment specialists. It is then much more difficult to maintain these collections of software modules and to get a fast response time for modification. The user is always happy with his own product, but not necessarily the other users.

7.4 The lifetime of controls systems

The lifetime of large accelerators (not the prototype nor the test facility for accelerator research) is today bigger than 10 to 15 years. Typically, 20 to 30 years (or more). The lifetime of one generation of controls systems is around 10 years, for interface equipment it is 5 to 10 years. The lifetime of software modules can be counted with about 5 years, the one of process computers and peripherals with less than 5 years. All these figures imply that the controls system is in constant modification and that the happy situation of starting from scratch is very unusual.

Normally, one is confronted with compatibility problems, transition periods and phases, with difficulties of retrofitting the new facilities and last but not least the intolerable statement (of the young dynamic developers): "the option was chosen for historical reasons".
8. Hardware for control systems

The hardware depends very much on the computing and electronic “culture” of the laboratory. The current trends decrease needs for special equipment with the new products of the manufacturers, and encourage the use of standard (manufactured) equipment for process computers, workstations, computer network, bus for local or embedded systems, and on-board computers and I/O. Special systems are needed for time and events distribution, the analog signal distribution system and the low cost arbitrary waveform generator.

Key features for hardware are the intensive use of standard electronic equipment (I/O boards, buses, embedded computer) and standardization of local mecano.

9. Domains of research and development

9.1 Modeling

Modeling is used since a long time but more in off-line mode. The computing power of the new workstation and mini/micro processor allows the use on-line of large and powerful simulation programs. The programs in use are accelerator design programs (lattice computation and optimization programs (close orbit correction).

9.2 Expert systems

Embedded expert systems are appearing in control systems. They will be used for trouble-shooting, operator help and beam or accelerator optimization. There are two ways of development of these systems. One is the dedicated expert system for a very limited domain, built directly with the fourth generation language and small rules and facts base; the other one is a large expert system built around the generator of the expert system with a very large knowledge base (partially derived from the control system’s data base). The connection with the controls system is done by message exchange.

The main difficulty here is to extract from the various experts (accelerator, equipment, control, operation) their knowledge and to translate it into expert system representation. The long term research for these expert systems is the automatic learning system.

9.3 Applications generator facility

One attractive approach is the “spreadsheet” model. It means programming by the end-user without language (no grammar), and without installation procedure. It includes acquisition and control, algorithmic correlation and human presentation. It is a way to converge with the office automation computing.

9.4 Control Protocols

Their goals are to define stable control communication protocols between controls systems and equipment, and standard behaviour of the equipment. They are mainly applicable to power converters and instrumentation.
Bibliography


3. B. Kuiper, Controls for Particle Accelerators, CERN/PS/CO/Note 86 – 21.


6. B. Kuiper for the PS Controls Group, Controls for the LEP Preinjector, CERN/PS/85 – 21 (CO).


8. The PS Staff, presented by R. Billinge, The CERN PS Complex: a Multi-purpose Particle Source, CERN/PS/83 – 26


17. J Poole, Using Oracle Databases in a Particle Accelerator Control System, LEP Controls Note 68, 24.4.1986


25. R. Giachino, J. Miles, A. Spinks, Contention Resolution in a Distributed Control System, CERN/SPS/AOP/Note 87 – 12.


33. R. Rausch, Real-time Control Networks for the LEPI and SPS Accelerators, CERN/SPS/87 – 35 (ACC).


36. R. Bailey, J. Ulander, I. Wilkie, Experience with using the SASID Methodology for Production of Accelerator Control Software, CERN/SPS/AOP/Note 87 – 16.
1 Introduction

1.1 What is Robotics?

There have been many attempts to define the term robot. By and large, such definitions have been hopeless, since they have simply summarised the state of industrial systems current when the definition was framed. Most notably, consider the widely cited definition proposed by the Robot Association of America:

“A robot is a reprogrammable, multifunctional, manipulator designed to move materials, parts, tools or specialised devices, through variable programmed motions for the performance of a variety of tasks.”

This definition fails to mention sensing.

Robots that cannot sense (more precisely perceive) their environment inevitably are incapable of modifying their programmed motions to accommodate unexpected situations, uncertainties such as varying the positions of parts, or non-uniformities in the speed of conveyers. Such robots must be confined to an environment that is perfectly ordered and perfectly modeled. The real world is certainly like that, though the “perfectly understood world” assumption underlies the application of robots to spot welding, pick and place tasks and spray painting.

A robot that is confined to sensing its environment is seldom of much use. Robots are active devices; they change their environment by performing (hopefully useful) work. The simpler the effectors we equip robots with, the simpler and more circumscribed the actions they can perform. The parallel jaw grippers found on most current industrial robots can pick up conventionally-located, small workpieces such as computer chips, and they can place them in pre-assigned locations in conveniently-located printed circuit boards. Parallel jaw grippers are not much use if the task is to change a car’s distributor, however.

Simple sensors and simple effectors allow a robot to operate successfully in a world that has little uncertainty, and they can work from a simple model of that world. Richer sensors and more dextrous effectors potentially support a wider range of applications and potentially enable a robot to tolerate greater uncertainty. There is a price to be paid for versatility, however, and it has become the central theme for much research in robotics: The robot’s model of the world needs to be more complex, and the robot needs to have a greater understanding of it. In short, the robot needs to be more intelligent.

This observation has lead the robotics community to rally around the following, more general, definition of robotics (Brady 85):

“Robotics is the intelligent connection of perception to action.”
Modern robotics is all about the intelligent interpretation of sensor information (perception) in terms of tasks (actions) that can be performed by mechanisms and machines. The most difficult and interesting part of robotics lies in the "connection" of sensors to mechanisms. Advances in this area have only become possible with the advent of increasingly powerful computing systems and programming tools. Of all disciplines, robotics is probably the most computationally demanding; it represents the most sophisticated fusion of computing resources with the real world that current technology is able to achieve.

By way of introducing these lectures, we will expand on the above by employing the following definition:

A robot system consist of a mechanism for acting on and in the environment, a sensor system to obtain knowledge about the state of the mechanism and the environment, a controller and drivers to guide the mechanism and sensors in a desired manner, and a planning and control system that decides on the actions and sensing in that environment. The function of a robot system is to accomplish a specified goal by the intelligent interpretation of sensor information and mechanical actuations in terms of task, plan and model.

This draws together the components or "connections" that will be the subject of these lectures; sensor and mechanism control, sensor data interpretation, and task or goal planning.

A word on this preoccupation with definitions. Robotics is fundamentally interdisciplinary; researchers in the field come from many different backgrounds and have many different interests, they may be specialists in areas ranging from materials technology to differential geometry. Almost all branches of the physical sciences find roles to play in developing robotics technology. It is because of this wide-ranging remit, and because the development of the subject is still at such an early stage that roboticists have trouble defining their work.

1.2 Overview of Lectures

This whirlwind tour of robotics comprises three one-hour lectures. In an attempt to show the diversity of the subject, we have sacrificed a great deal of the interesting detail. At the end of each lecture is a short bibliography in which much of this detail can be found.

Each lecture concentrates on one aspect of the "connection" between perception and action: control, sensor interpretation, and planning. In each, we have attempted to give a good idea of the technology and techniques used, as well as indicating some of the more important remaining research problems. Each lecture concludes with a short video of a major project in the associated area of work.
2 Manipulation and Control

Robotic control spans a multitude of functions, ranging from simple servo control of a manipulators links, through path-control of mechanisms, to sophisticated high-level task control. The main point to note is that robot controllers are hierarchical (Figure 1). At the lowest level is the joint servo control, providing for the motion of actuators and the measurement of positions, velocities and forces. This layer in the control hierarchy is what most engineers would describe as "the control problem". Joint servo control usually requires knowledge of the kinematics and dynamics of the mechanism itself, the formulation of which can be rather complex. We shall spend as little time on this subject as is possible. The next level up in the control hierarchy comes the trajectory and motion planner, providing for the specification of positions and velocities for the joint controller to follow. This level is concerned exclusively with the kinematics of the mechanism involved, and can range from simple point-to-point teach-pendant programming methods through to quite complex path following or geometric control primitives. This is approximately the stage that has been reached by most commercially available industrial robots. Above this path control level is "geometric model" control, providing a basic description of the geometry of the operating environment from which paths and locations can be generated. It is at this level that sophisticated sensors such as

![Diagram](image-url)

Figure 1: Robot Control is Hierarchical
vision can most readily be used. At the uppermost level is task planning control, able to decompose logical specifications of a task into geometric operations that can be implemented by the lower levels. Artificial Intelligence has a big role to play at this level.

