1993 CERN SCHOOL OF COMPUTING

Scuola Superiore G. Reiss Romoli, L'Aquila, Italy
12-25 September 1993

PROCEEDINGS
Eds. C.E. Vandoni, C. Verkerk

GENEVA
1994
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 desins ou modèles susceptibles de dépôt qui pourraient être décrites 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.

© Copyright CERN, Genève, 1994

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.

ISSN 0304-2898
ISBN 92-9083-065-4
ABSTRACT

These Proceedings contain the majority of the lectures given at the 1993 CERN School of Computing.

Artificial neural networks were treated with particular emphasis on applications in particle physics. A discussion of triggering for experiments at the proposed LHC machine provided a direct connection to data acquisition in this field, whereas another aspect of signal processing was seen in the description of a gravitational wave interferometer.

Some of the more general aspects of data handling covered included parallel processing, the IEEE mass storage system, and the use of object stores of events. Lectures on broadband telecommunications networks and asynchronous transfer mode described some recent developments in communications.

The analysis and visualization of the data were discussed in the talks on general-purpose portable software tools (PAW++, KUIP and PIAF) and on the uses of computer animation and virtual reality to this end. Lectures on open software discussed operating systems and distributed computing, and the evolution of products like Unix, NT and DCE.
PREFACE

The 1993 CERN School of Computing was the sixteenth in the series. For technical reasons it took place for the second time in succession in the "Scuola Superiore Guglielmo Reiss Romoli" in L'Aquila, Italy. Forty-nine students from fourteen countries took part.

The participants enjoyed again the excellent facilities offered by the "Scuola Superiore G. Reiss Romoli". We express our sincere thanks to its Director, Professor S. Rotella and his deputies Dr. G. Ciccarella and Dr. B. Santariga, for having made the facilities of their School available to us. We are very grateful to Messrs. Collabianchi and Dundee for their extensive technical assistance in setting up the 3-D video projection. Mrs. Ingiacomo and Messrs. De Meo and Aglioti also provided invaluable help.

The lecture programme was again much appreciated by the participants. All lecturers are sincerely thanked for their efforts in preparing and delivering their lectures, for providing - in most cases - written accounts of their presentation, and for their availability to discuss with the students.

Very special thanks are due to Mr. Michael Bauer, who, in addition to his lectures, made a very impressive three-dimensional video presentation on applications of virtual reality at the NASA Ames Research Center. He spent a large amount of time, together with Messrs. Collabianchi and Dundee, to perfect the set-up. Their efforts were greatly appreciated by the students, who asked for a "bis".

Silvano de Gennaro led the practical exercises on "Techniques for making presentations". The equipment for the exercises was made available at very short notice by Mr. A. Santariga from Computime, Rome. We thank him warmly. We are very grateful to Silvano, not only for his tutorial and exercises, but also for the musical entertainment he offered in the evenings.

An excursion to the nearby underground laboratory of the Gran Sasso was appreciated by the participants.

Everybody has doubtlessly good souvenirs of the excellent room and restaurant facilities of the Reiss Romoli Centre. The manager, Mr. Bardelli, and all his staff are sincerely thanked for having taken such good care.

We express our sincere gratitude to Mrs. Ingrid Barnett, for her secretarial assistance. During many years she executed her secretarial duties with great competence and dedication. We wish her success in her new job!

Finally, we thank the students for their active participation and we wish them success in their professional life.

C.E. Vandoni,
C. Verkerk,
Editors.
1993 ADVISORY COMMITTEE

R. BRUN
CERN, Geneva, Switzerland

B.C. CARPENTER
CERN, Geneva, Switzerland

S. CENTRO
Università di Padova, Italy

R.F. CHURCHHOUSE
University of Wales, Cardiff, U.K., (Chairman)

R.W. DOBINSON
CERN, Geneva, Switzerland

C. VERKERK
CERN, Geneva, Switzerland (Scientific Secretary)

D.O. WILLIAMS
CERN, Geneva, Switzerland

P. ZANELLA
Services Informatiques, Geneva, Switzerland and Centro di Ricerca, Sviluppo e Studi Superiore in Sardegna, Cagliari, Italy

I. BARNETT
CERN, Geneva, Switzerland (School Secretary)

EDITORS' NOTE

The material presented by Mr. Michael Bauer in his lectures on "Computer animation for scientific visualization" and on "Virtual reality; case studies at NASA Ames" are reproduced here in three papers: two (co-)authored by Mr. Bauer and one by Dr. S. Bryson.

To their great regret, the editors did not succeed in obtaining written contributions from a number of lecturers, in spite of repeated efforts.
CONTENTS

Abstract ................................................................. iii
Preface ................................................................. v
Advisory Committee and Editors' note ...................................... vi
Table of contents ........................................................... vii
Lecturers ............................................................... viii

Neural networks, S. R. Amendolia ........................................ 1
Databases and parallel processing, N. Winterbottom .................. 36
The IEEE mass storage reference model, J. Shiers ....................... 44
Working with object stores of events using ptool, R. Grossman ....... 66
Level-1 triggers for LHC experiments, E.F. Eisenhandler ............... 98
Visualization of scientific data for high energy physics, R. Brun et al. 121
Computer animation for scientific visualization, M. E. Bauer and K. C. Hu 179
Virtual reality - A general overview, M. E. Bauer ....................... 207
Virtual reality in scientific visualization, S. Bryson ................. 217
Open software: UNIX, DCE, and Competitors, T. W. Doeppner .... 228
Broadband telecommunications networks and ATM, G. Ciccarella and T. Tofoni 249
Searching for gravitational waves, B. F. Schutz ....................... 274

List of participants (students) .............................................. 295
1993 LECTURERS

S.R. AMENDOLIA
University of Sassari and INFN of Pisa, Italy

G. BATE
Institute for Information Technology, Santa Clara University, U.S.A.

M.E. BAUER
Sterling Software, Palo Alto, CA, U.S.A.

T. BERNERS-LEE
CERN, Geneva, Switzerland

R. BRUN
CERN, Geneva, Switzerland

G. CICCARELLA
Scuola Superiore G. Reiss Romoli, L'Aquila, Italy

M.S. DE GENNARO
CERN, Geneva, Switzerland

T.W. DOEPPNER
Brown University, Providence, U.S.A.

C. ECK
CERN, Geneva, Switzerland

E.F. EISENHANDLER
Queen Mary and Westfield College, University of London, U.K.

M. ERNST
DESY, Hamburg, Germany

R. CROSSMAN
Laboratory for Advanced Computing, University of Illinois at Chicago, U.S.A.

A.J.G. HEY
University of Southampton, U.K.

C.H. LLEWELLYN SMITH
CERN, Geneva, Switzerland

J.F. MORRISON
Los Alamos National Laboratory, U.S.A.

B.F. SCHUTZ
Department of Physics and Astronomy, University of Wales, U.K.

J. SHIERS
CERN, Geneva, Switzerland

N.M. STEPHENS
Department of Computing Mathematics, University of Wales, U.K.

N. WINTERBOTTOM
University of Southampton, Parallel Applications Centre, U.K.
1993 CERN SCHOOL OF COMPUTING

Scuola Superiore G. Reiss Romoli, L’Aquila, Italy
12-25 September 1993

PROCEEDINGS
Eds. C.E. Vandoni, C. Verkerk

GENEVA
1994
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 desains 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.

© Copyright CERN, Genève, 1994

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.

ISSN 0304-2898
ISBN 92-9083-065-4
1993 CERN SCHOOL OF COMPUTING

Scuola Superiore G. Reiss Romoli, L'Aquila, Italy
12-25 September 1993

PROCEEDINGS
Eds. C.E. Vandoni, C. Verkerk

GENEVA
1994
ABSTRACT

These Proceedings contain the majority of the lectures given at the 1993 CERN School of Computing.

Artificial neural networks were treated with particular emphasis on applications in particle physics. A discussion of triggering for experiments at the proposed LHC machine provided a direct connection to data acquisition in this field, whereas another aspect of signal processing was seen in the description of a gravitational wave interferometer.

Some of the more general aspects of data handling covered included parallel processing, the IEEE mass storage system, and the use of object stores of events. Lectures on broadband telecommunications networks and asynchronous transfer mode described some recent developments in communications.

The analysis and visualization of the data were discussed in the talks on general-purpose portable software tools (PAW++, KUIP and PIAF) and on the uses of computer animation and virtual reality to this end. Lectures on open software discussed operating systems and distributed computing, and the evolution of products like Unix, NT and DCE.
PREFACE

The 1993 CERN School of Computing was the sixteenth in the series. For technical reasons it took place for the second time in succession in the "Scuola Superiore Guglielmo Reiss Romoli" in L'Aquila, Italy. Forty-nine students from fourteen countries took part.

The participants enjoyed again the excellent facilities offered by the "Scuola Superiore G. Reiss Romoli". We express our sincere thanks to its Director, Professor S. Rotella and his deputies Dr. G. Ciccarella and Dr. B. Santariga, for having made the facilities of their School available to us. We are very grateful to Messrs.Collabianchi and Dundee for their extensive technical assistance in setting up the 3-D video projection. Mrs. Ingiacomo and Messrs. De Meo and Aglioti also provided invaluable help.

The lecture programme was again much appreciated by the participants. All lecturers are sincerely thanked for their efforts in preparing and delivering their lectures, for providing - in most cases - written accounts of their presentation, and for their availability to discuss with the students.

Very special thanks are due to Mr. Michael Bauer, who, in addition to his lectures, made a very impressive three-dimensional video presentation on applications of virtual reality at the NASA Ames Research Center. He spent a large amount of time, together with Messrs. Collabianchi and Dundee, to perfect the set-up. Their efforts were greatly appreciated by the students, who asked for a "bis".

Silvano de Gennaro led the practical exercises on "Techniques for making presentations". The equipment for the exercises was made available at very short notice by Mr. A. Santariga from Computime, Rome. We thank him warmly. We are very grateful to Silvano, not only for his tutorial and exercises, but also for the musical entertainment he offered in the evenings.

An excursion to the nearby underground laboratory of the Gran Sasso was appreciated by the participants.

Everybody has doubtlessly good souvenirs of the excellent room and restaurant facilities of the Reiss Romoli Centre. The manager, Mr. Bardelli, and all his staff are sincerely thanked for having taken such good care.

We express our sincere gratitude to Mrs. Ingrid Barnett, for her secretarial assistance. During many years she executed her secretarial duties with great competence and dedication. We wish her success in her new job!

Finally, we thank the students for their active participation and we wish them success in their professional life.

C.E. Vandoni,
C. Verkerk,
Editors.
1993 ADVISORY COMMITTEE

R. BRUN
CERN, Geneva, Switzerland

B.C. CARPENTER
CERN, Geneva, Switzerland

S. CENTRO
Università di Padova, Italy

R.F. CHURCHHOUSE
University of Wales, Cardiff, U.K., (Chairman)

R.W. DOBINSON
CERN, Geneva, Switzerland

C. VERKERK
CERN, Geneva, Switzerland (Scientific Secretary)

D.O. WILLIAMS
CERN, Geneva, Switzerland

P. ZANELLA
Services Informatiques, Geneva, Switzerland and Centro di Ricerca, Sviluppo e Studi Superiore in Sardegna, Cagliari, Italy

I. BARNETT
CERN, Geneva, Switzerland (School Secretary)

EDITORS' NOTE

The material presented by Mr. Michael Bauer in his lectures on "Computer animation for scientific visualization" and on "Virtual reality; case studies at NASA Ames" are reproduced here in three papers: two (co-)authored by Mr. Bauer and one by Dr. S. Bryson.

To their great regret, the editors did not succeed in obtaining written contributions from a number of lecturers, in spite of repeated efforts.
CONTENTS

Abstract ........................................................................................................................................ iii
Preface ................................................................................................................................................ v
Advisory Committee and Editors' note ............................................................................................. vi
Table of contents ............................................................................................................................. vii
Lecturers .......................................................................................................................................... viii

Neural networks, S. R. Amendolia .................................................................................................. 1
Databases and parallel processing, N. Winterbottom .................................................................... 36
The IEEE mass storage reference model, J. Shiers ......................................................................... 44
Working with object stores of events using ptool, R. Grossman .................................................. 66
Level-1 triggers for LHC experiments, E.F. Eisenhandler ............................................................. 98
Visualization of scientific data for high energy physics, R. Brun et al. ........................................ 121
Computer animation for scientific visualization, M. E. Bauer and K. C. Hu .............................. 179
Virtual reality - A general overview, M. E. Bauer ....................................................................... 207
Virtual reality in scientific visualization, S. Bryson ...................................................................... 217
Open software: UNIX, DCE, and Competitors, T. W. Doeppner ............................................... 228
Broadband telecommunications networks and ATM, G. Ciccarella and T. Tofoni .................... 249
Searching for gravitational waves, B. F. Schutz .......................................................................... 274

List of participants (students) ........................................................................................................ 295
1993 LECTURERS

S.R. AMENDOLIA
University of Sassari and INFN of Pisa, Italy

G. BATE
Institute for Information Technology, Santa Clara University, U.S.A.

M.E. BAUER
Sterling Software, Palo Alto, CA, U.S.A.

T. BERNERS-LEE
CERN, Geneva, Switzerland

R. BRUN
CERN, Geneva, Switzerland

G. CICCARELLA
Scuola Superiore G. Reiss Romoli, L'Aquila, Italy

M.S. DE GENNARO
CERN, Geneva, Switzerland

T.W. DOEPNER
Brown University, Providence, U.S.A.

C. ECK
CERN, Geneva, Switzerland

E.F. EISENHANDLER
Queen Mary and Westfield College, University of London, U.K.

M. ERNST
DESY, Hamburg, Germany

R. CROSSMAN
Laboratory for Advanced Computing, University of Illinois at Chicago, U.S.A.

A.J.G. HEY
University of Southampton, U.K.

C.H. LLEWELLYN SMITH
CERN, Geneva, Switzerland

J.F. MORRISON
Los Alamos National Laboratory, U.S.A.

B.F. SCHUTZ
Department of Physics and Astronomy, University of Wales, U.K.

J. SHIERS
CERN, Geneva, Switzerland

N.M. STEPHENS
Department of Computing Mathematics, University of Wales, U.K.

N. WINTERBOTTOM
University of Southampton, Parallel Applications Centre, U.K.
Neural Networks

S.R. Amendolia
University of Sassari and INFN of Pisa, Italy

Abstract

An introductory treatment of the subject of Neural Networks will be given. Topics covered will mostly be relevant for the use of Neural Networks in High Energy Physics, especially for triggering, and examples will be given of this application.

1. Basics of Neural Networks

1.1 Introduction

There exist many papers and books on the subject of Neural Networks in the literature, and this cannot be an attempt at making a better or more comprehensive treatment. We send to the relevant references [1,2,3,4,5]. The difficulty in this work, compared to previous treatments, is that the title does not quote a specific application of the Neural Networks, but deals with the entire matter instead. Since however the lectures were given to an audience of Physicists and Computer Scientists mostly involved in High Energy Physics related jobs, we shall narrow the scope and concentrate on aspects which can be useful in this domain of Science. Examples too will come from this field.

1.2 Complexity of tasks and structure of the human brain

It is customary to start treatments on Artificial Neural Networks (ANN) making a comparison with the complexity of the human brain. The name ANN itself is used as a reminder of the fact that we are dealing with an emulation of the brain. The differences mostly lie in the complexity of the structures. It is a common belief that the description of the biological neuron is well matched by the artificial one used in the ANN (ref. sections 1.4 and 1.5). As we shall see, this is a very simple processing unit, which performs a "sum" of the input stimuli and applies a threshold to it, releasing an appropriate signal to the other units connected. This happens via electro-chemical processes in the brain, and is emulated by electronics in the hardware implementations of the ANN's. The speed for a "neuron operation" in the former case is of the order of milliseconds, while is of the order of nanoseconds in a VLSI implementation.

Despite this disadvantage in speed, the ability of the brain at performing pattern recognition tasks is unsurpassed (while it is quite limited in sequential computations) compared to that one of a ANN. The computing power derives from the enormous number of neurons which are present in the brain (order of $10^{11}$), compared to a ANN (order of $10^2$, $10^3$), and on the high level of connectivity of the biological neuron ($10^2$ to $10^4$ connections per neuron), which is on the contrary limited in an artificial implementation. So the brain behaves as a very powerful Parallel Distributed Processor. The resulting computing power for the human brain can be estimated of the order of $10^{15}$ operations/sec, with a memory capacity of $10^8$ Mbytes. In real life applications, to overcome the above limitations, the ANNs are used to solve specific problems, which usually require the use of few neurons. Even though, their capability and performance in tasks which are hard for sequential computers and algorithms, is astonishing. We shall try in the following to explain the origin of this fact.

It is worth noticing that Neural Networks represent the current frontier of the developments in Artificial Intelligence. They are a manifestation of the connectionist
approach to computing, as opposed to the semantic one. The latter is better represented by Expert Systems, which require as input a certain knowledge basis, including rules. Neural Networks are the expression of an attempt at emulating the physical structure of the brain, more than at explaining the way in which concepts are stored and worked upon in the mind. The hope is that from the co-operation of a very large ensemble of simple units (the neurons), a global "intelligent" behaviour will show up.

1.3 Introducing the neuron

From the point of view of the architecture, an ANN is a set of single processing elements (called neurons) connected together, and thus exchanging information. There exists a "coupling constant" between any two connected neurons. Given the neurons \( i \) and \( j \), this coupling is called the weight \( w_{ij} \). Each neuron \( i \) is characterised by its state \( s_i \) and by its activation function \( f \).

![Neuron Diagram](image)

**Fig. 1:** the "dot-product" neuron

One of the models [6] most commonly used to describe the neuron is the so-called "dot-product" neuron (Fig. 1). In this model, a given neuron \( i \) performs the weighted sum of the outputs of the neurons connected to it:

\[
P_i = \sum_j w_{ij} \cdot s_j
\]

This can be obviously read as the dot-product of a weight vector and an output vector, formed respectively with the weights and with the outputs connected to the neuron \( i \), namely:
\[ \overrightarrow{W}_i = \{w_{ij}\}; \overrightarrow{S} = \{s_j\}; \overrightarrow{P}_i = \overrightarrow{W}_i \cdot \overrightarrow{S} \]

Each neuron acts on this weighted sum and produces an output (its actual state) according to its activation function:

\[ s_i = f(P_i) \]

There exist several possible forms for the activation function, usually they consist of applying a threshold and doing a sort of amplification. We will be mostly concerned with three types of functions, a simple step-like threshold cut, a linear function, and the most important of all, the non-linear, sigmoid-shaped function.

This model of the neuron is derived from biology: the synapsis plays the role of the connection between the dendrites and the neuron body: so the "weight" is in the synapsis, while the axons carry the information from a neuron to another.

Another model of neuron which we shall encounter is the so-called "distance" neuron (Fig. 2). This neuron computes, instead of the weighted sum, some sort of distance between the weight vector and the state vector outlined above; a typical distance function is the euclidean norm:

\[ d = |\overrightarrow{W}_i - \overrightarrow{O}| \]

![Diagram of neuron](image)

**Fig. 2**: the "distance" neuron

In many applications, the activation function, applied to this distance, reflects the notion that the neuron output is higher the higher is the similarity between the weight and the input vectors, so for example the function is bell-shaped, centred on distance equal to
zero. We shall come back on this when dealing with ANNs based on the competitive learning algorithm.

The ANN is completely described when the architecture is known, i.e. the number of neurons and the way in which they are connected together (ref. section 2.1), and when for each neuron the activation function and the operation mode are fixed. The values of the weights are usually obtained through a "learning" process (ref. section 3.1).

1.4 History of Neural Networks

This is not intended to be a serious history of this rapidly developing branch of research. We want just underline few milestones [7], relevant to the understanding of the progress in the field, and of the difficulties encountered. The dot-product model of the neuron of the previous section has been introduced in 1943 by McCulloch and Pitts. At the same time, Hebb suggested a rule to explain the learning mechanism which characterises the operation of a Neural Network (in primis, of the human brain): the idea was that the connection between any two neurons has to be reinforced when a high activity in one neuron is associated to a high reaction in the other one. This simple rule (which seems to have a biological justification) is at the base of some of the most known mechanisms for the learning of a ANN (including the Delta Rule, ref. section 3.2).

The first ambitious attempt at building a real machine based on this description of an artificial neuron has been the Perceptron, by Frank Rosenblatt of Cornell, in 1957. This was a pattern classification system to identify both abstract and geometrical patterns. It was robust, rather tolerant to noise and damage, and exhibited some learning capabilities (we shall devote a large amount of this paper at discussing the learning issue in relation with the Neural Networks). The Perceptron was primarily aimed at optical pattern recognition. The first models were made up with two layers of neurons, an input layer (connected to the detecting devices) and an output dot-product-neuron layer. The activation functions in these early models were linear.

This architecture, as we shall see in the following, lead to some strong limitations in the flexibility of the system, which was not able of performing more than linear separations between the input patterns' configurations. This has been the basis for a very strong critics of the early Perceptron, carried out by Minsky and Papert in 1959, which caused the cut of the fundings to these researches and the demise of many projects.

During the 70's, there has been a certain activity, mainly concentrated in the study of mechanisms which allowed a Neural Network to learn and thus reproduce patterns, without an external supervisor. The work by Teuvo Kohonen, and the mechanism of the competitive learning, were among the most relevant issues (ref. section 5).

In 1982, a renowned physicist, John Hopfield of Caltech, proposed a model of associative memory, based on a network of fully connected neurons. The cumulative behaviour of such a network carried close resemblance with the behaviour of physical structures, like the spin glasses. This, the rigorous treatment of the problem made by Hopfield himself, and the subsequent extensions to inglobe statistical mechanics concepts in the theoretical study of these networks, brought new light into the subject, and gave rise to a resurrection of interest in the field of Neural Networks (ref. section 4).

In 1985, Rumelhart and co. at MIT developed a method to perform the learning also on multilayer perceptrons, which were known by the time to be able to perform more than linear separations in the input parameter space, and thus able to do higher lever classification tasks (ref. sections 2 and 3).
Since then, the connectionist approach to Artificial Intelligence, based on Artificial Neural Networks, grew of importance exponentially with time.

2. Characteristics of Neural Networks

2.1 Types and architectures of Neural Networks

When dealing with ANN's, and with their cumulative behaviour and learning procedures, we tend to classify them on the basis of their structure. Two of the most commonly used ANNs in Physics are the feed-forward and the recurrent configurations. Fig. 3 shows a schematics of both.

![Feed-Forward Network and Recurrent Network](image)

Fig. 3: ANN architectures

The name feed-forward networks (FF-ANN in the following) derives from the fact that neurons are divided in separate layers, and neurons in each layer contribute to the input of neurons in the next layer only. Neurons are of the dot-product type. There is no interconnections between neurons inside one layer. We usually distinguish layers between input layer (connected to the external world, and receiving the external "stimulus"), output layer (producing the output result which is "sensed" by the external world), and hidden layers (hidden from the external world), whose role we will investigate in the next sections. This kind of networks is sometimes referred to also as "multilayer Perceptrons".

Recurrent networks (R-ANN in the following) on the other hand exhibit a fully interconnected structure: any neuron is in principle connected to any other (the weight eventually carries the information that two neurons are not interacting with each other, and in this case it would end up being zero in the learning process). It is evident how such a structure allows feed-back between neurons. These can be classified again as I/O neurons, and hidden neurons. Note that here a neuron can be at the same time an input and an output neuron. Implications of these facts will be discussed and will become clear in section 4.

A typical application of these networks is as an associative memory. If the input neurons are different from the output ones, the ANN will produce an output which is related (associated) to the configuration shown at the input; the input and the output
spaces being in principle different. This is an etero-associative network. If the input and output neurons coincide, the network is said to be auto-associative: in this case, it can be used to "complete" an input pattern which has been shown at the input, and which can be affected by noise or with missing information.

It should be noted that this classification of the architectures reflects only the most used networks in the domain of Physics, which is the one we are interested in. Combinations of these architectures are equally important, thus care should be taken in forcing the ANNs into one or the other category. For example, feed forward networks with a layer of neurons fully connected between themselves, and interacting after the competitive learning rule, are the customary architecture for unsupervised, classification networks; also, FF-ANN with a final autoassociative layer of fully interconnected neurons proved extremely useful in understanding some forms of brain damage like dislexy [8].

2.2 Information storage

As in every computer, information (or better knowledge?) is stored inside an ANN. In a conventional machine, any datum is recorded in a precise memory location, and uniquely identified by its address. In an ANN, two basic types of information storage apply. One is when a piece of information is univoquely associated with the state of a neuron (local information storage): then the weights correspond to the precise relationship which has to exist between any two pieces of data, and which would determine, in a conventional algorithmical approach, the computational procedure. In this case, the weights can be computed from outside, and the Network performs as an advanced parallel processor. We will see this behaviour in some examples of recurrent networks.

Another occurrence is when the data are shared among two or more neurons, and each neuron itself can contribute to represent different pieces of information (distributed information storage): though one can in principle trace back the role of each neuron, the impressive task of finding out the best values for the weights is carried out through a "learning" procedure.

2.3 Parallel with conventional approaches in Science

In order to keep track of the parallelism between ANN operation and conventional algorithms, we discuss the way we perform classifications in Physics (and in other domains of Science as well). Doing a classification (i.e. calling a certain set of physical variables' values as the representation of a particular "class" - an object, a feature, a type of event,...) goes through the application of cuts [9]. These can be linear and non-linear ones.

A linear cut, for a one dimensional distribution of a physical variable, is equivalent to a step function $\Phi$ (see Fig. 4a)

$$\Phi(\text{variable} - \text{threshold})$$

where

- $\Phi=0$ for variable $<$ threshold
- $\Phi=1$ for variable $\geq$ threshold
We call this a linear classifier. Fig. 4b shows a diagram of the logical operation performed: the bubble which provides the output stands for an operator which applies the step function to the input, which in turn is a linear combination of the outputs coming from the input bubbles. The coefficients ("weights") of this linear combination are shown on the lines connecting the bubbles. The bubble labelled THRESH always gives a "1" as output.

For two or more variables, the linear cut can work only if the categories are linearly separable. The choice of the primitive variables is not necessarily the best one, and a "rotation" in the hyperspace of the input variables may be needed. This is shown in Fig. 5a, where the linear classifier which discriminates class "A" from class "B" in a 2-dimensional problem is the step function

\[
\Phi(aX + bY + c)
\]

with a,b,c constants. The straight line in the figure has equation \(aX + bY + c = 0\). Fig. 5b shows the equivalent logic diagram.
2.4 Significance of hidden neurons

We saw in the previous section how the occurrence of non-linear cuts could be overcome using an approximation of the boundary curve through linear cuts. This required the use of logical computational structures similar in architecture to a multilayer feed-forward ANN.

![Non-linear separation in XOR boolean function](image)

This is connected to the fact that a single multi-input boolean neuron cannot implement all the possible boolean functions.

A n-dimensional boolean neuron is a neuron with a step-like activation function which performs an application from an input set of n boolean variables \( \{x_1, x_2, x_3, ..., x_n\} \) to the set \( \{0, 1\} \).

Even for the simple case of the XOR function, where the input space dimension is 2, a neuron is not enough to provide the result, since the two classes of input boolean values corresponding to the two possible outputs are not linearly separable (Fig. 7 - the two input variables are called A and B in the figure). One needs a neuron at least in a hidden layer, contrary for example to the OR and AND functions, which can be represented by the same boolean neuron, where only the threshold value needs to be changed.

Another way \([1]\) of getting a feeling at the relevance of hidden units (this name is often used for neurons when dealing with ANNs) is in term of statistical information carried by the Network, something which is related to the amount and the way in which data are stored in the Network.

Independently of the structure, consider a Network with a number \( v \) of visible (i.e. input/output) units, which evolves (reacts) when a vector is presented at its input terminals \([6]\). The first order statistical information carried by the ANN is related to the probability that a single unit be on; this is described by the thresholds (\( v \) numbers). The second order statistical information is related to the correlation between any two visible units (vector components); this is described by the weights (\( v^2/2 \) numbers).
As an example, assume that the possible patterns of input/outputs configurations in a ANN with binary neurons are the following four 3-dimensional vectors (so, three visible units):

\[(1,1,0) - (1,0,1) - (0,1,1) - (0,0,0)\]

These vectors are only a subset of the possible 8 vectors which one can build with three binary components. So the fact that only these vectors constitute the "environment", i.e. the set of meaningful existing configurations, is not explained by the statistical information carried by the visible units: they appear with a even probability \(p = 1/N = 1/4 = 0.25\), much as it would happen in the larger 8-vector set (where \(p = 1/8 = 0.125\)), and each component has a 0.5 probability to be present, as it would be in the larger sample. There must be some extra "knowledge" in the structure if only these four vectors "exist". This can be solved introducing a fourth unit (a hidden unit), not related to the external world, but driven by the values of the other visible units. The vectors are then described by four components:

\[(1,1,0,1) - (1,0,1,0) - (0,1,1,0) - (0,0,0,0)\]

where the fourth unit is required to be the AND of the first two, thus carries the information that both the two first units are ON; it acts as a "feature detector". This is the fundamental role of hidden units in ANN structures: they capture higher order statistical features of the input space set. It is easy to understand that the example above is precisely the XOR case, where units 1 and 2 are the input boolean variables, unit 3 is the output, and unit 4 is the unavoidable hidden neuron.