As good programming practice dictates, as the higher levels of this control hierarchy increase in competence, the lower levels become transparent to the user of a robot system. Ideally, a robot operator will, one day, be able to say, “Build me a new car...”, and it will be done! Although this is not possible today, the competence of robot systems is gradually moving up the control hierarchy, and significant work has been done at all levels.

2.1 Mechanism Control

For most mechanism control problems, the manipulator is treated as a series of connected, rigid links. Figure 2 shows a Puma-560 manipulator, comprising three lower-arm links together with a wrist mechanism; six revolute joints in all. The dynamics of this mechanism can, most simply, be modelled by the (Euler-Lagrange) equation:

\[ J(q)\ddot{q} = t - H(q, \dot{q}) - G(q) \]  

(1)

where \( q \) is the \( n \)-dimensional vector of joint locations. Equation 1 describes how robot acceleration is related to the set of applied torques \( t \) by the inertia matrix \( J(q) \) with some of the available torque being taken up by Coriolis and Centripetal forces \( H(q, \dot{q}) \) and gravitational forces \( G(q) \).

Do not be deceived by the simplicity of this equation! There are many problems associated with the derivation and implementation of these equations in joint-servo control. Note, for example, that the all terms are configuration dependent, typically comprising a hundred or more trigonometric terms for each of the joint vari-

![Figure 2: The Puma-560 Manipulator Mechanism](image-url)
ables. Also note that we have made no mention of friction, motor non-linearities, or the flexibility of links.

2.1.1 Position, Velocity and Acceleration Control

The mechanism control problem can briefly be stated as specifying required trajectories for the joint variables q (and their derivatives), then providing suitable inputs \( t \) that allow these trajectories to be followed, subject to the response of the dynamic system described by Equation 1. These trajectories are typically some collection of points that the end effector must pass through (position control), or some velocity or acceleration sequence that the links of the mechanism must be constrained to follow.

The simplest form of control (and that used in most industrial controllers) is to provide a PID (position-integral-derivative) feedback loop around each of the joint actuators. This provides an input to the actuator proportional to a weighted sum of the difference between measured and desired joint positions, their integral, and their rate of change. The object is to successively reduce the error between a desired joint-trajectory and that demanded by the controller. No allowance is made for non-linearities in either mechanism or actuators. The consequent conservative choice of control parameters can cause the mechanism to respond slowly or with varying, configuration-dependent performance. In simple, sensorless, industrial assembly situations, however, this type of control is perfectly adequate.

One method of overcoming the problems associated with configuration-dependent non-linearities in servo-control is to calculate the values of the non-linear terms in Equation 1, on-line, and compensate for them, so linearising the equations governing the response of the mechanism. This method is called Inverse Dynamic Control and is shown in Figure 3. Inverse dynamic control first requires that Equation 1 be solved using currently available information on position and velo-
ity. This is then used to cancel out the configuration dependent non-linearities associated with the mechanism. In it’s stead is substituted a second-order linear control for each joint.

In practice, there are two problems with inverse dynamic control; the computation of the dynamic non-linearities at the speeds required of the controller is almost impossible, and the effect of other non-linearities in the system (friction, for example) prevent complete decoupling.

A third means of obtaining precise position control of the actuators and mechanism is to use adaptive control procedures. The idea behind these methods is to dynamically compensate out non-linearities by comparing the response of the actual system to the response of an ideal model, and using the difference to adjust the gains in a second order controller. Although such techniques are well advanced in some areas of industrial control, the speed of manipulator motion and the consequent band-width limitations have so far restricted application these methods in robotics.

The result of the servo-loop control layer is to provide a means of translating desired joint trajectories into mechanism motion. After this level of control, no more reference need be made to mechanism dynamics.

2.1.2 Coordinate System Control

Controlling the positions of the joints of a mechanism as complex as that shown in Figure 2 is not, by itself, very useful. This is because it is not easy for a robot programmer or operator to describe a task in terms of the required locations of a manipulators links. Far more useful is the description of desired locations in terms of easily-understood cartesian coordinates. The specification of a robot task in terms of cartesian motions considerably simplifies the problems associated with programming manipulator motions.

An arbitrary end-effector location can be specified by six parameters, three describing position, and three describing direction. In general the specification of locations can be made with respect to any convenient coordinate system. The two most common coordinate systems are the base or world coordinate system, and the end-effector or tool coordinate system (Figure 4).

To provide cartesian control of a manipulator mechanism, both the forward and the inverse kinematics of the mechanism must be known. The kinematics of the mechanism describe the relation between joint angles and the cartesian location of the mechanism’s links and end-effector. In order to deal with the complex geometry of a manipulator, coordinate frames are fixed to each part of the mechanism and the relationship between all these different frames is found. These relationships are functions of the joint variables and joint angles. To implement cartesian control, these relationships must be inverted for any desired end-effector location to yield the joint angles which will put the mechanism in the required configuration. These calculations can be exceptionally tedious, though usually not as computationally expensive as for the mechanism’s dynamics.
2.1.3 Trajectory Generation and Tracking

A common way of causing a manipulator to move from here to there in a smooth, controlled fashion is to cause each joint to move as specified by a smooth function of time. Commonly, each joint starts and ends its motion at the same time, so that the manipulator motion appears coordinated. Exactly how to compute these motion functions is the problem of trajectory generation (Figure 5).
This problem also includes the human interface problem of how we wish to specify a trajectory or path through space. In order to make the description of the manipulator motion easy for a human user of a robot system, the user shouldn’t be required to write down complicated functions of space and time to specify the task. Rather, we must allow the capability of specifying trajectories with simple descriptions of the desired motion and let the system figure out the details. For example, the user may specify the desired goal position and orientation of the end-effector, and leave it to the system to decide on the exact shape of the path to get there, the duration, the velocity profile and other details.

Typically, path generation methods work by taking end-points and transit-points of a motion specified by a robot programmer (or task generating program), and interpolating between points using spline functions to produce a smooth trajectory. Constraints on velocity and acceleration characteristics are incorporated directly into the spline optimisation process.

Usually paths are calculated off-line, but generation can also occur at run-time and in the most general case, position, velocity and acceleration are computed. Typical path update rates for dedicated controllers now approach 200Hz.

2.1.4 Robot Mechanism Programming Languages

Robot manipulators differentiate themselves from fixed automation by being “flexible”, which means programmable. Not only are the movements of manipulators programmable, but through the use of sensors and communications with other factory automation, manipulators can adapt to variations as the task proceeds.

Before the rapid proliferation of microcomputers in industry, robot controllers resembled the simple sequencers often used to control fixed automation. Modern approaches focus on computer programming and issues in programming robots include all the issues faced in general computer programming, and more. There are two basic levels of robot mechanism programming (leaving aside task-level programming):

Early robots were all programmed by a method that we will call teach by showing, which involved moving the robot to a desired goal point and recording its position in a memory which the sequencer would read during playback. During the teaching phase, the user would guide the robot by hand, or through interaction with a teach pendant. Teach pendants are hand-held button boxes which allow control of each manipulator joint or of each cartesian degree of freedom.