2.5 Properties of Neural Networks

We shall discuss some of the most important properties of an ANN, resulting from their architecture and from their operation mechanism. Some of these concepts have already been introduced in the previous sections, other will become even more evident when we shall analyse the way ANNs work, and when we shall show examples.

* Distributed associative memory

Distributed memory means that the values of the weights represent the status of knowledge of the ANN, without a specific association of a piece of knowledge to a particular neuron (refer to the discussion above about information storage, section 2.2). Associative means that if the ANN is shown a partial input, the network will choose the closest match in memory to that input, and generate an output corresponding to the full input. If the network is autoassociative, this results in the completion of the input vector. Thus ANN's can handle incomplete, noisy or previously unseen data: this last feature is called generalisation; this is mostly evident in non-linear multilayer networks.

* Fault tolerance:

Destruction or alteration of one or more processing elements causes only a slight degradation of the network performance: this is a consequence of the distributed information storage. This is to be opposed to the functioning of conventional computers, where the failure of a single element causes a failure of the entire computational process, with the highest degree of damage. ANN are thus extremely well suited for applications where failure of control equipment means disaster (nuclear power plant, missile guidance, space probe operation). High Energy Physics applications are more questionable, to the extent that the advantage of an algorithmical approach can sometimes provide a clearer understanding of the systematic errors. We shall expand on this point in section 6.5.

* Pattern recognition:

Pattern recognition tasks require the ability to match large amounts of input information simultaneously and to generate categorical or generalised output (plus response to noise or incomplete input). The ANN is intrinsically a massively parallel computational
device with the above characteristics. It is well known indeed (as stated in section 1.2) that the superior skills of the human brain for image treatment reside in its largely parallel structure. ANNs, with due respect to the different size of the implementation compared to the brain, try to emulate its more intriguing features as a pattern

In traditional expert systems the knowledge is made explicit in the form of rules. There is no learning properly said. ANN's on the contrary learn by being shown examples, i.e. via the stimulation of the input/output units by externally provided patterns. Learning is the ensemble of the procedures through which the weights in an ANN are adjusted. There are two distinct ways in which learning can take place:

- Unsupervised learning: only input stimuli are shown to the network, which organises itself so that each processing element responds strongly to a different set of stimuli or group of stimuli. These sets represent clusters in the input space, usually related to distinct real world concepts.

- Supervised learning: recognition device.

3. Operation of Neural Networks

3.1 Learning types and rules

for each input stimulus, a desired output configuration is presented to the network's output neurons, and the weights are changed so as to achieve the desired input/output mapping.

It is customary to talk of an external "teacher" in the case of the supervised learning, i.e. some external entity which knows the desired output and corrects the ANN to achieve it. Actually, since these learning procedures are a set of rules usually translated in a software code, the external action is implicit in both techniques. Beyond these procedures, a special case are the recurrent ANN's, in which the weights can be imposed from the outside: this usually happens when these R-ANN are used for the task of finding the optimal solution to a problem (ref. section 4.2), and in this case it would be improper to speak of learning.

During the learning phase, the weights change as a function of time:

\[ w_{ij}(t) \rightarrow w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \]

The most important rules for weight adjustment are:

- Hebbian learning: a weight increases in proportion to the product of the activation status of the two neurons involved:

\[ \Delta w_{ij} = \eta \ s_i(t) \ s_j(t) \]

This reflects the obvious notion that a coupling constant has to be larger if there is a strong coupling between the input stimulus and the output reaction. In biological terms, this means that a neural pathway is strengthened each time it's used.

- Delta rule learning:
based on reducing the error between the output of a processing unit and the desired output. This rule implements a gradient descent along the error function, defined in the space of the weights. It will be thoroughly treated in the following sections.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

- Competitive learning:
  processing elements compete among each other and the ones which yield the strongest response to a given input modify the weights related to them to become more like that input.

$$\Delta w_{ij} = \eta (X - W_i) \quad \text{where}
\begin{align*}
X &= \text{input to units } j \text{ (a vector)} \\
W_i &= \text{weights from units } j \text{ to unit } i \\
&\quad \text{(a row of the matrix } W_{ij})
\end{align*}$$

3.2 The Delta Rule

The Delta Rule is a paradigm of supervised learning, normally applied to FF-ANN (there exists an extension to the case of recurrent ANNs which can be found in [10]).

Let's consider a two-layer FF-ANN. The neurons in the first layer (the input one) just

Beyond competition, another kind of operation can take place among neurons of the same layer: normalisation, which takes the vector of values corresponding to the layer's output and scales them so that the "total" output is some fixed value. In biological systems this is done connecting the layer's elements among themselves (they sense the total layer's output and adjust accordingly).

repeat the input values and propagate them to the second layer (Fig. 8). Assume to present a set of input patterns \(\{A_p\}\) to the input layer of the network. The index \(n\) ranges on the number of patterns. These patterns are \(n\)-dimensional vectors, where \(n\) is the dimension of the input space. For each of these patterns, the network produces an output vector \(O_p\) (the dimension of the output needs not be the same as that one of the input, the two spaces are in general different). And for each input pattern, assume that the desired target output \(T_p\) is known.

![Fig. 8: Supervised learning (two layer FF-ANN)](image)
Then a signed difference $O_p(i) - T_p(i)$ can be computed for each component of the output vector $O_p$, and we can define a quadratic form measuring the mismatch between the output $O_p$ and the target $T_p$:

$$E_p = \frac{1}{2} ||O_p - T_p||^2 = \frac{1}{2} \sum (O_p(i) - T_p(i))^2$$

where the summation extends over the index $i$. $E_p$ is thus proportional to the square of the Euclidean distance between output and target.

For the entire set of patterns, we define a global error function $E$ which measures the quality of the approximation to the set $\{T_p\}$ given by the set $\{O_p\}$:

$$E = \sum \limits_p E_p$$

$E$ is a quadratic form of the weights, and its minimum corresponds to the optimal configuration for the network. The Delta rule consists in modifying the weights proportionally to the rate of decrease of the error function with respect to that particular weight, i.e. a gradient descent which can be written:

$$\Delta w(i,j) = -\eta \frac{\partial E}{\partial w(i,j)}$$

where $\eta$ is a parameter.

Let's recall here that the output of a dot-product neuron is a function of the weighted sum of the inputs:

$$O_p(i) = f[\sum w(i,j) A_p(j)] = f[s_p(i)]$$

where we have defined $s_p(i) = \sum w(i,j) A_p(j)$.

With some simple mathematics, we get for the gradient of the error function:

$$\frac{\partial E}{\partial w(i,j)} = \sum \frac{\partial E_p}{\partial w(i,j)}$$

$$= \sum \frac{\partial E_p}{\partial O_p(i)} \cdot \frac{\partial O_p(i)}{\partial s_p(i)} \cdot \frac{\partial s_p(i)}{\partial w(i,j)}$$

$$= \sum (O_p(i) - T_p(i)) \cdot f'(s_p(i)) \cdot A_p(j)$$

where the summations extend over the index $p$. Thus the weight change can be written:

$$\Delta w(i,j) = \sum \delta_p w(i,j)$$

$$= -\eta \sum (O_p(i) - T_p(i)) f'(s_p(i)) A_p(j)$$

where we defined: $\delta_p(i) = (O_p(i) - T_p(i)) f'(s_p(i))$.

This is known as the Widrow-Hoff formula. Note that the form of the variation of the weight induced by the presentation of pattern $p$ (i.e. $\Delta w(i,j)$) reminds of the Hebb rule, in the sense that the weight change depends on the strength of the "cause" (the value of the component of the input vector which is related to that particular weight, $A_p(j)$), and on the strength of the "effect" (in this case the value of the mismatch between output and target, $\delta_p(i)$) This term includes the derivative of the activation function, since it gives a measure of the intensity of the reaction of the output neuron to its input (the weighted sum $s_p(i)$).

- Linear units
This is the trivial case of a so-called linear associator. Note that in this case (a two-layer FF-ANN with linear units), the network's operation is a simple matrix multiplication. If, instead of two, there would be many linear-unit layers, each of them would perform a matrix multiplication, and the operation of the whole network would be the product of these matrices, thus again a matrix multiplication.

In case of linear units, the activation function \( f \) of each neuron is a multiplication by some constant number, i.e.

\[
O_p(i) = \text{constant} \times s_p(i).
\]

Thus the derivative of the activation function appearing in \( \Delta w(i,j) \) is just a constant number. Most often, for simplicity, this constant is taken to be 1; in this case, the term \( \delta_p(i) \) represents exactly the difference between output and target vectors' i-th components. Note that if a threshold \( \theta \) is present (i.e. if \( s_p(i) = \Sigma w(i,j) A_p(j) + \theta(i) \)), this can be learned as any other weight. The error function \( E \), in case of linear units, has only a global minimum.

- Non-linear units

The fact that the first derivative of the function \( f \) appears, prevents the use of this learning rule when neurons with a discontinuous activation function are present (e.g. the threshold neurons, ref. section 2.3).

There is a variety of functions which can be used, though the most common one is the sigmoid, given by:

\[
f(s) = \frac{1}{1 + e^{-s}}
\]

which ranges between 0 (\( s \to -\infty \)) and 1 (\( s \to +\infty \)).

Note that in the applications where the output neurons should reach a binary state the sigmoid, due to the property that \( f'(s) = f(s) \cdot (1 - f(s)) \), tends to induce stronger changes for the units which are more "undecided" (i.e. values next to 0 or 1 - Ref. Fig. 8). This adds to the stability of the system. For more on the sigmoids ref. to [11].

![Fig. 9: sigmoid derivative](image)

3.3 Multi-layer Networks and Backpropagation

We applied the Generalised Delta Rule to the output units of a two-layer FF-ANN, for which the difference from a target pattern is readily computed. For multi-layer networks, we need to define a correction procedure for the weights connecting the input neurons to the neurons in the hidden layer, or between hidden layers if there are more than one; these hidden layers' neurons do not communicate with the external world, so we are left with the problem of computing the "error" associated with the output of these units. In order to do that, we still start from the gradient descent, applied to a three-layer network (Fig. 10): now the weights in the gradient computation are those linking units from the first layer to, e.g., neuron \( i \) in the hidden layer.
Fig. 10: Backpropagation

The gradient descent now reads:

\[
\Delta w(i,j) = -\eta \sum \frac{\partial E_p}{\partial w(i,j)}
\]

\[
\frac{\partial E_p}{\partial w(i,j)} = \frac{\partial E_p}{\partial O_p(i)} \cdot \frac{\partial O_p}{\partial s_p(i)} \cdot \frac{\partial s_p}{\partial w(i,j)}
\]

= \frac{\partial E_p}{\partial O_p(i)} \cdot f'(s_p(i)) \cdot A_p(j)

but the term \(\frac{\partial E_p}{\partial O_p(i)}\) no longer is related directly to a difference output vs. target, since neuron \(i\) does not "see" the target. To compute this term, we expand it in terms of the errors computed for the output layer:

\[
\frac{\partial E_p}{\partial O_p(i)} = \sum_k \frac{\partial E_p}{\partial O_p(k)} \cdot \frac{\partial O_p(k)}{\partial s_p(k)} \cdot \frac{\partial s_p(k)}{\partial O_p(i)}
\]

= \sum_k \delta_p(k) \cdot w(k,i)

remembering the definition for \(\delta_p(k)\), so that:

\[
\Delta w(i,j) = -\eta \sum \left\{ f'(s_p(i)) \cdot A_p(j) \cdot \left[ \sum_k \delta_p(k) \cdot w(k,i) \right] \right\}
\]

= \sum \eta \sum \delta_p(i) \cdot A_p(j)

where the "generalised" error \(\delta_p(i)\) is a weighted sum of the errors in the subsequent layer, times the derivative of the activation function.

In a multilayer network, the formula can be applied to any hidden layer, simply the input \(A\) will be substituted by the output of the previous layer. This algorithm is known as "backpropagation", since the error computed at a downstream layer is propagated backward to the more upstream ones.

3.4 Problem of minima and strategies

The Generalised Delta Rule with backpropagation, in the case of non-linear units, still implements a gradient descent in the weight space, but there are several possible minima of the error function. An empirical way out of this difficulty is to choose appropriately the learning parameter \(\eta\), even modifying it during the learning phase.
For example, when the gradient descent is started, higher values of $\eta$ ensure that the descent will take place fast enough, but when a minimum is reached, lower values of $\eta$ are needed to adjust the weights with the optimal precision. Also, when a minimum is reached, if a rather large variation is generated, this would lead the error function to jump out of the local minimum area (if this was the case); then the descent towards the global minimum can resume. There is no exact rule about this.

Also, to speed up the descent, especially in the first learning iterations, a momentum $\alpha$ is used, which "remembers" the fact that the weight updating in the previous iteration was still intense, meaning that one is far from the minimum and can move faster. So the actual prescription for the weight updating becomes:

$$\Delta_n w(i,j) = - \eta \frac{\partial E_n}{\partial w(i,j)} + \alpha \Delta_{n-1} w(i,j)$$

where $n$ indicates the $n$-th iteration.

The initial values of the weights and of the thresholds are randomly chosen, to avoid biasing the result.

Often, instead of computing the error function for the entire set of input vectors, and then performing the weight updating, one makes the updating after a certain smaller number of vector presentations; it can be proved that this causes a slight deviation from the pure gradient descent, which does not seriously affect the convergence of the method towards a minimum. An iteration on the entire set of input vectors is called an "epoch".

3.5 Learning and testing

When a FF-ANN is trained with backpropagation, the learning is usually carried on a sequential computer, where the ANN is simulated in a high level code.

First, one has to divide the input patterns (and consequently the corresponding target patterns) in two sets, the so-called training sample and the testing sample.

The first phase of the learning process consists of the presentation of the input patterns from the training sample, and the application of the algorithm to compute the change of all the weights in the network. This is iterated until the error reaches a minimum (for example, the error variation is below a predefined value). This is a lengthy process, since quite often one has to loop many thousands on the input set before the weights settle. Techniques exist to accelerate the process [12].

The validation of the values of the weights found during the first learning phase is done measuring the performance of the Network on the patterns of the testing sample, by construction different from the ones used for the learning phase.

There is no precise rule as to how many patterns must be used in the learning sample. On a heuristic basis, it is advisable to have a number of input patterns few times the number of weights to be optimised. This poses, especially in practical applications where the input patterns cannot be generated via computer simulations, the limit of keeping the ANN architecture as small as possible. Also, there is no rule to determine how many hidden units are necessary (this is rather easily determined only for the input and output layers, where the number of units is driven by the number and kind of the input data and by the kind of answer the network has to provide). The number of hidden units is usually determined on a trial basis, as the minimum number which gives good performances. If too many hidden units are present in a network used for classification tasks, the risk arises of overtraining the network, i.e. the ANN starts "memorising" the input patterns: in this case, the performance on the learning sample is extremely good, but it degrades unacceptably on any test sample. This is because the networks knows the input set "by heart", and is no longer capable of generalising, i.e. of associating previously unseen patterns to a specific class which it should have learnt to identify.
3.6 Quality of classification using Neural Networks

The optimal classifier is a Bayes classifier: in a classification problem between two classes of events A and B, for example, the Bayes classifier gives the probability of belonging to either class. This is accomplished finely binning the ranges of the input variables, and defining the probability for an event falling in a bin to belong to class A as the total number of events of class A in that bin divided by the total number of events in the bin, and vice versa for class B. Assume that a FF-ANN is used for classifying between the two classes, and there is one output neuron which has to reach value 1 if the input event belongs to class A, and value 0 if it belongs to class B [9]. Then the learning is performed minimising the error function:

\[ E = \frac{1}{2} \sum_i [O(i) - T(i)]^2 \]

where the summation is on the events in the learning sample. Remembering that T can only have values 1 or 0, being \( \alpha_A \) and \( \alpha_B \) the percentages of events of class A and B in the sample, \( P_A(i) \) and \( P_B(i) \) the probabilities for event i to belong to class A or B, the error function can be written:

\[ E = \sum \{ \alpha_A \ P_A(i) \ [O(i) - 1]^2 + \alpha_B \ P_B(i) \ O(i)^2 \} \]

Minimum condition is obtained through differentiation with respect to \( O(i) \), equating to zero:

\[ \alpha_A \ P_A(i) \ [O(i) - 1] + \alpha_B \ P_B(i) \ O(i) = 0 \]

\[ O(i) = \frac{\alpha_A \ P_A(i)}{\alpha_A \ P_A(i) + \alpha_B \ P_B(i)} \]

\[ O(i) = \frac{n_A}{n_A + n_B} \]

Thus finding the minimum of the error function corresponds to mapping each event to its bayesian probability of being in class A. This is the best job one should try to obtain from a Neural Network. In effect a three-layer ANN trained with backpropagation approximates a Bayes classifier [13], the accuracy depending on the number of hidden units. Referring to the non-linear cuts discussed in section 2.2, the Neural Network with continuous activation functions (sigmoids), instead of selecting a multidimensional region with a sequence of linear cuts, approximates on the input variable space the probability of belonging to a specific region.

3.7 Simple example with Feed-Forward ANN

An instructive example is the FF-ANN which is trained to compute the Square Root operator. This FF-ANN operates on any two numbers in the range [0,1], and tells if the two numbers are one the square root of the other. This way of using a Neural Network is called "function mapping", but we will see that it is indeed equivalent to a classification network, where the two classes are the one made of the pairs of numbers which are in square root relation, and the one of those which are not. The architecture of the network is in Fig. 11a. Fig. 11b shows the values of the output neuron as a function of the two input values, and Fig. 11c makes evident the contribution of the five hidden units to the result. Each of these units specialises in providing a definite cut. More hidden units could have been used in this case, improving the Network performance. Understanding which cut every neuron provides, or which feature of the
Fig. 11a: SQRT Problem: ANN architecture

Fig. 11b: approximation to SQRT function by ANN

Fig. 11c: SQRT problem: cuts operated by the hidden units
problem it is specialised to recognise, is a difficult issue, when raised in concurrence with more complex networks.

4. Recurrent Neural Networks

Fig. 12 shows the architecture of a Recurrent ANN (R-ANN); in this particular example, input and output units coincide. Hidden units can of course be present.

![Recurrent Neural Network Diagram](image)

Fig. 12: Recurrent Network (from [9])

In a R-ANN there is no direction of flow of the information, thus feed-back between neurons takes place. A R-ANN, after having been stimulated with an input configuration, settles at a stable value for its neurons' activation state. For a R-ANN there exist learning techniques [1,10], but quite often the Network is used as a sort of "Parallel Computer" to provide optimal solutions to problems, especially NP-complete ones. We shall give a brief survey of the fundamental aspects of R-ANN's, more can be found in specialised literature.

4.1 Hopfield associative memory

This is a R-ANN introduced for the first time by J. Hopfield in 1982, and is made of \(n\) neurons fully connected among themselves, each neuron having a threshold \(\theta_i\); the connections are symmetrical \((w_{ji} = w_{ij})\); there are no auto-connections \((w_{ii} = 0)\). The total number of weights is \(n(n-1)/2\). The network is binary, i.e. the neuron states \(s_i(t)\) at any time \(t\) can only have value 0 or 1. The activation law for each neuron, governing the transition from \(s_i(t)\) to \(s_i(t+1)\) is given by:

\[
\begin{align*}
    s_i(t+1) &= 1 & \text{if } P_i = \sum w_{ij} s_j(t) - \theta_i \geq 0 \\
    s_i(t+1) &= 0 & \text{if } P_i < 0
\end{align*}
\]

The state of the network at time \(t\) is \(S(t) = s_1(t), s_2(t), ..., s_n(t)\) The network evolves from an initial state \(S(0)\) to a final state \(S(f)\) of stable equilibrium. Indeed, to each state one can associate an "energy" function:

\[
E(S) = -\frac{1}{2} \sum_{ij} w_{ij} s_i s_j + \sum_j \theta_j s_j
\]
It is easy to show that \( \partial E/\partial s_i = -P_i \) so that, given the updating rule written above, any change in a neuron's state results in a change \( \Delta E \) in the total energy which is never positive, i.e. \( \Delta E \leq 0 \), as it should be if we are approaching an equilibrium state. Actually, the network evolves towards a relative minimum of this energy function. One can associate a pattern to be memorised to each of these minima, thus the network, if stimulated with an input configuration (read: a state \( S(0) \)) close enough to one of these minima, will evolve and settle in this closest minimum. The fact that the network will converge to a specific pattern even if the input is not complete or if some noise exists in the initial state, is what makes this R-ANN an "associative memory".

The function \( E(S) \) is an iper-surface in the \( n \) dimensional space of the states of the neurons; some of the minima of this surface correspond to the patterns memorised, but others may exist, which do not correspond to a meaningful pattern (though being stable configurations): this may happen if too many patterns are memorised in the network. There is no precise rule to determine how many patterns can be memorised by such a R-ANN, since it depends on the quality of the "separation" which one wants to achieve between similar patterns, i.e. on the precision in recollecting a given pattern from an associated input configuration. Typically a ratio \( m/n \) of the order of 0.14 is the edge beyond which the network sensibly degrades in performance (\( m \) being the number of patterns memorised, \( n \) the number of neurons).

4.2 Hopfield network and optimisation problems

One of the most interesting applications of the Hopfield R-ANN is in the solution of NP-complete problems, i.e. problems whose complexity (proportional to the time needed for the solution) grows more than polynomially as a function of the dimension of the problem. Combinatorial problems found in certain physics applications (ref. section 6.4) fall in this category.

Expanding the line of thought of the previous section, the idea is that a R-ANN can be equated to a dynamical system, whose behaviour is described by a set of general coordinates (the activation states of its neurons). A point in state space represents the instantaneous condition of the system. The equations of motion of the system describe a flow in state space. One can associate to the states of the system an energy function such that the trajectories in state space lead to a global minimum of the energy. If this energy function is directly related to the cost function of an optimisation problem, then the state of minimum energy corresponds to the best solution for the problem. The system architecture itself is a description of the problem in these cases. Determining the form of the energy-cost function is the same as determining the values of the weights, which in this case are no longer learnt via a training procedure. This energy-cost function has to take into account the constraints of the problem (these can be weak or strong constraints: strong constraints are automatically satisfied by the trajectories in state-space, while weak constraints appear in the energy function as parameters behaving in the minimisation process much like Lagrange multipliers).

Once a quadratic form \( E \) in the activation states \( s_i(t) \) is found, this can be used to determine \( P_i(t) = -\partial E/\partial s_i \), and this \( P_i \) will be the argument of the transfer function \( f \) of each neuron:

\[
s_i(t+1) = f(P_i(t)).
\]

Here \( f \) is the step function. Starting from an initial configuration \( S(0) \) of the system, the process is iterated until the energy \( E \) stays constant for few successive iterations. This approach however does not prevent the system from settling into a local minimum of the energy function. A way out of this is the use of the "simulated annealing" process.
4.3 The simulated annealing

This technique is mediated from the physics of spin glasses, and it basically introduces a probabilistic component in the treatment of the otherwise purely deterministic Hopfield network described so far. The technique was introduced for the first time by Hinton, Sejnowsky and Ackley in 1984. It is worth noticing that the rigorous mathematical description of a R-ANN evolving via the simulated annealing process is closely related to the mathematics of systems in statistical mechanics: these R-ANN are thus often called Boltzmann Machines. The justification of the method in terms of the Mean Field Theory approximation of statistical mechanics to the partition function of a probabilistic system can be found elsewhere.

We shall give a simple explanation of the method, derived by [5], where also applications, among others, to the Travelling Salesman Problem and to the Graph Partition Problem can be found. The method consists of allowing some transitions of the neuron states, induced by the threshold condition

$$P_i \geq 0 \implies s \rightarrow 1 \quad P_i < 0 \implies s \rightarrow 0$$

even if the criterion of the energy decrease is not exactly satisfied. More precisely, if:

- $\Delta E \leq 0$ the transition is accepted
- $\Delta E > 0$ the transition is accepted with a probability $p = \exp(-\Delta E/T)$

where $T$ is a parameter (called temperature). This means that given any two states of energy $E_1$ and $E_2$, the relative probabilities of existence are in the ratio

$$\exp[-(E_1 - E_2)/T]$$

which is the Boltzmann distribution, from which the name temperature.

Such a technique allows the system to wander around in state-space more than if the deterministic approach of section 4.2 is followed. In particular, the higher the temperature, the more "wrong" transitions are allowed: this has the consequence that the system can (probabilistically) jump away of trajectories leading to local minima, and then resume its motion towards a (possible) global minimum of the energy.

Referring to the optimisation problem discussed in section 4.2, the simulated annealing can be realised defining an energy-cost function $E$ in the same way as before, computing the $P_i$, and applying the transfer function $f$ which is no longer a step function, but a "sigmoid" with parameter $T$, i.e.:

$$s_i(t+1) = f(P_i(t)) = 1/(1 + \exp(-P_i(t)/T))$$

The process is repeated with increasingly lower values of the temperature, always looking for a stable minimum of the energy. Note that in the limit $T \rightarrow 0$, the sigmoid above becomes a step function.

5. Unsupervised networks

We shall only give a partial description of unsupervised Neural Networks, i.e. ANN which learn following an unsupervised learning technique. This, as already stated, means that the external "supervisor" does not provide a target vector together with the input, thus the network has to optimise the values of the weights accordingly to a
different criterion, basically it tries to extract features inherent in the input sample. The fundamental learning rule used is the "competitive learning".

5.1 Competitive learning

This requires a layer of neurons which have lateral connections, as for example in Fig. 13; this layer is called a "feature layer". A neuron, or a set of neurons, specialises in the recognition of some patterns. This happens through a competition among the neurons. Namely, for each pattern presented, \( \bar{x} = (x_1, \ldots, x_k) \), a winning neuron in the feature layer is selected, which has the highest activation state. The procedure is called the "winner-takes-all" procedure. Be the winning neuron the \( m \)-th one \([14]\), then:

\[
s_m = \max_j(s_j)
\]

\[\text{Fig. 13: Competitive learning}\]

Given the set \( M \) of the input vectors which give the \( m \)-th neuron as winner, an error can be defined as:

\[
E_m = \frac{1}{2} \sum_M (\bar{x} - \bar{w}_m)^2
\]

where the weight vector is \( \bar{w}_m = (w_{m1}, \ldots, w_{mk}) \). The gradient descent rule gives:

\[
\Delta w_{mk} = -\eta \frac{\partial E}{\partial w_{mk}} = \eta \sum_M (x_k - w_{mk})
\]

where \( \eta \) is a learning parameter. A normal quicker practice is to make the weight updating each time a vector is presented, thus:

\[
\Delta w_{mk} = \eta (x_k - w_{mk})
\]
Convergence is obtained modifying the learning parameter, which has to decrease slowly with time (i.e. with iterations). A normalisation process usually follows this learning phase, such the weights are normalised to a unit norm. Then at the end of the process, the weight vectors mimic the structure of the input vectors, and can be analysed accordingly. See the examples in section 5.2.

5.2 Kohonen maps

They are a typical implementation of the concepts defined above. They can be mono- or bi-dimensional. Remember that the feature layer has lateral connections, i.e. there are weights connecting the neurons in this layer: these weights are not adjusted in the learning process, but are fixed and constitute a characteristic of the map. They define the extension of the region (bubble) around the winning neuron, which is also affected "positively" by the winner-takes-all process, while neurons outside this bubble compete with the candidate winner. These weights are thus a function of the distance between neurons in the same layer, and they can be both excitatory and inhibitory. The "mexican hat" distribution is a good example of a the weights' variation as a function of distance from a given neuron (which is the "centre" of the bubble); ref. Fig. 14.

During the learning phase described above, all the neurons belonging to the bubble of the winning neuron change their weights accordingly to the gradient descent rule outlined above. If they do not belong to the bubble of the winning neuron, no action is taken on their weights. The size of the bubble should also be decreased with time: large bubbles allows a larger flexibility, but narrower bubbles give a larger selectivity.

![Interaction](image)

**Fig. 14: lateral neuron interaction**

6. Real time applications

This section is devoted to the description of some applications of the Neural Networks to High Energy Physics (HEP) which are suitable of hardware implementations, and
Fig. 15: 3-layer cylindrical chamber typical event (real data)

Fig. 16: drift chamber subdivision in subsections each subsection $\leftrightarrow$ one FF-ANN
thus could be used in the trigger system of an experiment. Here the requisites are robustness, unbiased operation (a wrong decision at the trigger level results in the unretreivable loss of a potentially interesting event), and speed.

6.1 Interaction vertex finding

This application of FF-ANN to HEP consists of finding the primary interaction vertex in a high energy collision between two particles\[15.\] The case was given by experiment E-735 at Fermilab, where a proton (p) and an anti-proton (p\bar{p}) collide head-on to produce an event made of many outgoing charged particles. While the transversal position of the vertex is well known due to the limited size of the colliding beams, the longitudinal coordinate z is uncertain by tens of centimetres. A better knowledge would allow a prompt rejection of events due to interactions not in the fiducial region, and would give a handle for more refined trigger algorithms at the second trigger level.