With the arrival of inexpensive and powerful computers, the trend has been increasingly toward programming robots via programs written in computer programming languages. Usually these computer programming languages have special features which apply to the problems of programming manipulators, and so are called robot programming languages. There are three categories of robot programming languages: Specialised manipulation languages; languages that have been built by developing a completely new language addressing robot-specific areas. These languages are common in industrial manipulators. Robot libraries for an existing computer language have been developed by starting with a popular computer language (e.g., Pascal) and adding a library of robot-specific subroutines. These are most common in research areas. Robot libraries for a new general-purpose language have been developed by first creating a new
general purpose language as a programming base, and then supplying a library of predefined robot-specific subroutines.

2.2 Task Level Control

The most complex control systems available on current industrial robots are generally limited to the specification of cartesian trajectories, in a number of different coordinate systems. Research is now largely aimed at making the specification of robot motions task-driven, that is motions and manipulations being automatically generated from a geometric description of the operating environment. The need for this kind of facility is made clear when it is considered how humans go about the problem of, say assembly: We talk in terms of placing surfaces in contact, or pushing one object inside another, that is, we talk in terms of the objects of interest themselves, rather than the arm and hand motions that are required to achieve these results.

This idea of task-oriented control of manipulator mechanisms is usually considered in two parts:

1. Given the initial geometry and location of an object in the environment, what application of rigid-body translations and relations to this object will result in a specified final geometric state, and what cartesian manipulations are required to implement these object motions?

2. Given a set of objects in the manipulators workspace, with some known, or sensed initial state, how should the objects be moved around in order that some final, desired, composite structure be constructed.

We will refer to these as geometric task planning and logical task planning.

2.2.1 Geometric Tasks

The basis for geometric task planning is a geometric model of the objects and the environment in which the manipulator operates. This geometric description provides a basis from which manipulation plans and assembly operations can be generated.

Typically, the geometry of the world is described in terms of a CSG (constructive solid geometry) world model. Geometric task planning then proceeds in four phases;

- Objects and relations between objects are modelled in initial and final states.

- Objects are located by matching observed features in to a data-base of expected objects.

- Paths for the manipulator and objects are calculated, by considering intersections of primitive geometric shapes, to transform an initial geometric configuration into a final desired state.

- Grasps positions are calculated from object models and generated paths.

These phases are all described geometrically, in terms of the volumes occupied by objects and the surfaces offered for contact. The robot is instructed by specifying
Figure 6: A plan found by a geometric path planner. The first image is on the top left; it shows the initial position of the part identified by a simple vision system. The final position of the part, specified by the user, is on the bottom right.

Spatial relations that are established between parts being manipulated in successive stages of the assembly process. Figure 6 shows a geometric plan found by geometric task planner.

The computations involved in planning geometric tasks can be very complex, particularly if there are many objects to be manipulated or the task itself is complex. The result of programming a geometric task is usually a set of sensorless manipulations and motions, executed by a cartesian controlled robot mechanism. A geometric plan is usually generated off-line.

2.2.2 Logical Tasks

A logical description of tasks must run in parallel with the geometric description of tasks. The task-level description of a sequence of manipulation operations consists of an initial logical and geometric description of the operating environments and a final logical end-state. By application of manipulation rules, a sequence of manipulation operations are applied to the initial state to yield a final state. This logic planning stage reduces a complex series of operations in to simple one-stage motions or manipulations that can be further analyzed by the geometric task planner.
Logical task planning is still at a very primitive stage of development. Much has still to be learned regarding the complex interaction between geometry and planning which cannot be separated as naively as described above.

2.2.3 Design-to-Product

The ultimate goal of industrial robotics is to fully automate the integration of product lifecycle: from design through manufacture to field servicing and maintenance. This is design to product (DtoP); the integration of production with design and product specification.

The DtoP philosophy is suprisingly quite advanced. Major work has been done on a number of important components:

- Product design systems that encapsulate the limitations of materials and machines.
- The description of parts in terms of the machining and assembly operations that need to be performed on them.
- The automatic decomposition of a product description in to work-cell and manipulator operations.

Indeed, “Build me a car...” may not be as far away as we think.

2.3 Hands and Grippers

Industrial uses of robots typically involve a multi-purpose robot arm and an end-effector that is specialised to a particular application. End effectors normally have a single degree of freedom: parallel-jaw grippers, suction cup, spatula, ‘sticky’ hand or hook (Figure 7 shows a typical industrial gripper). The algorithms for using such hands are correspondingly simple. However, many tasks, particularly those related to assembly require a variety of capabilities, such as parts handling, insertion, screwing, as well as fixtures that vary from task to task.

The importance of hands, manipulation and contact to intelligent robotics as a means of providing “action” is really only just being realised. The complexity, both of the mechanics and control of dextrous effectors, has so far limited there application in industrial areas, although they have recently provided some important advances; Figure 8 shows a recently developed advanced dextrous hand.

Experience with manipulators has pointed to a need for hands that can adapt to a variety of grasps and augment the arm’s manipulative capacity with fine position and force control. Currently a significant proportion of the cost of operating a manipulator is in the development of specialised end effectors for each task. The lack of fine force control capacity limits most robot applications to only coarse tasks. Articulated hands appear to offer some solutions to these two problems. The ability of an articulated hand to reconfigure itself into a variety of grasps reduces the need for specialised grippers. The proximity of low mass, powered joints to the objects being manipulated reduces modeling errors and dynamic complexity. This facilitates achieving high bandwidth and fine control of motions.
Figure 7: A Typical End-effector for Use with an Industrial Robot

Figure 8: The Stanford/JPL Articulated Hand
2.3.1 Mechanisms and Contact Geometry

In contrast with current industrial end-effectors, a human hand has a remarkable range of functions. The fingers can be considered to be sensor-intensive 3- or 4-DOF robot arms. The motions of individual fingers are limited to curl and flex motions in a plane that is determined by the abduction/adduction of the finger about the joints with the palm. The motions of fingers are coordinated by the palm, which can assume a broad range of configurations. The dexterity of the human hand has inspired several researchers to build multi-function robot hands.

A hand mechanism is composed of a collection of rigid bodies called links. One link, sometimes referred to as the palm, is fixed in a reference frame. The object being manipulated is also counted as one of the links. The links in the mechanism are connected by joints and contacts. A contact results when the surface of any link touches the surface of the object that is being grasped or manipulated. The contact may be located anywhere along a link’s surface. Figure 9 shows a number of contact configurations for a single finger.

The design of a mechanism may be approached in at least three ways:

- **Number Synthesis** deals with the number of degrees of freedom in a mechanism by looking only at the freedoms in the joints and contacts between the links.

- **Type synthesis** looks at the relative motion allowed by each connection within a mechanism and the net effect of all such connections on the motion of each link within the mechanism.

- **Dimensional synthesis** deals with the specification of the major dimensions of a mechanism, such as link lengths, and their effect on motion of link.

![Figure 9: Contact configurations: The eight ways in which a 3-link finger may touch an object are shown](image)

436
The kinematics of a hand mechanism (Figure 10) is needed to understand the connectivity and mobility of objects. An analysis of contact mechanisms is also required. Many of the techniques used in hand design come from much older work on screw mechanisms.

2.3.2 Contact Sensing

Most industrial end effectors have no built-in sensors. Those that do typically incorporate devices that give a single bit of information. The most common are contact switches and infra-red beams to determine when the end effector is spanning some object. Certainly very few robot programming languages make allowance for path or task modification based on sensory information. This is gradually changing as force-sensing wrists and other sensors providing detailed contact information become available.

2.3.3 Grasping and Manipulation

For a manipulator or hand mechanism to do useful work it must be able to predictably cause motion of, or apply force to, grasped objects. Unexpected and unobservable motions due to slippage or incomplete constraint of a grasped object place serious limitations on a mechanisms utility. Therefore, an important subset of hand mechanisms are those that are able to exert arbitrary forces or impress arbitrary small motions on the grasped object when the joints are allowed to move, and constrain an object by fixing all the joints.