The detector involved is a three-layer cylindrical drift chamber close to the beam, outside the beam pipe (Fig. 15). Real data from physical interactions were fed into a simulated Neural Network to study the feasibility of such a device. The task of the Network is then to reconstruct tracks. The chamber (3 x 108 drift wires) was divided in 18 subsections, each made of 6 wires on the three layers. Tracking in each subsection was made by a 3-layer FF-ANN (Fig. 16), which consequently had 18 neurons at the input layer. The input data were the drift times. Experimentally, it has been found an optimal configuration with 128 hidden units (Fig. 17). The output layer had to code the z-coordinate of the origin of the straight lines recognised by the Network, corresponding to the trajectories of the charged tracks in that subsection (in presence of an axial magnetic field, trajectories in the r-z plane next to the interaction region are well approximated with straight lines). The range of ±30 cm. for the interaction point was coded by 60 output neurons (corresponding to a 1 cm. binning in z), plus 2 neurons for underflow and overflow. In order to achieve a better resolution, the actual z-vertex position in the learning phase was described with a gaussian histogram, the z-coordinate being the mean of the histogram, and its width being the experimental resolution. Thus, the output of the 18 ANNs were histograms, which were summed linearly: the resulting histogram is centred at the position of the primary interaction vertex. The resulting vertex position agrees with the more accurate off-line reconstruction (Fig. 18), and gives a spatial resolution three times better than other online methods based on time-of-flight information. Using existing ANN hardware (e.g. the INTEL ETANN chip\[16\]), the system could be implemented as part of a hardware second level trigger producing an output in times of order 5-8 μsec.

6.2 Muon track trigger

The importance of a fast accurate tracking is stressed in trigger applications where the knowledge of the precise transverse momentum p_t of a particle allows to lower the threshold above which an event triggers the apparatus. Muons are typical charged particles which provide a strong signature for heavy flavour decay. The possibility of implementing an hardware muon tracking trigger based on Neural Networks has been investigated, and a test set-up using an ETANN chip has been operated in a test beam at Fermilab \[17\]. Fig.19 shows the experimental set-up, where three planes of drift chambers each with two wires constituted the tracking device. The advantage of using an ANN is that the analog drift times can be used, thus providing a more accurate angle measurement than using only the wire number.

Each pair of sense wires provides a drift time (converted to a voltage value through a TVC) and a latch voltage identifying the upper or lower wire. Of the 64 output units, 32 provide the slope of the track (in bins of 50 mrad) and 32 the intercept (in bins of 0.625 cm). Sixty-four hidden units were required; using fewer units degraded the performance of the Network. The Network was trained (in an off-line simulation) via backpropagation, using 10,000 tracks generated by Montecarlo. As for the primary
Target Distribution:

Output Distribution:

Output Units:

Hidden Units:

Input Units:

Wire #:

Fig. 17: FF-ANN architecture for interaction vertex finding

Fig. 18: residual from comparison of vertex position from offline fit with: a) FF-ANN c) TOF method
Fig. 19: test-beam set-up for muon tracking with VLSI ANN

Fig. 20: typical tracks reconstructed by the Network (real events)
vertex trigger, the target patterns were gaussian histograms (one for slope, one for intercept) with a r.m.s. = 1 bin. The weights so calculated were downloaded into the actual electronic circuit (an ETANN chip) through an ETANN Development System, and a further adjustment was performed with a chip-in-the-loop process. This way, the specific features of the analog ETANN chip were accounted for.

Results from operation of the system on real tracks in the test set-up give a position resolution of 1.2 cm, which is only twice the best one obtained by an off-line analysis, and 4 times lower than that one obtained with conventional triggers. The time required to output the result is ~8 μsec. Fig. 20 shows some typical events. In one of them, two tracks were reconstructed by the Network, both being compatible with the recorded drift times, due to the classical left-right ambiguity of drift chambers.

6.3 CDF isolation trigger

This trigger is being implemented as an hardware contribution \cite{18} to the second level trigger of CDF (p-pbar collision experiment at the Fermilab Tevatron). The existence of an isolated electron (outside a jet shower) is a strong signature for W boson decay. The Network in this case acts as a fast parallel processor: the electromagnetic calorimeter's surface is unrolled in a η-phi plane, where calorimeter towers are cells of 15° x 0.2 η. Clusters of energy release were identified by a cluster finder, and the calorimetric content of regions of 5x5 cells around the cluster centre was fed into the ANN.

![Diagram](image)

**Fig. 21: CDF isolation trigger FF-ANN structure**

Fig. 21 shows the Network configuration, with 25 input neurons, 4 hidden units, and one output neuron giving the isolation flag. The 4 hidden units were specialised in recognising a structure similar to one of the 4 templates in Fig. 22.

![Templates](image)

**Fig. 22: isolation templates**

The Network computes the energy release in the inner (shaded) area, and in the outer area of the 5x5 region, and flags the electron as "isolated" if in at least one hidden unit one has

\[ f \cdot E_{\text{inner}} - E_{\text{outer}} > 0 \]
From training, the best value for $f$ was found to be 0.16, in good agreement with the best values obtained from off-line analysis. Implementing this trigger in hardware using the ETANN chip allows to perform isolation cuts at the second level instead of third level trigger, lowering the energy threshold for electrons and reducing at the same time the background by a factor 4.

6.4 Secondary vertex trigger with a Feed-Forward Network

A challenging task in High Energy Physics trigger is the selection of heavy flavour production (B meson). In fixed target experiments, the goal is particularly ambitious since the cross section is extremely low compared to the minimum bias production. An attempt [19] has been made at selecting events with a long-lived meson decay from background minimum bias events, using a FF-ANN. The only information used was the charged track parameters (slope and intercept), which comes from a dedicated hardware tracking system [20]. The goal is to select the interesting events on the basis that they have secondary vertices (the B-meson decay vertices) distinct from the primary interaction vertex. From Fig. 23 it is seen how tracks coming from the same vertex can be described by the equation:

$$D = R \cdot \sin \left( \arctan \frac{X_Y}{Z_Y} - \Phi \right)$$

or, for small angles, 

$$D = -R \cdot (\Phi - \text{const})$$

![Diagram showing secondary vertex trigger]

**Fig. 23: from X-Z space to D-Φ space**

So lines in the D-Φ plane become points, and vertices become lines, with a negative slope ($R$, the decay length, is always positive). Fig.24 shows D-Φ plots for some typical events, obtained through a Montecarlo simulation of 450 Gev/c proton interactions on a thin nuclear target with tracks measured by a Silicon Microstrip Telescope, which gave only digital (i.e. no pulse height) information. Events with B-meson production show points (i.e. tracks) offset with respect to the horizontal primary vertex line ($R=0$), which is on the other side the normal appearance of the background events.

Problems arise from the fact that the track finding is performed by an hardware device (the Associative Memories, fully simulated in the Montecarlo data production), which are subject to the creation of spurious tracks, and have a limited track finding efficiency (order of 85%) in this experimental configuration. Thus, in the D-Φ plots, tracks can be missing which would help in easily identifying the signal event, or fake tracks can make a background event look similar to a signal one. Since the track finding process
Fig. 24: D-Φ plots in XZ and YZ projections for two signal and two background events

Fig. 25: Identification efficiency vs. cut on the output (fraction of output full scale)
takes place independently in the two $xz$ and $yz$ projections, information from both projections were used.

An ETANN chip was used, working as a FF-ANN with 64 input units, 64 hidden units and 8 output units. The meaning of the 64 inputs was:

- **1-16** $D$ of the first 16 tracks in the $xz$ projection
- **16-32** $\Phi$ of the first 16 tracks in the $xz$ projection
- **33-48** $D$ of the first 16 tracks in the $yz$ projection
- **49-64** $\Phi$ of the first 16 tracks in the $yz$ projection

$D$ and $\Phi$ values were normalised in a (-1,1) range, and tracks beyond the 16th were discarded. The output units coded the result as:

- (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1) for signal (SIG) events, and
- (-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1) for background (BG) events

This way of coding (instead of using only one bit to distinguish between two classes) can average out problems with the analog components of the ETANN chip: each neuron in a group of four was actually trained to give the same answer, -1 or +1.

Montecarlo events were divided in 4000 BG + 800 SIG for training, and 1000 BG + 200 SIG for testing. The software training required about 2 million iterations. After this, a *chip-in-the-loop* training was done, to adjust the weights according to the actual variations of the analog signal inside the chip; 12,000 iterations were enough to reach the same performance of the software simulation. The following table reports the results after a "strict" cut (i.e. requiring that both the SIG and the BG output neurons agree with the target):

<table>
<thead>
<tr>
<th>Simulation</th>
<th>good</th>
<th>bad</th>
<th>ambiguous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>57%</td>
<td>37%</td>
<td>6%</td>
</tr>
<tr>
<td>Background</td>
<td>92%</td>
<td>6%</td>
<td>2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ETANN</th>
<th>good</th>
<th>bad</th>
<th>ambiguous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>48%</td>
<td>48%</td>
<td>4%</td>
</tr>
<tr>
<td>Background</td>
<td>92%</td>
<td>6%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Fig.25 shows the efficiency of the trigger as a function of the cut applied on the output SIG neurons only (this is the "limited" criterium, opposed to the "strict" above). Results indicate that the method allows a reduction of a factor 20 in the background with an efficiency of about 40% on the signal.

6.5 Tracking with a recurrent ANN

This is a typical off-line application in which a R-ANN is used as a parallel processor to solve an optimisation problem. There are no hidden units in the Network, whose task is to identify curved tracks from a bi-dimensional set of points ($R$-$\Phi$ plane), measuring the hit co-ordinates of charged tracks bent by an axial magnetic field inside a Time Projection Chamber (TPC). First attempts were tried in parallel by Denby and Peterson [21], the former using Montecarlo generated data from the Delphi TPC. The neurons in the Network are meant to represent the links between any two hits: thus the number of neurons grows quadratically with the number of hits (this being one the draw-backs of the method). The energy-cost function is defined as:

$$E = \frac{1}{2} \sum_{i,j} w_{ij} o_i o_j$$
The weight matrix is symmetrical, non-diagonal; the weights' values (and consequently the cost function) are defined on the basis of the geometrical properties of the problem (Fig.26).

![Diagram](image)

**Fig. 26: parameters for weight definition in tracking with R-ANN**

For two links sharing one hit head to tail:

\[ w_{ij} = A \cos \frac{\theta_{ij}}{L_i L_j} \]

where \( n \) was determined by trial, and \( L_i, L_j \) are the "lengths" of the two links, while for the two links head to head or tail to tail,

\[ w_{ij} = -B \]

This form for the weight favours smooth curves. A simulation of this Network (i.e. a computer program which minimises the energy-cost function) shows that about 5-8 iterations are enough to reach stable values for the activation states of the neurons. Fig.27 shows the status of the neurons at different steps in the evolution of the Network. This approach works satisfactorily up to charged track multiplicities of order 15, and is particularly suited for applications in which the tracks cannot be easily parametrized (e.g. non-uniform magnetic fields). It does not however take in account all the features of the tracks (e.g. the helical form of the curve).

6.6 Pro's and Con's of Neural Networks for triggering

The application of ANN's to triggering problems has several advantages:

- the highly non-linear cuts operated, which extend the scope beyond the conventional threshold-and-cut approach obtainable with discriminators
- the capability of operating as an associative memory (i.e. the skill to classify structures even starting from incomplete data), which lowers the sensitivity to the detector's inefficiencies
- the tolerance to noise
- the speed, due to the intrinsic parallelism of the Network, and to the easy implementation as electronic VLSI circuits
Fig. 27: evolution of Recurrent ANN for tracking in a Time Projection Chamber (all neurons in active state at the beginning of the iteration)
• the programmability, which allows a quick adaptation of the Network to new running conditions or to new trigger requisites

As a result, triggers implemented with ANN's are candidates for operation at Level 2. The ANN's allow to perform selections which are customarily made through algorithms running in Level 3 trigger processors. This sophisticated kind of triggers become then affordable at a level one step closer to the Front End Electronics. The immediate advantage is the higher interaction rates manageable by the system.

Disadvantages in the use of ANN's for triggering are a less evident interpretation of the "cuts" applied, and of the ability of the Network to approximate the best classifier.

When the training, as in most applications, has to rely on Montecarlo generated events, the dependence on the model chosen can introduce biases. And if the supervised training can use real data (for example, data which have been classified independently with lengthy analysis procedures), the problem arises of using consistent sets of input information, if real data are mixed with Montecarlo data. This leads to the question whether the Montecarlo is a faithful interpretation of reality.

This problem is far less important however when the ANN's are used for low level pattern recognition applications. In these cases, the quality of the Montecarlo simulations is usually quite high (shower simulations, detector response, etc.), thanks also to the experience gained in detector construction. A possible limitation of the Network in doing this kind of low level pattern recognition only affects the local image reconstruction, and the resulting inefficiency can be recovered by the subsequent global decision making mechanism. For example, if a track-finding ANN fails on one track of an event, the information on other tracks can still be enough to make a decision based on the event topology (presence of vertices).

But when ANN's are used to make decisions resulting in a global event classification, any inefficiency reflects more directly on the quantitative Physics results. This is particularly dangerous at the trigger level, where any event loss is unrecoverable.

Another source of inefficiency in the use of ANN's for Physics decisions arises from the non precise knowledge of the physical processes themselves. While it is difficult to define to which extent a Network is limited by Montecarlo approximations, it is unquestionable that a supervised Network can only classify events which it has been trained to examine. Thus, in particular, unknown phenomena are out of the reach of a supervised ANN. This poses a strong challenge when using ANN's at the analysis level.

References

[16] "80170NX, Neural Network technology and applications", INTEL Corp., pub. no. 241359
S.R.Amendolia et al., "Research and Development Program on a fast Trigger System for Heavy Flavour Tagging in Fixed Target and Collider Experiments" CERN/SPSC/89-5, SPSC/P244, Proposal to the SPSC, 13 January 1989
C.Peterson, Nucl. Instr. and Meth, A279 (1989) 537

See Ref. [9] for an extensive bibliography on applications of Neural Networks to High Energy Physics
Databases and Parallel Processing

N Winterbottom
University of Southampton Parallel Applications Centre, Southampton

INTRODUCTION

Databases are at the core of most commercial data processing today. Over the last few years there has been a revolution in computer technology that has remarkably reduced the cost of basic computing power and data storage devices. The outstanding issue is "How is the technology to be harnessed to this key application area?"

Unfortunately, basic physical constraints decree that the compute power and the storage capacity come in small discrete units, hence the growth of "parallel processing" technology, which aims to put together many individual units to provide massive computing power for today's applications.

Complexity abounds in computing, not least in parallel processing. The Southampton Parallel Processing Centre has been set up to address some of the issues by entering into collaborations with industrial and commercial organisations that see parallel processing as a possible answer to their processing needs. The Centre provides the expertise needed to move parallel processing out of academia and into the commercial world. The Centre works on scientific, real time and database applications. This paper draws on the work done at the Centre in the database field and on the prior experience of Centre staff.

THE DEFINITION OF "PARALLEL DATABASE"

The terms "parallel" and "database" are often used in a wider sense than is intended in this paper. They will therefore be defined for the purpose of setting the scope of the paper.

In this context "database" means a relational database. This reflects mainstream thinking on the best way to represent structured data in commercial systems. The most common model is the 3rd normal form N-ary relational model, although the simpler binary relational model has been suggested as a possible successor. Practical implementations often deviate in detail from the pure model for performance reasons.

The alternative "network" and "hierarchical" models are not discussed, even though they still have significant presence in the commercial world.

The term database is often used referring to high volume special format data. Examples are unformatted text, digitised images and video, and data from physical sensors. These kinds of data usually imply application dependent coding for efficient storage together with high performance file systems. This paper does not discuss these applications although there is overlap in the file storage technology.

Parallelism occurs at many levels in a typical computer system. It is the intention in this paper to restrict discussion to the use of the relatively new "parallel processor" that incorporates multiple CPU units and multiple disk drives in a single machine. Loosely coupled mainframes, workstation clusters, distributed (possibly heterogeneous) DB systems and RAID (Redundant Array of Inexpensive Disks) technologies are not truly massively parallel and so are not discussed here.
CURRENT STATE OF THE ART

Hardware

Advances in semiconductor chip technology have made parallel processing viable. Connecting multiple CPU chips and multiple disk drives by high speed interconnection logic, gives a huge improvement in raw cost performance over traditional mainframe technology.

Machines from a number of manufacturers have recently appeared on the market. The problem facing a commercial organisation considering using parallel processors for database work is which one to select when it is not yet clear which machines will survive long term. The issue is particularly crucial because high performance database applications are often critical for the organisations' core business.

Software

Standards for general purpose operating systems on parallel processing hardware are not yet established. However, database server software for parallel processors is being vigorously developed by a number of suppliers who have identified the opportunities that are opening up. The existence of the widely accepted relational model, together with the standard Structured Query Language (SQL) facilitates this. SQL defines an industry standard interface behind which the complexities of parallel processing can be hidden.

Some software suppliers produce DB systems which are proprietary to particular machines (e.g. Teradata) but there is sufficient similarity in the structure of the machines available that DB server code can be structured such that a large proportion of the code can be portable across machines from different hardware suppliers (e.g. ORACLE).

Database Systems

A database system consists of hardware, software and people. The activities of the end users are usually well removed from the complexities of the computer technology. It is the database administrators in the computer departments who must master the technology of new systems. Parallel processing introduces an additional level of complexity to their task. The training required to pick up the new technology is a significant start-up cost for organisations wishing to carry out exploratory feasibility trials.

Taken together, the start-up training cost, the multiplicity of new competing hardware designs, and the criticality of the applications to business operations go a long way towards explaining the slow take-up of parallel processing in the commercial world.

TWO EXAMPLE SYSTEMS

Teradata DBC

The Teradata DBC is an example of a machine running proprietary database server programs.

The first layer of hardware consists of a number of input processors that accept network messages consisting essentially of SQL statements specifying database queries or updates. The processors analyse the statements, determine an execution strategy, and decompose the requests into operations destined for particular disk drives.

Operations inside the machine are transmitted over proprietary communications links, the "Y-net", to disk controller processors, each of which is connected to one or more disk drives. Each disk controller initiates appropriate disk read or write operations and performs initial processing on locally retrieved data. Conflicts for access to data blocks are detected and appropriate serialisation of transactions is ensured.
Data is returned, over the Y-net to the input processors for merging or is redistributed for further processing before the complete answer is returned to the external network.

The numbers of input processors, disk controllers and disks can be chosen independently to match the processing load for a particular installation.

The data for each relational table is spread over all the available disk drives. The distribution is determined by primary key values in some particular field chosen by the database administrator. The field values must be such as to ensure a reasonably uniform distribution of table rows across the drives. The administrator may also specify that unique or non-unique secondary indexes be maintained on other fields to meet application response time requirements. Data may be duplicated across disk drives to allow recovery after hardware failures. The administrator must choose a strategy for his application that provides an appropriate balance between availability, recovery time and cost.

**Meiko Relational DataCache**

The Meiko machine is an example of a system running ORACLE database software. The structure of the hardware is essentially similar to that of the Teradata, with a first level of processors, each running ORACLE server code processing SQL requests from local procedures or from an external network. An internal network built using transputers and proprietary switch chips provides communication between the input processors and disk controllers. A disk controller can have one or more disks attached.

The Meiko solution to access conflicts between transactions is to run a global lock manager on one or more transputer nodes. This process mediates ownership of data blocks between the first level processing nodes.

The DB administrator controls the distribution of data across the disk drives by defining "table spaces" on drives. He then allocates relational tables to table spaces. This has to be done with a view to ensuring a reasonable balance of activity across the drives taking into account the usage profile of the application.

**MONITOR, MODEL, PREDICT AND TUNE**

It is clear, from the brief system descriptions above, that design for performance is a large part of the database administrators' task. It is also true that performance prediction is a major concern today in organisations planning to run databases on parallel machines, particularly since there is as yet no great body of practical operational experience to draw on.

At the Southampton Parallel Applications Centre we have been addressing some of the issues in collaboration with commercial and industrial partners. The following remarks arise from this activity.

Parallel database systems are complex. Modelling low level system events with full statistical descriptions would be a major exercise. We have therefore adopted a policy of setting up analytical models that are as simple as we can devise consistent with still producing useful predictive accuracy. The models incorporate a number of basic mechanisms common in database systems. We make various judicious assumptions and approximations to limit the complexity of the model. These are discussed below.

A database system contains a number of distinct non-interchangeable resources types: for example, CPUs, disk drives, disk controllers, and network interconnections. The system throughput will be limited when any one of these becomes fully saturated with work. Cache storage space is also a distinct resource that is weakly interchangeable with other resources. A maximum utilisation of 100% can be assumed for a first upper bound approximation to system throughput, but a more refined calculations acknowledge that transaction response times
degrade as utilisation approaches 100%. The model therefore calculates degradation in
response time and allows the user to specify permissible degradation thus implying lower
utilisation values on resources.

In general, a database system will be running a statistical mixture of query and update
transactions. The characteristics of the most important transactions are defined to the model.
These include the tables accessed, the amount of data retrieved, and the access methods in
used. The relative frequencies of transaction types are specified. From this, the utilisation of
the machine resources is calculated. From a knowledge of the parallel and sequential execution
structure of the components of a transaction the best elapsed time is calculated for the
transaction running on an unloaded machine. A degraded response time is calculated for the
transaction in a machine loaded by a specified mix of transactions.

It may be the case that some resource is clearly not critical. For example, the network
interconnection bandwidth is more than adequate in systems we have modelled. In this case we
choose to ignore network bandwidth in our calculations with no significant error in the results.

A model must be calibrated against real world hardware and software. We do this by running
test cases that are designed to separate the effects of the various components of the system.
This determines the constants to be inserted into the performance calculations.

To support this activity we have developed driver and monitoring programs to apply
transaction loads to systems and to measure many variables in the running system. The monitor
can be installed on a production system to provide guidance in tuning production systems.

ECONOMICS AND COMMERCIAL ACCEPTABILITY

The current publicity about parallel computers, and, in particular, their use in running database
systems, emphasises the significant improvement in cost/performance over traditional
mainframe technology. Our tests on actual machines bear out this optimistic view.
Improvements of the order of 10 to 1 would seem to provide a powerful incentive to adopt the
new technology. The fact that commercial exploitation is apparently proving slow to take off
requires further analysis.

Cost/performance

The improvement in cost/performance stems from the use of commodity components in parallel
computers. CPU and memory chips and cheap disk drives, as found in mass production PCs,
are commonly used. The high bandwidth internal network components may be to proprietary
designs but, in the systems of which we have detailed experience, our observations indicate
that you do not need particularly esoteric network technology to service the CPU and disk
controller nodes in a typical parallel processor configuration running database operations.

Let us estimate the basic hardware costs to support one particular database application scenario.
Consider an amalgamation of all the administrative transactions initiated by a typical "man in
the street" in an industrialised country. These transactions are performed on his behalf by
organisations such as banks, travel agencies, retail shops, large employers and government
agencies. The costs will be calculated per end user. Assume that one person originates 30
transactions per day and that each transaction gives rise to 100 bytes of storage in a database.
This amounts to about 1 megabyte of storage per year. At current disk storage prices this costs
£1 per year (assuming all history is kept on line). CPU time is required for these update
transactions and also for additional query transactions. Assuming that 100 transactions per
person per day are required, and that the equivalent of 1/300 the power of a typical PC will
satisfy these, the CPU cost is of the order of £1 per year.

For this scenario the basic database hardware cost is £2 per year per end user. This can not be
the holding factor in the take-up of the technology!
Expandable systems

The hardware for parallel database systems can be expanded in small increments. Reconfiguring the hardware to include more CPUs or disk drives is a simple operation. Redistributing data across disk drives can be time consuming for large amounts of data but is not difficult. The incremental upgrade facility is a distinct advantage over traditional mainframe systems.

High entry point

Although the cost/performance ratio of parallel processors is attractive, even at small machine sizes, the start-up cost for an organisation moving into parallel database technology is significant. A major component is the cost of training database administrators in the additional skills and procedures involved in running parallel systems. These people are usually already fully occupied running existing database systems on which the organisation relies for its daily operations so it is not surprising to find IT managers thinking long and hard before diverting resources to the new technology.

The emerging parallel database systems are predicated on data being organised in relational form. If an organisation has its data in relational form it is not difficult to migrate it to one of the new systems. Similarly, if existing applications are written to an SQL interface, it is not likely to be difficult to migrate the applications, although extra work may have to be done to achieve optimum performance in the parallel environment.

Organisations that have not yet moved to a relational philosophy will find themselves at a disadvantage when moving to parallel databases since the main thrust of the manufacturers offerings is relational.

Reliability

Because many databases are vital to daily business operations, system reliability and availability is of prime concern. Parallel processing is a maturing technology and this is reflected in the systems currently available. For example, in some designs a single hardware failure may stop the whole machine. This is not a fundamental defect of the technology, but a characteristic of early versions of machines. Indeed, we may expect that, in time, parallel machines will be more robust against failures than typical mainframe installations.

Fluidity of technology

An IT manager looking for a parallel processor is faced with a marketplace in which a new machine design appears every few months. The number of suppliers and the fluidity of the situation makes it difficult to assess which machines are likely to survive over the lifetime of any proposed major application.

Special purpose machine

The machines on offer are presented as database servers complete with database system software. Their use for applications not connected with the database is operationally questionable. The machines must therefore be justified on the database operation alone. In many cases this will not be difficult. However, it is clear that the parallel machines will not displace existing general purpose facilities in the short term. To the IT manager the parallel machine will be an additional and different system to manage.

Expectations and actuality

The database world today is split into mainframe and PC worlds.
The mainframe

The concept of the corporate data led to the complex corporate database systems we see today. They are run on large mainframes and staffed by specialist IT staff. The driving force behind the original development of corporate databases was the notion of data sharing. Making this a reality required a common data architecture and sufficient technical development effort to get adequate performance at acceptable cost. The very complexity of the systems precludes fast response to demands for new applications to be introduced.

The first phase in setting up a new database application is to specify a relational data structure to satisfy the logical requirements of the application. From this is derived a physical design to run on the chosen database system. This latter step is a major part of the work and requires an understanding of the complex factors which determine the performance of the system.

First generation parallel database systems introduce some additional considerations into application design if best use is to be made of the parallelism.

The PC

PC based databases have been made possible by recent advances in hardware and software technology. It is now possible for individuals or departments to set up and run databases without formal IT support. These smaller systems can provide comprehensive database functionality but lack the performance associated with the larger installations. To the PC database user, databases are easy and quick to implement. Unfortunately, this ease of use temporarily hides the major problems of sharing data across a wide population. These issues were faced (and solved?) by the early mainframe database visionaries.

Performance is not usually an issue in a small PC database installation, but as an application grows in data volume or number of users the database administrator needs to become aware of the tuning techniques common in larger systems.

Some consequences

The apparently incompatible characteristics of high performance and ease of use are giving rise to heterogeneous mixtures of database systems within organisations. To allow data sharing across system boundaries, a new category of software is appearing in the marketplace, namely "glue logic". This comes either in the form of stand alone front ends which can access various database back ends, or as modular additions to database packages giving access to "foreign" database stores, either directly, or indirectly via data import/export routines.

This proliferation of complexity, driven by parochial commercial interests, is not a proper long term technical solution for the industry.

Let us summarise the negative characteristics of the position in which a large organisation finds itself.

1. The organisation must acquire skills in a variety of database systems. The cost of the PC database expertise may be hidden by fragmentation, but is real nevertheless.

2. The technical complexity of designing mainframe databases leads to long lead times in starting up new applications. The time scale is long enough to interact destructively with time scales associated with changes in business plans, company reorganisations, hardware acquisitions, announcements of new parallel processors, and training of database administrators.

3. The technical complexity of designing and running mainframe databases leads to a reluctance to put small applications on the mainframe. This is what drives users to local PC
solutions. It inhibits the efficient use of shared mainframe hardware and central IT expertise.

4. Fragmentation inhibits data sharing. Efforts to overcome this add complexity and cost to the operations.

The first generation of databases on parallel processors promises a quantum leap forward in hardware cost/performance and available absolute performance, but leaves the above problems largely untouched.

THE POTENTIAL OF PARALLEL DATABASES

Our experience in running parallel databases at the Southampton Parallel Applications Centre substantiates manufacturers' claims of a significant improvement in price/performance over traditional mainframe installations for the hardware component of the system. Figures of the order of 10 to 1 are quoted. This can only get even better as machines make more intensive use of commodity components, and the burdens of learning costs of a new technology are written off. We are now into a phase where industrial strength reliability and availability are being engineered into the hardware and software.

We have also shown to our own satisfaction that systems are practically can, in practice, be scaled to much higher performance levels than are possible with single CPU mainframes or sensible with clusters of such machines.

However, as argued above, other considerations dilute the attractions of moving to parallel processing for databases. Could parallel processing contribute something new towards solving the problems which stem from the technical complexity of large database systems? Let us analyse the situation further to identify some factors which could allow us to trade off cheap cost/performance benefits for simplicity of operation.

A large part of the complexity is associated with the physical design of a database. This involves the database administrator making numerous decisions about data distribution across disk drives and deciding when and where indexes should be specified. It requires, as input, detailed knowledge of the logical structure of the application data, the pattern of query and update references to the data, and the architecture of the machine. To do the job the administrator needs an understanding of computer science as applied to performance issues.

The traditional view is that it is not possible to apply the human intellect, in the form of a database administrator, until the application dependent information is available. Hence the need to train people in computer science at each application location. This is not ideal. The people best qualified to tune the system would be the machine designers but there are not enough of them to go round! Is there another way?

The answer is yes: dynamic tuning. The logical data structure is available now in data dictionaries. Usage patterns can be monitored by the computer itself as the system runs. This incurs minimal overhead. Operating parameters could be adjusted and data incrementally redistributed to optimise the system in real time.