To determine if a particular grasp on a body imposes sufficient constraint to immobilise it completely, we need both an understanding of the geometry and mechanism during grasping, and a means of describing the forces and contacts that must be preserved to achieve a particular grasp. The subject of grasping and manipulation is still poorly understood. Specific solutions have been obtained for simple operations like aligning, pushing and the geometry of stable grasps.

Bibliography


3 Sensors and Perception

Sensors are a fundamentally important part of any intelligent robotic system. Without sensing robots would be unable to locate objects of interest or “understand” the environment in which they operate. Sensors are only half the story though, it is the automatic interpretation of sensor information that distinguishes robotic sensing problems from more conventional purely data-acquiring sensing. This processes of sensor data interpretation is unfortunately referred to as perception.

A great variety of sensing devices are used in robotics, ranging from sophisticated three dimensional vision systems through to simple binary contact sensors. Current industrial sensing systems are extremely primitive, and are severely limited especially when compared with laboratory systems. This is because, although the sensing devices themselves are quite highly developed, the algorithms used to interpret sensor information are both (very) computationally expensive and relatively poorly understood. However rapid strides are being made in the area of sensor data interpretation, as more computer power becomes available and our understanding of the problems involved increases.

We will divide the subject of sensing in to four parts; vision, contact sensing, ranging, and sensor integration. The first three parts are concerned with different types of sensors, the data that can be obtained, and the algorithms that are currently used to interpret this information. The fourth concerns the use of a number of different sensors, acting cooperatively to provide environment descriptions.

3.1 Machine Vision

It is probable that three-quarters of robotics research is now dedicated to the problem of sensing and perception. Of this, more than half is dedicated to research in to machine vision – vision is a hard problem!

Vision is our most powerful sense. It provides us with a remarkable amount of information about our surroundings and enables us to interact intelligently with the environment, all without physical contact. Vision is also our most complicated sense, we know almost nothing about how our own vision system works, let alone how to build an artificial vision system. Nevertheless, today one can find vision systems in operation, capable of quite complex operations.

Most progress has been made in industrial applications where the visual environment can be controlled and the task faced by the machine vision system is clear-cut. Less progress has been made in those areas where computers have been called upon to extract ill-defined information from images that even people find hard to interpret. The “universal” vision system is still a long way in the future.

3.1.1 Sensors and Images

A machine vision system analyzes images and produces descriptions of what is imaged (Figure 11). The front-end of the system is the camera. These days, the cameras used by a vision system usually consist of an array of charge coupled devices (CCD arrays), together with lenses and associated electronics. The cameras can now be made quite small, typically 3cm cubed. The CCD array lies on the
imaging plane of the camera, and each element (pixel) of this array collects charge proportional to the intensity of light falling on it. A typical CCD array comprises 512×512 square array of pixels (over 250,000 in all). Images are acquired from the camera through a processes of digitising the charge in all the pixels; quantising the analog intensity value to provide a binary number (usually 0–255), called the grey-level (in a grey image !), which describes the intensity of light falling on the image surface. A typical image captured from a CCD array is shown in Figure 12. This image acquisition processes is the starting point for all vision systems. The camera and digitiser are now quite standard, commercially available devices.
3.2 Edge Detection and "Low-Level" Vision

A great deal of effort has been devoted to understanding how significant intensity changes, or edges, can be extracted from an image. Edge detectors of some kind have been an essential part of many computer vision systems. The edge detection process serves to simplify the analysis of images by drastically reducing the amount of data to be processed, while at the same time preserving useful information about object boundaries. The edges extracted from an image are often termed a primal sketch, resembling a pencil sketch of the imaged scene. The edges extracted from an image can be used in a number of specific vision applications, such as object identification or motion detection.

The basic edge detection process proceeds by differentiating the array of image intensities $I(x, y)$, to find places of maximum change: intensity changes (edges) correspond to maxima of the gradient of the image surface, or equivalently a place at which the second derivative crosses zero and change signs. There are a great many variations on this theme of edge detection, we will mention just a few:

Differences of boxes or the Sobel operator is the simplest form of edge detection. It is just the difference between adjacent pixel values. It has very good edge localisation properties, but is very prone to noise-induced errors. Edges are found by locating maxima in the resulting image and thresholding.

First derivative of the Gaussian, reduces the noise problem by first blurring the image with a Gaussian, and then differentiating. This degrades localisation but is generally an improvement on the Sobel operator. Figure 13 shows a typical one-dimensional image signal operated on by these two masks.

Zero crossings of the second derivative of the Gaussian $\nabla^2 G$, also provide a means of locating maximum rates of change in an image.

These simple edge detection procedures can be implemented by applying a convolution mask to each pixel in the image array; that is, a discrete convolution in a neighbourhood about each pixel. Different masks can be chosen for different applications; finding different types of edges, or edges oriented in particular directions, for example. Figure 14 shows a number of typical gradient masks.

There are still many problems in the basic edge detection process of which we can only mention a few:

- Edges are not always adequately described by a step, they often look like a roof, or have several smaller changes.

- Edges occur over different scales, some are "instantaneous", some occur over a large distances. No single convolution mask will detect edges reliably at all scales.

- It is difficult to find an "optimal" choice of localisation quality and noise insensitivity. As previously indicated, blurring reduces noise but also decreases the accuracy with which an edge is detected.

 More sophisticated edge detectors exist that use a variety of convolution masks, supplemented by other algorithms to localise and "grow" edges. One worth men-
Figure 13: (a) a noisy stereo edge. (b) Difference of boxes operator. (c) Differences of box operator applied to the edge. (d) First derivative of Gaussian operator. (e) First derivative of Gaussian applied to the edge.

Figure 14: Various Gradient Operators
tioning is the Canny edge detector. This uses the first derivative of a Gaussian at a number of different scales, coupled with a mechanism for suppressing local non-maxima and iteratively growing edges. A typical output from a Canny edge detector is shown in Figure 15 – the improvements are obvious.

There are a number of other basic image processing problems that can be reduced to the application of a convolution mask to an image array; masks exist for extracting different texture types, for extracting different spatial frequencies, for smoothing and blurring, for example. These convolutions are computationally very expensive; an 8x8 mask operating on a 512x512 image requires about 20 million multiplications. Until recently this limited the application of edge-detection. However, it is now possible to obtain commercial systems capable of performing these convolution processes at rates of 50hz or more; approaching 1,000 MIPS, for less than the cost of most micro computers.
3.2.1 Stereo Vision and Motion

A single image, or edge map, is only able to provide two-dimensional information about a scene of interest. To obtain three dimensional information, two or more cameras are required.

The basic stereo-vision processes involves the acquisition of two images of the same scene, taken from slightly different locations, and, by finding corresponding features in both images, triangulating for the depth to different points in the environment. Figure 16 shows the typical geometry of the stereo process. Each feature in the environment appears in both left and right images, and different distances from the optical axis. The offset between a feature in one image against a feature in the other image is termed the disparity. By knowing the baseline distance between the cameras, and other calibration constants, this disparity can be used to calculate the absolute depth of different observed features.

The key to an automated stereo system is a method for determining which point in one image corresponds to a given point in the other image. This is known as the correspondence problem. There are a number of different approaches to this problem:

- **Correlation Methods** attempt to match, pixel by pixel grey level intensities in one image with those in another. These methods do not work very well – correlating two arrays as large as 250,000 data points is computationally, very difficult.

- **Edge Matching** reduces the computational load and decreases noise sensitivity by only matching pixels that correspond to edges (edgels) in the image. This is still computationally expensive, although the cost can be significantly be reduced by using various geometric constraints, such as the epipolar constraint; requiring matched edges to lie in the same horizontal plane. Edge matching is still the most popular stereo method.

---

![Diagram](image)

Figure 16: Simple Camera Geometry for Stereo Vision. The Optical Axes are Parallel to each Other and Perpendicular to the Base Line.
Figure 17: A Stereo pair, edges extracted from both images and the matches obtained by a line-matching method.

- **Line Matching** reduces computation still further by only attempting to match complete lines extracted from each of the left and right images. The advantage of this method lies in its relative speed, the disadvantage is that fewer matches are obtained. Other, more structured features could be used in the matching process.

A typical pair of images and the matches obtained are shown in Figure 17.

The result of the stereo process is a so-called **2.5-D sketch**; detailed information in two dimensions, together with a sparse depth map. This depth map is used in later vision stages, from which surfaces can be interpolated or objects recognised.

Motion analysis is similar to stereo in that depth information is extracted by triangulation. Usually a sequence of images is used, and successive depth
measurements are estimated from one frame time to another. Related to this is a variety of optical flow techniques for segmenting features in an image, based on common apparent velocities of adjacent pixels. Figure 18 shows an example segmentation from one of these algorithms.

3.2.2 Other Depth Cues

We will briefly describe a number of other means of obtaining depth from an image or sequence of images:

- **Depth from vergance:** by verging the cameras until a feature in each image is made to overlap, depth can be calculated from the angles subtended by the two cameras.

- **Depth from focus:** by bringing small windows of an image in to sharp focus, the focal length of lens and camera can be used to determine depth.

- **Depth from texture:** textures vary in their spatial-frequency spectrum as their observed depth changes. By knowing these variations, depth to a textured surface may be extracted.

- **Depth from shading:** by knowing the reflectance properties of objects and some geometry and physics about the illumination on a scene, depth can be estimated by considering the change in intensity of reflected energy from a given surface.

There are many other algorithms; often coming under the general heading Shape-from-X algorithms.
3.3 Shape Representation

The underlying basis for the vast majority of vision research is an understanding of the geometry or shape of objects and their projection on the image plane. Representation is central in connecting observed data with objects that can be understood by a computer system. The majority of robotics systems stick rigidly to a world described by straight lines and planer surfaces. Although this is sufficient for man made, industrial environments, it is all but useless for 'natural' scene understanding; identifying peoples faces, for example. We can only briefly describe some of the more common shape representations used in robotics; the subject is one of the most difficult encountered in robotics, but is also extremely interesting.

 Broadly, three dimensional shape representations can be broken down into three distinct classes; surface or boundary, generalised sweeping, or volumetric descriptions.

 The enclosing surface or **boundary**, of a well-behaved three-dimensional object should unambiguously specify the object. Since surfaces are what is seen, these representations are important for computer vision. **Facet models** describe an object by breaking up it's surface into (planer) facets, each individually parameterised. This is most popular in environments where the objects are naturally polyhedral. Figure 19 shows a boundary representation for an industrial part. Related to facet models is the **winged edge** representation, which gives a natural, logical description of the relation between surfaces that can be manipulated by a computer (Figure 20). The idea of segmenting an object boundary into patches has been extended in a number of ways by increasing the complexity of individual patch descriptions. **Spline models**, model each patch by an appropriately dimensioned polynomial. Other patch descriptions include **spherical harmonics**

\[ \text{Figure 19: A Volume and the Faces of a Boundary Representation} \]
Figure 20: A Subset of Edge links for a tetrahedron using the winged edge representation

and superquadrics (fractional quadrics in trigonometric terms). Quite complex objects can be modeled using some of these techniques. Figure 21 shows a scene modeled by superquadric patches. All can be combined with some flavour of winged-edge computer representation.

The volume of many biological and manufactured objects is naturally described as the swept volume of a two dimensional set moved along some three-space curve. Simple shapes sweep out cylinders, polyhedral or constant cross-section shapes (Figure 22). General sweeps are quite a popular representation in computer vision, where they go by the name generalised cylinders. A generalised cylinder is a solid whose axis is a 3-D space curve. At any point on the axis a closed cross section is defined. Usually it is easiest to think of an axis space curve and a cross section point set function, both parameterised by arc length along the axis curve. Figure 23 shows the type of models that can be generated.

A representation of objects in terms of primitive solids is often useful and usually have the advantage of computational simplicity. The simplest form of volumetric description is the spatial occupancy array. Volumes are represented as a three-dimensional array of cells which may be marked as filled with matter or not (Figure 24 for example). Spatial occupancy arrays require much storage if resolution is high. It is sometimes useful to convert an exact representation into an approximate spatial occupancy representation. A step up from spatial occupancy is cell decomposition. In cell decomposition, volume elements (voxels) are more complex in shape but still 'quasi-disjoint', so the only combining operation is 'glue'. Cell decompositions are not particularly concise and many objects are not amenable to this type of analysis. An advance on cell decompositions is a technique known as constructive solid geometry (CSG). Solids are represented as compositions, via set operations, of other solids which may have undergone rigid motions. At the lowest level the primitive solids are bounded intersections of closed half-spaces, defined over some well-behaved analytic function. Figure 25 shows
an example of a CSG construction. The value of CSG as a modeling technique is adequately demonstrated by its almost uniform use in computer aided design systems.

Figure 21: A complex scene modeled using superquadric patches

Figure 22: A translational sweep of a two-dimensional outline, forming a three-dimensional solid.
Figure 23: Generalised cylinder representation of two kidneys and a spinal column, generate from CAT data.

Figure 24: A solid (the shape of a human red blood cell) approximated by a volume occupancy array.

Figure 25: Constructive Solid Geometry

The subject of shape understanding has recently undergone something of a revival as researchers have become more familiar with the powerful mathematical techniques used in physics for describing space. Of note, is the use of differential geometry to describe ‘squashy’ objects, and the use of wave mechanics in extracting symmetry properties. Research in these areas is still at an early stage.
3.3.1 Model-Based Vision

One simple method of recognising objects in images is to first model the object of interest in an appropriate form, and then only look for features that are instances of this model in an image. **Model-based recognition** considerably simplifies the image understanding problem and finds application in a number of industrial systems. There are two basic techniques in model-based vision:

**Template matching** is a simple filtering method of detecting a particular feature in an image. Provided that the appearance of this feature in the image is known accurately, one can try to detect it with an operator called a **template**. This template is, in effect, a subimage that looks just like the image of the object. A similarity measure (correlation) is computed which reflects how well the image data match the template for each possible template location.

**Feature-based matching** is based on the idea that salient features, extracted from an image, can be used to identify and localise a specific instance of a set of possible objects. Typical features are corners, or parallel lines; features that offer as much geometric constraint as possible. Given a set of these features, the database of possible objects is searched to find an interpretation which is consistent with the observed data. Figure 26 shows a simple feature-based matcher at work;

---

Figure 26: Model-Based Visual Identification and Localisation
edges are found, and relations like 'parallel' and 'perpendicular' are used to identify and localise the object.

It is worth emphasising that model-based vision techniques are search-intensive, that is, the complexity is exponential in the number of objects needing to be recognised. This is why so much emphasis is placed on constraints which can be used to reduce the search space. (searching and constraint appear with depressing regularity in vision)

3.3.2 Surface Reconstruction

Visual reconstruction is the problem of reconstructing geometric surfaces from visual intensity information. Two important techniques have been developed that are able to segment and fit surfaces to intensity arrays with associated sparse depth maps. The starting point for these techniques is an x-y array of depth values. The object of the exercise is to break such depth maps, generally at edges or perceived discontinuities, into piecewise analytic surfaces. Surface reconstruction is an essential step in translating image information into meaningful descriptions of objects.

The first technique models potential surfaces as weak membranes (continuous functions that are breakable) or as weak plates (functions whose first derivative may be discontinuous). An energy function is set up which contains terms for the attraction between data points and the elastic breaking strength of the surface model. By variational (finite element) means, a set of surface functions are found that will minimise the energy functional used to describe the surface. This type of visual reconstruction processes are founded on least-squares regression. Figure 27 shows one such algorithm in action. There are a number of variations on this basic idea, including the use of multiple resolution depth maps, the use of weak 'strings' to model discontinuities, and the use of additional constraining information.