The tuning facilities in current database implementations are not dynamic enough to support real-time tuning. Many adjustments can only be made when the system is stopped. Software schemes have been proposed which simplify the structure of the database engine and make it more amenable to dynamic tuning. If this could be achieved, a large part of the database administrators job would disappear, directly addressing the crippling problems of lack of responsiveness of central database services to changing business needs. Extra computer power and disk space would be required to implement the necessary late binding mechanisms. Parallel processing provides cheap CPU power and storage space. This can eventually make ease of use affordable in the database world.
(If you doubt that it is worth paying for computer power to enhance ease of use, note what has happened in the PC world. Most PC computer cycles are devoted to drawing easy to use "Windows" on screens, not on doing "real" work.)

It will require imagination and determination to escape from traditional assumptions about database system design, but critical observation of the structure of the industry today leads me to believe that a rethink is overdue.
The IEEE Mass Storage System Reference Model

J. Shiers
CERN, Geneva, Switzerland

Abstract

The IEEE Reference Model for Mass Storage Systems provides a basis for the development of standards for storage systems. The model identifies the high level abstractions that underlie modern storage systems. The model itself does not attempt to provide implementation specifications. Its main purpose is to permit the development of individual standards within a common framework.

High Energy Physics has consistently been on the leading edge of technology and Mass Storage is no exception. This paper describes the IEEE MSS Reference model in the HEP context and examines how it could be used to help solve the data management problems of HEP.

1 Introduction

These lectures cover the evolution of the IEEE reference model for mass storage systems and examine the suitability of the model for High Energy Physics.

The first lecture provides an overview of data handling in High Energy Physics, followed by a description of a software solution that was developed at CERN to meet the needs of the LEP experiments. Finally, the requirements of future experiments, such as those planned for the Large Hadron Collider (LHC) at CERN, and the Superconducting Super Collider (SSC) in the US, are discussed.

The second lecture reviews existing storage systems, with particular emphasis on those that influenced the IEEE MSS model and those that have been developed along the guidelines of the model. Included in this lecture is a discussion of network file systems. Although such systems were typically developed independently of the model, they are sufficiently important to warrant inclusion.

The final lecture is devoted to the model itself and how it can be used to build a global solution to the problems of data management in HEP. It covers the history of the model, the current status and the possible impact of the model upon HEP. Areas that the model does not cover are highlighted and possible solutions to these problems described.

Non-goals are a discussion of storage hardware, the data model or languages. Although these will be touched on in passing, they are not the main theme of the lectures.

Given the title of the lectures, it is perhaps useful to explain why topics other than the IEEE MSS model itself are discussed. There are a number of important reasons for this, as described below. Firstly, the model itself is still evolving and it is useful to examine its origins and explore future directions in which it may develop. Secondly, many of the systems that will be discussed had strong influence on the development of the model. Indeed, it was the belief of the developers of the model that a credible standard could not be achieved without demonstrable prototypes. In addition, it is important to understand how the model could be used to solve the particular problems in High Energy Physics.

Virtually all of the information that is contained in these lectures has been presented at an IEEE Mass Storage Symposium.

2 Data handling in High Energy Physics

2.1 Introduction

My own interest in data management started soon after starting my physics PhD at the University of Liverpool. The High Energy Physics group had a small IBM 360
computer that ran the MFT operating system. I was immediately faced with statements like the following:

```c
//G.FT11F001 DD DSN=GEOHIB.P4C79.FILM47,DISP=NEW,
//  VOL=SER=939727,DCB=(RECFM=VBS,BLKSIZE=32756),
//  UNIT=6250
```

This particular statement contains a simple error, namely that the /LABEL qualifier is missing. As a result, the first file on the volume will be overwritten. Such mistakes do not increase one’s popularity with fellow students, particularly when they have spent many hours of computer time to write the files destroyed by this omission.

Like many other experiments at the time, our tape catalogue consisted of a sheet of computer paper on which we were supposed to mark those tapes that we had used. Surprisingly, some groups still use electronic sheets of paper—a unstructured file that they modify with a text editor.

The above JCL statement is somewhat atypical, in that a semi-meaningful dataset name is used. This is still uncommon practice, even today. Many experiments prefer to use the same dataset name for all files.

Another lesson in the need for data management came shortly afterwards, when using the so-called PUBLIC EXEC DISKTAPE. As the name suggests, this exec file could be used to copy files from disk to tape. Unfortunately, it did not make use of one of the more esoteric features of IBM’s job control language, namely the UNIT=AFF qualifier. This allowed one to specify that one would like to use the same physical drive as was used for a specified stream in a previous job step. Failure to specify this absurd level of detail meant that the system rewound and unloaded the tape between each step and then requested a new mount, almost invariably on a different unit. This, naturally, did not increase one’s popularity with the tape operators.

After these and many similar experiences, it was clear to me that some form of intelligent software was needed. Whilst working on the NA9 experiment at the Max Planck Institute in Munich, I developed a package that provided access to data by name. The system automatically generated the necessary JCL statements, even managing obscurities such as UNIT=AFF etc. The system knew the correct record attributes of the different data types, and generated appropriate DCB statements on output accordingly.

Surprisingly, to me at least, this system was not popular. People preferred to hard-code specific tape numbers into their jobs. I used to liken these people to those who still kept (and probably still keep) dusty punched card decks in their desk drawers. However, I had the last word as I would occasionally repack volumes (automatically, of course) and so their jobs with hard-coded tape numbers would fail.

### 2.2 Data processing

Data processing in a High Energy Physics experiment involves a number of steps:

- Raw data is acquired (or simulated)
- The raw data is processed through the reconstruction program, producing Data Summary Tapes (DSTs).
- The DSTs are then reduced for further analysis, typically by reduction into Ntuples for processing by PAW.

It is interesting to note the widespread use of farms in the data processing of a modern HEP experiment. The simulated data is typically generated on a dedicated farm, such as the Central Simulation Facility (CSF) at CERN. The reconstruction is generally done at an experiment specific farm close to the detector itself. The data reduction is increasingly performed on dedicated farms such as Shift. Finally, we see the use of farms such as PIAF for parallel visualisation. This strong move towards distributed computing
has a major impact on the whole question of storage and data access. This shift is often paralleled to the move from the Ptolemaic view of the Universe, where the earth (or CERN VM) was considered the center of all things to the Copernican view. [1] As we shall see later, one of the main problems with the storage systems developed in the 1980’s was that they were based on a centralised model and do not adapt well to the distributed environment.

Until recently, only one level of offline storage has been used. This has changed slightly with the introduction of tape robots, where robotically mounted volumes are placed higher in the hierarchy than manually mounted volumes. It is clear that we will use multiple types of media in the future and it is important that the data access characteristics of the different types of data are taken into account. Raw data, for example, is processed infrequently - ideally only once - and sequentially. DSTs are processed much more frequently - at least once a day in the OPAL Shift configuration. Here random access is essential. Finally, Ntuples are processed many times per hour and should, for efficiency, be mapped to memory. 1)

2.3 Data volumes and rates

The CERN tape management system currently controls over half a million volumes. Assuming that all of these volumes are 3480s containing 200MB of data, this corresponds to some 100TB. Approximately 2% of the volumes are moved in or out of the central tape vault per week with a tape being mounted every 45 seconds on average. Over 300 thousand commands are executed by the tape management system per week.

Raw data sizes are currently in the range 100-500 KB/event. After reconstruction, the size of an event tends to grow by between 10-20%. This is because the raw data is normally carried through to the output stream to facilitate reprocessing.

2.4 Event directories

A relatively new technique that is used by two of the LEP experiments is the concept of event directories [2]. This technique, developed independently at DESY and on the OPAL experiment at CERN, uses a list of events with their characteristics, file, record number and offset. Rather than make multiple streams from a common set of DSTs, one stream for each analysis group, event directories permit a single set of files to be shared by all groups and users. It is clear that such techniques will be increasingly important in the future, if one is to reduce the overall disk space requirements.

2.5 Data import and export

Data import is primarily concerned with simulated events which are generated outside of CERN. These events are often generated using unused cycles on workstations and frequently arrive on Exabyte or other low cost media.

Data export is primarily at the DST level, although there is a growing trend which favours the export of Ntuples. Data export in particular is a non-trivial operation, requiring the allocation of media, shipping requests and the updating of file and media databases at sending and receiving sites. To what extent networks will alleviate this problem in the LHC era remains to be seen.

2.6 FATMEN

Shortly before the startup of LEP, a working group was established to investigate the requirements of the various experiments for a file and tape management system. This working group was given the somewhat unfortunate name of FATMEN, for File and Tape Management: Experimental Needs. The mandate emphasized the following points:

---

1) This is already done for the new column wise Ntuples. Active columns are unpacked and stored in memory. Inactive columns remain packed on disk.
The working group should also look at the software which the experiments use to locate events, the interfaces between the packages used to generate production jobs and the packages used to manage dismountable media.

Encourage the use of commercially available software or common developments between the experiments. If a common approach proves impossible then agreed specifications for inter-package interfaces must be generated.

Look at the problem of interfacing to tape management software that may be installed at different LEP production centers.

Some of the commercial packages that were investigated will be covered later. Unfortunately, none of these packages were suitable at the time of LEP startup.

2.6.1 The FATMEN review

The FATMEN committee investigated as many commercial packages as possible. Unfortunately, these proved without exception to be unacceptable. Most targeted only one operating system and those that offered multiple platform support did so in the most rudimentary of manners. One vendor even suggested that tape access should be offered on Monday, Wednesday and Friday on the VAX and on all other days on the IBM.

At the time of the review, the Rutherford Appleton laboratory in the UK were in the course of writing a new tape management system. This system was also to be installed at IN2P3 in Lyon, and seemed to offer most of the features that were needed for our purposes. The one feature that was not initially available was the ability to allocate tape volumes at run-time, now available via the GETPOOL command.

At that time, CERN was in the process of installing a tape robot. If one is to use a robot safely, some sort of tape management is mandatory. Tape volumes can be write locked by hardware. In the case of round tapes, this is via a write ring. For cartridges or cassettes, it is via a thumb wheel or tab similar to that on audio or video cassettes. Although most robots are up to the task of inserting a cartridge into a drive, they are uniformly incapable of write locking or enabling a volume. Hence, some sort of software control is required.

2.6.2 Recommendations of the FATMEN committee

The FATMEN committee made the following recommendations: [3]

- A file database system should be designed and written at CERN. Users would interact with a Zebra RZ database, whereas ORACLE would be used to maintain the master database.
- A Tape Management System should be imported from the Rutherford Appleton Laboratory in the UK.
- Servers should be used to supply data from the central systems to workstations.
- The evolution of commercially available distributed mass storage systems should be carefully followed. In particular, the IEEE Computer Society Reference Model for Mass Storage Systems and systems based on it should be studied.
- ..

The file database system is what is now known as the FATMEN package.

2.6.3 The FATMEN model

The FATMEN package was designed around a layer model. Although this is not the same as the layer model used to describe the IEEE reference model, it resembles it closely.

The model is composed of the following layers:

1. Event Tag Database
2. Production Database
3. File Database Layer
4. Network Storage Management
5. Tape Management System
6. Stage/Setup Layer
7. OS Tape Interface
8. Robot/Operator Layer

The first two layers were considered to be experiment specific, although a general purpose database which may be used to record details of production, calibration constants, detector geometry and so on is now available through the CERN Program Library under the name HEPDB. At the time of the review, none of the experiments expressed interest in what was then termed the Event Tag Database. It is interesting to note that two of the LEP experiments now use precisely such a system - the Direct Access to Data (DAD) system of OPAL and the event directories of Aleph. [2]

This, and the production database, were considered to be experiment specific. HEPDB [4], based heavily on the OPAL and L3 packages, is now available through the CERN Program Library to address these needs.

One of the purposes of this model was to enable different solutions at any level. This was to be achieved by defining standard interfaces, although it was clear that it would usually involve a certain amount of glue.

Although we have not yet described the IEEE model, it is perhaps useful to introduce some of the terms and compare the IEEE definitions with those used in the FATMEN context.

The IEEE model includes two layers which map very closely to the above model. These are the Physical Volume Repository, or PVR, which is almost exactly the equivalent of the Robot/Operator layer. The next layer is the Physical Volume Library (PVL), which encompasses the Tape Management System and the Stage/Setup layer in our model. The Stage/Setup layer consists of software that is layered on top of the host operating system which, amongst other things, provides an interface between the native tape mounting commands the Tape Management System. In the IEEE model, cooperating operating systems are supposed to conform to the PVL interface. In the opinion of the author, it is more reasonable to add a small layer, as is done with the NFS server for example, rather than expect the operating system itself to interface directly to the PVL.

The Network Storage Management layer was the subject of much discussion. In fact, we realised that the network would be omni-present, but were somewhat at a loss as to how best represent this. There is a growing trend for all of the above services to be available transparently anywhere in the network. This is not yet completely true for tape staging, but a common staging system is currently being developed at CERN which should be in production later this year.

An important feature of the above model was that the actual implementation of any one layer was fairly flexible. Thus, when moving from one platform or site to another, one staging system could be unplugged and another inserted. As the IEEE model evolves and systems that conform to it begin to appear, one could imagine replacing some of the layers with commercial software.

2.6.4 Features of the FATMEN package

The FATMEN system consists of a number of components. These include:

1. A file catalogue
2. A set of distributed servers that maintain copies of the catalogue at remote sites
3. Fortran callable and command line interfaces
4. Tools for data management and export

The file catalogue provides allows experiments to refer to their data via a device, operating system and location independent manner. The recommendation is that the naming scheme should describe those attributes of the data that are most useful to the physicist. This includes such information as

- Is the data 'real' or simulated? (Or test beam, cosmics etc.)
What stage of the production chain does it represent? (Rawdata, DST, ntuple etc.)

Physical characteristics, such as the beam energy, type, target, magnetic field, polarisation etc.

An example of a naming scheme is shown below:

```
FM> ls -w


RUN0391 RUN0392 RUN0393 RUN0585 RUN0631 RUN0391 RUN0392 RUN0393 RUN0585
RUN0631
Total of 10 files in 1 directory
```

It is important to stress that the generic name provides many forms of transparency. The same name may be used to access data on CERNVM, Shift, VXCRNA, UVVM etc. and regardless of whether the data is on disk, 3480, 8200, DAT, optical disk, in a Shift disk pool, accessed through NFS, AFS, DFS etc. to name but a few possibilities. Today, we begin to see location transparency through AFS or DFS, but we still do not see the other forms of transparency, which are an important feature of FATMEN.