The second approach is to model the depth information as a Markov random field (MRF). The state of each depth point is modeled by a temperature-

Figure 27: Multiresolution reconstruction of a hemisphere from depth information
dependent Gibbs distribution. Energy functions are defined which describe the desirability for continuity of surfaces and the existence of discontinuities between regions. Minimisation of these energy functions is accomplished by a process of simulated annealing over the MRF. Figure 28 shows one such algorithm in action.

One interesting advantage of both these techniques is that they are naturally parallel computations. It is likely that they can be implemented at frame-rate speeds.

3.4 The Interpretation of Visual Information

Assuming that, using the algorithms described above, we have ended up with a geometric description of the environment, extracted from visual images of a scene: we are still left with the problem of interpreting this information in terms of our knowledge of the environment and the tasks we wish to perform. Visual understanding relates input and its implicit structure to explicit structure that already exists in our internal representations of the world. This involves a process of translating or labeling geometric entities, obtaining some higher-level representation that can be manipulated and used to make decisions about an environment and the tasks that can be performed in it.
An example serves to introduce the nature of the problem; Figure 29 shows a collection of rectangles that have been extracted, by some means, from an image. The rectangles together constitute a model puppet. The task is to label the rectangles appropriately, so that the composite structure provides a consistent, part-by-part, description of the puppet.

The solution of such problems involves two stages; knowing what possible labels we can apply to each component in the scene, and knowing what rules we can apply that will reduce the number of possible labels, to a single (correct) interpretation.

The "possible labels" come in two flavours; either as a semantic net, relating parts of an object to each other (Figure 30), or as a set of alternative labels that are related to each other by a set of rules.

A semantic net describes the structure of an object and how its component parts are related. The nodes of the semantic net give the possible labelings of parts in a scene, perhaps with some local geometric constraints on size, shape or colour. The rules that are applied to reduce possible labelings are maintained in the arcs that relate the nodes together. For example, the legs of the chair are all connected to the seat. The labeling process proceeds by searching for supporting hypotheses to confirm or reject partial interpretations. The semantic net idea has received considerable attention in the AI literature.

The more general "scene labelling" problem initially considers all segments in the scene to take labels with some prespecified probability. The process of

![Diagram showing the labeling process.](image)

**Figure 29:** "Understanding" the puppet image by finding consistent interpretations.
search and relabeling proceeds by application of a number of simple rules which successively eliminate or reduce the probability of inconsistent labels. For example, "the sky appears above the ground" can be used to eliminate interpretations of an outdoor scene that place grass below the sky (!). A famous example of using rules like this to reduce possible interpretations occurs in the consistent labeling of line drawings. Figure 31 shows the labeling of a general scene and a line drawing.

Figure 30: A semantic net describing a table and chair

Figure 31: labeling objects in a scene
Visual understanding is difficult problem, most of which is unsolved. Many of the techniques developed rely more on AI principles rather than any explicit use of geometry or understanding of the vision process. It is worth noting here that we have again returned to the problem of search and search reduction by local constraint – a recurring theme throughout vision research.

3.5 Contact Sensing

Sensors other than vision are used in robotics, though none are as highly developed, both in terms of hardware and the algorithms that go to support them. Contact sensing is an important category of sensors that differ in many aspects from vision. Contact sensors are essential in in manipulation and assembly, where force and contact information is required to move or position objects and parts. We will briefly consider two types of contact sensors; force sensors and tactile or ‘touch’ sensors.

3.5.1 Force Sensing

The main motivation for force sensing is to achieve compliant assembly. A force sensor typically measures the forces that are felt by an end effector during object manipulation. Force sensors have improved considerably over the past five years. Typical sensitivities range from 0.001–10 kgm/s². Two types of force sensors are currently in use; direct force measurement at the manipulators joints, and force-torque wrists fixed between a manipulator and its end-effector. Figure 32 shows a commercially available force-measuring wrist.

Force information is used in conjunction with position information to specify a task such as a peg-in-hole insertion. Stiffness, or active force is specified in workspace coordinates, enabling the generation of compliant motions. Consider for example, programming a robot to write on a blackboard. Clearly the position of the arm must be controlled, but the more difficult problem is that if one presses too hard, the surface or writing implement can be damaged, and if one does not press hard enough, the writing implement may leave the surface. By controlling

![Diagram of a force-sensing wrist](image)

Figure 32: A force-sensing wrist in typical ‘maltese-cross’ configuration.
force as well as position, small errors in contact can be controlled. Figure 33 shows the complex compliant motion problem of shearing a sheep.

3.5.2 Tactile Sensing

While vision will continue to be the primary sense utilised in robots, tactile or touch sensing also fills an important role, not only when vision is inapplicable or not precise enough, but also in the very fine control of delicate manipulation operations and in the measuring of an objects physical properties. The development of tactile transducers is significantly less advanced than visual sensors, very few tactile sensors are, as yet, commercially available. However, touch sensing is currently the subject of intensive research. Manufacturing engineers consider tactile sensing to be of vital importance in automated assembly. Particular need for tactile sensors lies in obtaining contact information to verify visual information, and in supplying information in environments in which visual identification is unobtainable due to obscuration of the object, perhaps by a robot end-effector during a manipulation operation. Tactile sensors may also reveal information regarding object texture, hardness and elasticity which are not readily available from any other sensing technique. Tactile sensors will be vital in supplying feedback information for manipulations and in overcoming slip.

Figure 34 shows one of a number of laboratory tactile sensors consisting of an anisotropic silicon conducting material whose lines of conduction are orthogonal
Figure 34: A laboratory demonstration tactile sensor

to wires of a printed circuit board. The sensor has a resolution of 256 points per square centimetre.

Even less progress has been made in the algorithms needed for the interpretation of tactile sensor information. Simple pattern-recognition algorithms have been developed, capable of interpreting data in terms of small modeled objects. Methods of extracting surface orientations and tracking edges have also been developed.

3.6 Other Ranging Sensors

A number of other range sensors have been used to produce depth maps. These usually have the advantage of being faster than visual methods, but all lack the variety of information obtainable from vision. We will describe some of the more common ranging sensors.

3.6.1 Laser Ranging

There are two basic methods of laser ranging. First, one can emit a very sharp pulse and time its return. This requires a sophisticated laser and electronics for scanning the beam across a scene and for timing the pulses. The second technique is to modulate the laser light in amplitude and upon its return compare the phase of the returning light with that of the modulator. The phase differences are related to distance traveled. Either of these techniques can be used to scan a scene of interest. They both provide dense and accurate depth maps, but are both expensive and slow.

3.6.2 Structured Illumination

Light Striping is a particularly simple case of the use of structured light. The basic idea is to use geometric information in the illumination to help extract
Figure 35: Light striping. (a) A typical arrangement; (b) raw data; (c) data segmented into stripes; (d) strips segmented into two surfaces.

geometric information from the scene. The spatial frequencies and angles of bars of light falling on a scene may be clustered to find faces; randomly structured light may allow blank featureless surfaces to be matched in stereo views; and so forth. Figure 35 demonstrates this principle.

3.6.3 Ultra-Sonics

Just as light can be pulsed to determine range, so can sound and ultrasound. The time between the transmitted and received signal determines range; the sound signal travels much slower than light, making the problem of timing the return signal rather easier than it is in pulsed laser devices. Figure 36 shows a typical depth scan obtainable from a commercially available U/S transducer.

The advantage of U/S sensing is that it is a very cheap way of obtaining a depth map. It has a number of disadvantages; high attenuation in air, specular effects on apparently smooth surfaces, and low data-acquisition rates. It’s primary use is in mobile robots, to locate obstacles and to navigate.
3.7 Sensor Integration

As is probably clear from the preceding discussion, any single sensor available to a robot sensor system, is limited in it’s ability to obtain all the information that is required for reliable intelligent robotics. It has long been proposed that information from many different sensors should be integrated to overcome the limitations inherent in any one sensor. Good anthropomorphic examples of this abound; from the use of multi-modal vision, to hand-eye coordination.

There are, essentially three different approaches to the sensor integration problem:

*Image-fusion methods* integrate information by direct fusion of intensity arrays, obtained from electro-magnetic scanning sensors such as CCD arrays. They are generally applicable to the problem of integrating information from different imaging sensors, such as vision and passive infra-red. Their main advantages stem from the direct use of tried and tested vision algorithms, and the fact that they place no constraint on representation or interpretation of sensed data. Their primary application area is in image-enhancement and image-based feature detection. Their outstanding disadvantage is their inability to deal with sensor information which is not intrinsically image-based. This precludes, for example, the use of contact or manipulation information from tactile or force sensors.

*Geometric integration methods* are motivated by the view that sensors can be regarded as “geometry extractors”; sources of partial, uncertain, geometric information an operating environment. The most important aspect of these techniques lies in their explicit use of geometry as a model of information. This provides a common language for the communication of information between different sensors, and allows both the tools of formal geometry and of statistical
decision making to be utilised in the processing, integration and interpretation of sensor information. The principle advantage of using geometry to model sensor information is that it allows all types of sensors to be considered in a common framework. However, this explicit use of geometric representation also gives rise to the primary disadvantage of these techniques: The imposition of specific geometric environment descriptions restricts the operation of such systems and makes them difficult to extend into complex environments, such as outdoor terrain navigation for example.

Logical, or knowledge-based integration methods embody the view that sensors can be considered as sources of knowledge about the structure of an operating environment. The key element of these techniques is to abstract the physical sensing process in terms of the information or knowledge they provide. This then allows the processing, interpretation and, most importantly, the control of information to be independent of the methods used to extract this knowledge. The principle advantage of these methods are a consequence of this abstraction; because the sensors are described in terms of information, a natural form of knowledge redundancy, and not physical redundancy, can be used. This is important because in seeking to provide a robust description of a sensed environment, it is the information and its interpretation which is important, and not always the physical sensing process. The primary disadvantages of knowledge-based integration methods are implicit; they must necessarily encompass one or more of the previous techniques to provide information, and the mechanisms, organisation and use of these systems are poorly understood.

Bibliography


4 Locomotion and Planning

The purpose of this lecture is to introduce the subject of planning; taking interpreted sensor information and performing actions with some specific system goal. Mobile robotics offers the best test area for planning problems; a mobile robots world is more unstructured than a workcell, and a greater variety of tasks can be performed. This begs planning systems, to come to grips with the realities of operating in uncertain and changing worlds; something traditional AI planners find difficult to do.

Mobile robot planning systems are typically composed of a geometric planner, able to decide on paths to avoid or approach geometric objects detected by sensors, and a task planner able to decide on sequences of actions and long-term motion analysis. The geometric part of the planner deals with obstacle avoidance, navigation and sensing; all areas where uncertainty is important. Given that the geometric planning stages can be accomplished reliably, the task-planner need only organise and sequence, deterministic subtasks.

We begin by describing some common locomotion systems, then describe geometric planning, obstacle avoidance and navigation, concluding with some aspects of task planning.

4.1 Mobile Robots

Mobile robots are becoming increasingly common in both research and industrial applications. The primary reason for this is the variety of uses to which they can be put, both commercially and for research purposes. A bewildering variety of mobiles have been developed, from tanks to running machines, from submarine robots to domestic cleaning machines. They use a variety of locomotion and sensing systems.

4.1.1 Industrial Systems

Current industrial mobile robots are typically used to move material around factories. Recent applications have also included security systems and cleaners, although it would be a stretch of the imagination to yet call these “industrial”.

Industrial AGVs either use wheels or tracks for mobility, and typically have no sensors other than collision switches. They are made to take pallets, or other well-defined objects from one work-cell to another. Planning in these systems is a problem in scheduling the transition of material.

These AGVs get from one place to another in the factory by one of two mechanisms; either by following wire-guides that are embedded in the floor, or by triangulating from known beacons placed in the factory area. Wire-guided AGVs are reliable, easy to programme, and follow fixed paths. The cost of embedding wires in the floor limits the flexibility of such systems. Beacon-guided AGVs are considerably more complex but offer a number of advantages over wire-guided systems in terms of flexibility; new routes can be added or old ones changed with relative ease. Figure 37 shows a commercially available beacon-guided AGV. It’s base is a milk-float construction, it triangulates by scanning a laser over bar-codes embedded in the factory walls.
4.1.2 Indoor Mobiles

Most research AGVs consist of a wheeled-base, motors and drivers, supporting a sensor platform and its associated computing hardware, which operate in an indoor environment. They are very often tethered, there purpose being to explore issues in mobile sensing and planning. Sensors typically include ultra-sonics, vision, infra-red, compass and gyros. Figure 38 shows three mobiles developed at MIT.

Figure 38: Three Mobile Robots Developed at MIT
4.1.3 Terrain mobiles

Terrain mobiles, which operate outdoors, are on a different scale completely. The mobile base is often a tank or troop-carrier, the size being dictated by the necessity of having all the computer hardware onboard. Sensors include laser-scanners and vision (no ultra-sonics), and often inertial or satellite navigation systems. Figure 39 shows the DARPA-ALV mobile developed at Martin-Marietta.

Human-operated terrain mobiles have also been developed. Although these are not truly autonomous, they play an increasingly important part in planning for unstructured environments. Figure 40 shows the next generation Mars rover designed at JPL.

![Diagram of DARPA ALV terrain mobile](image1)

**Figure 39: The DARPA ALV terrain mobile**

![Image of JPL Mars Rover](image2)

**Figure 40: JPL Mars Rover**
4.1.4 Legged systems

In very rough terrain, and in some indoor situations, legs have considerable advantages over wheels or tracks. A number of legged mobiles have been developed, with variable success. The sensors and navigational requirements are the same as those for wheeled mobiles, most work has been directed at the mechanics of legs and the problem of gait and balance. Walking machines have concentrated on solving the gait-control problem; relying on the coordinated motion of the legs while maintaining a stable stance at all times. This has produced usable machines, which walk in a funny way! Figure 41 shows a gait-controlled biped.

4.1.5 Running Machines

Running machines developed from taking a quite different approach to legged locomotion; taking balance to be all-important rather than gait. A number of hopping and running machines have been designed that seem to confirm this opinion. Figure 42 shows one such machine. The control of balance is achieved by trying to maintain the orientation of the body fixed in the horizontal plane with respect to the ground. Gyros are mounted on the body to monitor the absolute orientation of the body. Initial experiments with one-legged machines have extended well in to two and four-legged machines.
4.2 Navigation

The most important thing a mobile robot needs to know is where it is and where it needs to go. This is generally known as the navigation problem. It consists of three phases, obstacle avoidance—being able to move without hitting anything, localisation—working out where it is from observation of the local environment, and navigation being able to move from one known point to another known point, using the obstacle avoidance and localization functions.

4.2.1 Obstacle Avoidance

Obstacle avoidance requires that the AGV detect and avoid unknown or unmodeled obstacles in a preplanned path. The sensor most commonly employed to detect obstacles is ultra-sonics, because the obstacles need to be detected quickly, but no detailed information is needed about them.

There are two basic methods of obstacle-avoidance; certainty-grids and potential-fields. Certainty grids work by tessellating the workspace, and assigning probabilities, based on sensor information, as to each tessellation being occupied by an obstacle. Paths are modified by choosing only those that avoid areas of the grid with high certainty of occupancy. Figure 43 shows an example certainty grid generated from a rotating ultra-sonic sensor. Potential fields work by assigning a fictitious charge to each sensed data point, and using an inverse-square law, calculating the cumulative repelling force on the robot, providing a direction
Figure 43: A certainty grid map, used for obstacle avoidance, generated from ultra-sonic data

vector which is most likely to avoid an obstacle. The advantage of potential-field methods is that they are computationally cheap, their disadvantage is that potential wells are common.

4.2.2 Localisation

Localisation is the processes of finding out where you are. Typically this is done by observering various kinds of beacons and by comparing them with where they are expected to be, updating estimated location. Figure 44 demonstrates this process.

Figure 44: The localization process
The beacons used depend on the sensor, the simplest are active beacons, placed at known locations, more generally they can be characteristic features, observed by different sensors, and matched or correlated with mobile motion.

4.3 Planning

Planning is one of the basic techniques of Artificial Intelligence, and considerable research on this topic has been undertaken. Early research had been devoted to non domain-specific planning or to rather abstract problems (e.g., blocks world problems). Only recently have real-world problems received increasing attention. This development has been influenced by research in mobile robotics and expert systems.

Planning techniques can be roughly classified into three categories:

1. **Planning in a single abstraction space**: This is the simplest form of planning, in which all operators and world states are on the same level of abstraction. One type of planning in this form is logical inference, which tries to derive the goal from the initial situation using a set of operators. The plan is constructed from the sequence of operators. Another type of planning of this form is heuristic search within a state-space representation.

2. **Planning in multiple abstraction spaces**: The basic idea of this technique is to avoid details in an early phase of planning to reduce the complexity of the state-space. If a plan is found on a higher level of abstraction, it is planned in more detail on the next lower level. There are two principal ways to abstract the problem space: situation abstraction and operator abstraction.

3. **Meta-Planning**: Meta-Planning can be characterized as “planning how to plan”. This means that there is an additional level of planning where strategic decisions are made. This includes planning which abstract strategies to employ in the actual planning process. These planning strategies can be thought of as operators of the Meta-Planning level.

Aside from this AI-oriented research, there has not been much work on real-world robot planning. Early work consisted of parameterizing of prototypical strategies (so-called procedure skeletons). Such skeletons contain a framework of motions, error checks and computations for a particular type of task. The planner performs geometrical computations and error computations and decides which strategy to apply and how to parametrize this strategy. The emphasis of this research was algorithms for collision-free path-planning for mobile robots.

Planning moves is invariably geometric, often using some appropriate subset of CSG to define a path and detect collisions or available freeways. Figure 45 shows a typical “geometry engine” used for describing paths. These geometric path planners have much in common with those used for planning manipulator paths: the environment and mobile are modeled as CSG primitives.
Figure 45: A CSG model of AGV and environment

Bibliography

There are very few books on mobile robotics, and none worth reading. Some special-issues of journals have appeared though.


LIST OF PARTICIPANTS

LECTURERS

F. ALLEN, IBM Thomas J. Watson Research Center, Yorktown Heights, U.S.A.
C. BENNETT, IBM Thomas J. Watson Research Center, Yorktown Heights, U.S.A.
A. CASACA, INESC, Lisbon, Portugal
A. DANTHINE, Liège University, Belgium
H. DURRANT-WHITE, Oxford University, U.K.
S. FISHER, Rutherford Appleton Laboratory, Chilton, U.K.
C.N.P. GEE, Rutherford Appleton Laboratory, Chilton, U.K. (presently at CERN)
A.J.G. HEY, Southampton University, U.K.
C.A.R. HOARE, FRS, Oxford University, U.K.
G. KELLNER, CERN, Geneva, Switzerland
M. LETHEREN, CERN, Geneva, Switzerland
B. LEVRAT, Geneva University, Switzerland
M. METCALF, CERN, Geneva, Switzerland
K. MIURA, Fujitsu America, Inc., San José, U.S.A.
R.P. MOUNT, Caltec, Pasadena, U.S.A. (presently at CERN)
F. PERRIOLLAT, CERN, Geneva, Switzerland
R. PHILLIPS, Los Alamos National Laboratory, U.S.A.
J. RABAEEY, University of California, Berkeley, U.S.A.
J.J. THRESHER, CERN, Geneva, Switzerland
E.C. TRELEAVEN, University College London, U.K.
H. VAN DER BEKEN, JET Joint Undertaking, Abingdon, U.K.
STUDENTS

S.J. ALVSAVAAG, Bergen University, Norway
G. APPELQUIST, Stockholm University, Sweden
P. ARNOLD, Atlas Computer Centre, Chilton, U.K.
L.A.T. BAUERDICK, Mainz University, Fed. Rep. of Germany
(presently at CERN)
P. BELTRAN, N.R.C.P.S. "Demokritos", Aghia Paraskevi, Greece
T. BOTNER, CERN, Geneva, Switzerland
M. CAMPANELLA, INFN, Milan, Italy
A.J. CASS, CERN, Geneva, Switzerland
P. CREHAN, University College Dublin, Ireland
C. DEAN, SERC Daresbury Laboratory, Warrington, U.K.
P. DEFERT, CERN, Geneva, Switzerland
M. DIMOU, CERN, Geneva, Switzerland
R. DIVIA, CERN, Geneva, Switzerland
J. FANCHON, LPNHE, Ecole Polytechnique, Palaiseau, France
G. FARRACHE, Centre de Physique des Particules, Marseille, France
P. FOUCAULT, Centre de Recherches Nucléaires, Strasbourg, France
J.B. GALVAN HERRERA, Inst. de Estructura de la Materia, Madrid, Spain
M. GERRATSOUIS, N.R.C.P.S. "Demokritos", Aghia Paraskevi, Greece
L. GERLAND, Hamburg University, Fed. Rep. of Germany
(presently at CERN)
K. GÖRING, Inst.f. Mittelenergie-Physik, ETH, Villigen, Switzerland
E. HATZIANGELI, CERN, Geneva, Switzerland
B. HENNINGSEN, CERN, Geneva, Switzerland
R. HOPKINS, CERN, Geneva, Switzerland
S. KINDE, SERC Daresbury Laboratory, Warrington, U.K.
S. KOKKOTOS, N.R.C.P.S. "Demokritos", Aghia Paraskevi, Greece
M. KRÄMER, Technische Hochschule Darmstadt, Fed. Rep. of Germany
A. LEVY-MANDEL, FAVAG Microelectronics, Bevaix, Switzerland
P. MALECKI, DESY, Hamburg, Fed. Rep. of Germany
P. MALZACHER, Gesellschaft für Schwerionenforschung, Darmstadt, Fed. Rep. of Germany
A. MASONI, INFN, Cagliari University, Italy
P. MATO VILA, CERN, Geneva, Switzerland
A. MIOTTO, CERN, Geneva, Switzerland
M.-C. NGUYEN, Commission of the European Communities, Netherlands
L. NORMANN, CERN, Geneva, Switzerland
T. OKSAKIVI, Helsinki University, Finland
A. PACHECO, Universitat Autonoma de Barcelona, Spain
A.F. PEREIRA, CERN, Geneva, Switzerland
J.M. PEREZ, C.I.E.M.A.T., Madrid, Spain

472
M. PILAWA, CERN, Geneva, Switzerland
J. PUIMEDON, Zaragoza University, Spain
M. QUESADA, Santander University, Spain
P. RANDLES, Rutherford Appleton Laboratory, Chilton, U.K.
P. RIBARICS, Max-Planck-Institute, Munich, Fed. Rep. of Germany
A. ROLLNIK, Bonn University, Fed. Rep. of Germany
J. ROSE, RWTH Aachen, Fed. Rep. of Germany
M. ROSENBERG, University of London Computer Centre, U.K.
I. SCHNEIDER, Frankfurt University, Fed. Rep. of Germany
G.P. SIROLI, Bologna University, Italy
N. SPENCER, SLAC, U.S.A.
E. TANKE, CERN, Geneva, Switzerland
M. VARELA, CERN, Geneva, Switzerland
G. VEDOVATO, INFN, Legnaro, Italy
M. VIDAL, Max-Planck-Institute, Munich, Fed. Rep. of Germany
A. WALRAEVENS, Université Libre de Bruxelles, Belgium
X.J. WANG, Institute of Atomic Energy, Beijing, People´s Republic of China (presently at INFN, Legnaro, Italy)
M. WARNS, Bonn University, Fed. Rep. of Germany
O. WILLM, CERN, Geneva, Switzerland
N. ZIOGAS, CERN, Geneva, Switzerland