### 2.6.5 The FATMEN naming scheme

There are a few limitations on FATMEN generic names, which mainly reflect the fact that the catalogue is based on the Zebra RZ package. These are as follows:

- The length of path elements may not exceed 16 characters
- The length of the filename may not exceed 20 characters
- The total length of the generic name may not exceed 255 characters
- The names are case insensitive

### 2.6.6 Generic names and physical files

The last example shows two entries for each generic name. A single generic name may point to an arbitrary number of physical files, or to another generic name. Multiple entries under the same generic name are permitted for a number of reasons. Firstly, one may have made copies that are to be exported to remote sites. Secondly, the files may reside on different types of media. Finally, copies may exist in different data representations or file formats. For example, one generic name might point to two copies: the first on a 3480 cartridge in a tape robot, on which the data is recorded in IBM native format 2), whereas the other may be on an optical disk in so-called exchange format. 3)

Under this scheme it is important to note that it is the user's responsibility to ensure that a new name is used if the data is modified.

### 2.6.7 Command interface

The FATMEN command interface is based on Unix. Thus there are commands such as `cp`, `mv`, `ls`, `pwd`, `rm`, `mkdir`, `rmdir` and so on. In addition, there are commands to manipulate catalogue entries, such as `add` and `modify`. Finally, there are utilities, such as `copy`, which provide a high level interface to file copying, described further in the section on data export, and commands to access the data itself.

### 2.6.8 Callable interface

The callable interface provides all of the functionality of the command line interface together with many additional features. For example, one may sort a list of generic names according to tape number and file sequence number within the individual tapes, and so on. The callable interface is recommended for efficiency, whereas the command line interface is probably more useful for casual use.

---

2) EBCDIC representation for formatted data, IBM floating point and VBS file format

3) Fixed length records, no control words; big-endian, IEEE floating point format, ASCII code for formatted data.
2.6.9 Interacting with the FATMEN catalogue

As mentioned above, the last example shows two entries for each file. In this particular case, one copy is on conventional round tape, e.g. 3420, and the other on 3480 cartridge. When attempting to access one of these files, a decision has to be made as to which copy to choose. This decision is based on a set of rules, which can of course be tailored or overridden.

The decision is not always trivial. In the simplest case it might be a choice of a copy on disk or a copy on tape, or a copy on a cartridge in a robot versus one that has to be mounted manually. However, the situation immediately becomes more complex if the disk based copy is on a different node, and is accessed via a server. Is it more efficient to access a tape copy rather than attempt a network access? Alternatively, one might find a disk copy in so-called Zebra exchange format but a tape copy in native format (e.g. VAX floating point, little endian). Is it more efficient to use the disk copy or wait for the tape mount?

Another feature that can be explained using the above example is that of catalogue subsets. For efficiency, we could limit all searches to a specific media type or set of media types. If, for example, we have no round tape support then we might as well mask out all such entries. The same can be done for data representation and location code.

```
FM> set/media 2
FM> ls -w


RUN0391 RUN0392 RUN0393 RUN0585 RUN0631
Total of 5 files in 1 directory
FM>
```

The latter is particularly important when many copies of a file exist. To explain this point further, we need to understand the default selection procedure in more detail.

Unless a specific copy is requested, or the default selection overruled, FATMEN will attempt to select the best copy according to the following rules:

1. For each type of medium, FATMEN loops over all copies in turn.
2. For disk datasets, a check is made to see if the file is accessible directly (e.g. via a Fortran INQUIRE), or via a server.
3. A local copy is always taken in preference over a served copy
4. For tape datasets, a check is made to see if the volume is available (e.g. not archived) and whether the device type required is available (or served) on the node in question.

If 30 copies of a DST have been made and sent to outside institutes, one can avoid making redundant queries to the Tape Management System by assigning location codes and masking off a subset of the catalogue appropriately.

2.6.10 Creating new data with FATMEN

It is certainly more complicated to create new data that is catalogued in FATMEN than to access existing data. However, the following model, which is designed for production chain jobs, is relatively straightforward to implement.

It is assumed that the generic name of an output file differs only in one directory name from that of the input file. Thus the input file

```
//CERN/NA44/RAWD/1991/PROT/450GEV/PB/HOR-3/RUN0391
```

might lead to the output file

```
```

Many of the fields in the output entry are generated automatically, such as the node name on which the job runs, the date and time, the user name and so on. Other fields, such as the Zebra format, record length and so may be copied from input or taken from a model entry, cf the so-called model DCBs on the DESY IBM system.
Assuming that the file is to be written to tape, the following operations may be required:
1. Allocate a new tape or a new file on a multi-file volume
2. Issue an output staging request
3. Write the data
4. If the job is successfully then
   (a) Request that the data be copied from the staging pool to tape
   (b) Write lock the tape
   (c) Add a comment (tag) to the TMS for this volume
5. If the job is unsuccessful then the volume (or space) is freed and the catalogue entry dropped
To be more explicit, one might wish to allocate a tape volume from a certain pool, e.g. XX_FREE. After writing the output dataset this volume might be moved to another pool, e.g. XX_DSTS.

2.6.11 Data export using FATMEN

Data export can be performed through FATMEN in a number of ways. Firstly, one may use the routine FMCOPY, or the corresponding command COPY. This provides numerous copying options, including the use of STAGE CHANGE, full Zebra I/O (thus permitting data representation and file format conversion), network copying (both TCP/IP and DECnet) and finally transmission via satellite, as described below. Alternatively, one may use the FATMEN primitives directly, as has been done by OPAL for the tool EXPUCART.

2.6.12 Data export using CHEOPS as transport

CHEOPS [5] is a project that uses the Olympus satellite to transmit physics data to remote sites. Transfer requests are made through FATMEN, either by using the command COPY or through the FMCOPY library routine.

The transmission is asynchronous. During the day, requests for copies are received and the data pre-staged to disk. The data is then transmitted at night.

When the user makes a copy request, the input data are verified by FATMEN. This includes ensuring that all parameters required by CHEOPS are supplied, either explicitly or implicitly via an appropriate catalogue entry.

An entry is added immediately to the FATMEN catalogue with the comment field set to something like

Copy request queued to CHEOPS on 930725 at 1315

At the same time, a copy request is queued to the CHEOPS server. The CHEOPS server processes the request and sends back a reply which is processed by the FATMEN server. This reply might indicate success, in which case a request ID is returned, or an error condition. In both cases the comment field of the corresponding FATMEN entry is updated, so that the user can track the progress of his request.

Finally, once the transmission has completed successfully, a further report file is sent from the CHEOPS server to the FATMEN server. At this time, the original comment supplied by the user is restored in the FATMEN catalogue.

About 37 GB were transferred by CHEOPS in July 1993. Nearly all of these data was sent to Helsinki, on behalf of the DELPHI collaboration.

The CHEOPS protocols do not depend on a satellite and could be generalised to use any medium, such as a normal terrestrial network. Similarly, the code in FATMEN to handle asynchronous copying is independent of CHEOPS and could equally well be used with cooperating tape copy stations etc.

2.7 Requirements for LHC and SSC

The data handling requirements for experiments at the LHC or SSC are expected to be significantly larger than those of the current generation. For example, some 10-
100MB/second are expected at the 3rd level trigger at the LHC. Although these rates sound large, they can, in theory, already be handled today. The E791 experiment at Fermilab built a data acquisition system that wrote to 42 Exabyte 8200 drives in parallel [6]. This system achieved an average throughput of 228KB/second, which is certainly very respectable when compared to the theoretical capabilities of the Exabyte 8200 drives, which are 246KB/second. Using this system, the experiment managed to acquire 50TB of data on 24K 8mm tapes, during their run in 1991.

Using similar ideas, one could cope with the anticipated data rates from LHC or SSC by using technology such as the Ampex D2 drive, which has a theoretical throughput of 14MB/second, in parallel.

We expect several (tens of) PB ($10^{15}$ bytes) of data per year. It is already possible to build a multi-PB library, using technology such as assembled by E-Systems [7].

In summary, although the hardware requirements for the LHC would appear to be very demanding, we could, given sufficient money, build working solutions today. However, the software requirements are much less clear.

2.8 Data management in the LHC era

One of the most critical problems for the LHC era (in the area of data management) is the question of data access. Should access be at the file, event or byte level, where we expect some $10^8$, $10^9$ and $10^{15}$ objects respectively. Although some people argue that this is a database problem, it is clearly a non-trivial one. Current database technology is not capable of handling such large numbers of objects, particularly when good performance is required. The D0 experiment at FNAL already have of the order of $10^8$ files. There are no intrinsic reasons why today's methods will not work with an order of magnitude more files. However, an increasing number of people argue that access by filename is inappropriate and that access via high level attributes is what is really required. This remains a largely unsolved problem.

However, it is clear that we do not wish to continue to write and maintain our own solutions for the next generation of experiments, but would prefer to obtain standard software, whether de-facto or de-jure. Open questions include distributed data, processing power and people versus distributed people, centralised processing and data [8]. Given the enormous volumes and the rapid decrease in cost of processing power, the latter solution would appear to be the more logical. Carried to its ultimate extreme, it implies keeping the data as close as possible to the experiment itself, and installing large processor farms there. This would be largely a continuation of an existing trend.

3 Review of existing storage systems

The developers of the IEEE MSS Reference Model recognised a clear need for prototype systems that were deployed in real environments. All of the following mass storage systems have been commercialised and have greatly influenced the development of the model.

A common feature of these systems is that they are designed to support super-computers. In addition, the applications are very different to HEP. Although the data volumes are large, the data rates are relatively low.

3.1 The origins of the IEEE Mass Storage Reference Model

The Mass Storage Reference Model finds its origins at a workshop held at NCAR in 1983. Although there had been several Mass Storage Symposia prior to this workshop, the idea of a generic model was first formulated at this meeting. The objectives of the workshop were to assemble a group of mass storage specialists whose task was to document ideas and experience from existing systems and to present these at the 6th IEEE Mass Storage Symposium. It is interesting to note some of the topics of discussion, which included centralised versus distributed systems, the integration of the system with the
host operating system and whether the system or user should be responsible for the placement of files in the storage hierarchy.

3.2 The Los Alamos Common File System

The Los Alamos Common File System (CFS) [9] is probably the best known mass storage system. It is marketed under the name DataTree, by General Atomics. CFS was developed in 1979 and is still used in production at many sites today.

CFS uses an IBM (or compatible) mainframe, running MVS, as storage server. Client systems are available for a large number of platforms, including COS, Unicos, CTSS, NOS, VMS, Unix, VM and MVS systems. CFS was originally developed in 1979 and has since been installed at some 20 sites, including the European Centre for Medium Range Weather Forecasting in Reading, UK. It is mainly written in PL/1.

CFS consists of a mass storage processor connected to a high speed network. The processor maintains a directory and journal file online and migrates files between a disk cache and offline storage. Migration depends on file size and last access. Files that are above a certain threshold in size are never stored in the disk cache but always offline.

To avoid disk fragmentation, disks are separated into classes with varying allocation units. This small files will be placed on disks with small allocation units and so on.

This is similar to the cluster size attribute for disks on VAX/VMS systems. The cluster size is the number of 512 byte disk blocks that are allocated at a time. The default is 3, which is clearly unsuitable for large files, such as those on the staging disks. On the other hand, the cluster size of 250 blocks which is used on the staging disks would be completely inappropriate for home directory style files, as is shown by the following example.

AXCRNB? create disk$stage:[cnsupport]jamie.test
This is a test to show how wasteful a large cluster size can be for small files.
Exit
AXCRNB? dir/siz=all disk$stage:[cnsupport]jamie.test

Directory DISK$STAGE: [CNSUPPORT]

JAMIE.TEST;1 1/250

Total of 1 file, 1/250 blocks.
AXCRNB?

Files stored in CFS are accessed via a Unix-style pathname. Files may be manipulated via special CFS client software, which provides commands such as GET, MOVE, REMOVE etc. In addition, an ftp interface has been developed, which is particularly interesting in today's environment, as it removes the problem of client software installation and support on multiple platforms.

One of the key features of CFS is that of device transparency. It provides access to data in a device independent manner, but also permits old technology to be replaced transparently. This means that the system can automatically migrate data off obsolete media onto newer media in the background.

It is almost guaranteed that the predominant medium at the startup of LHC will be obsolete during the lifetime of the machine. Given the volumes of data involved, the possibility of retiring old media is almost a requirement.

3.2.1 A user's view of CFS

A user typically sees both a local file system and the CFS. Files may be copied between the local file system and CFS. Once in CFS, the files may then be retrieved by any other node running a CFS client. Users may also specify attributes such as access frequency, which influences the decision on eligibility for migration.
3.3 The NCAR Mass Storage System

Like CFS, the NCAR MSS [10] is also based upon a central dataserver running MVS. It was developed at the National Center for Atmospheric Research in Boulder, Colorado after experience with CFS. Again like CFS, it is mainly written in PL/1.

The NCAR systems includes a feature which is frequently emphasised in the IEEE model, namely the separation of control and data. Control messages are passed over a standard network, whereas the data transfer is made over a so-called fast path, optimized for bulk transfer. In today’s environment one might use standard Ethernet for control, but FDDI or HIPPI for bulk data transfer.

The NCAR system provides automatic repacking of tape volumes. As files are deleted, either explicitly or via an expiry mechanism, holes occur on tape volumes. Eventually the data is rewritten to a new volume and the original volume freed.

The system also protects against media aging, by reading tape volumes at random. Should errors occur, the data are copied to a new volume without user intervention.

3.4 The NASA-Ames Mass Storage System

The NASA-Ames MSS-II [11] system is centered on Amdahl storage server running the UTS operating system. Its goal is to appear to the end user as just another Unix system with an infinite supply of very fast disk space. This system, the first we have described that was written for Unix, incorporates the following components:
- A striping file system (equivalent to Berkeley RAID-5)
- A hierarchical storage manager for Unix
- A volume manager, e.g. a Tape Management System
- A striped network

It is also important in that it is hidden under Unix. That is, the user does not have to learn a new set of commands, but can use standard Unix commands. In addition, the system is completely transparent to high level languages, such as Fortran or C.

The migration component will migrate files under two conditions. The first is periodic migration, which will typically be invoked using the cron utility. The second occurs when file system full condition occurs. Migration only affects data blocks - the inodes and directory structure remain intact. Demand restore occurs when the users issues an open for a file whose data blocks are not on disk. I/O will only block if a block that is not already on disk is referenced. As data is restored, it is delivered directly to the application from the system buffers in parallel to the restoration to disk.

Tapes are written in ANSI labelled format and contain only the files of a single user. This provides a convenient means of dealing with files which have not been accessed for an extremely long period - the associated directory structure is deleted and the appropriate tapes are mailed to the user in question.

3.5 The Lawrence Livermore Storage System

The Livermore Storage System is more commonly known by the name Unitree. Unitree is in fact a port of the Livermore system from their NLTSS operating system to Unix. Unitree is also marketed by General Atomic (since taken over by Open Vision). NLTSS is the Network Livermore Time Sharing System, a follow on the LTSS. In NLTSS, there is a single mass storage server, running under UTS (Amdahl’s version of Unix). The system is coded in C.

Given the relatively wide-spread availability of Unitree, the remaining discussion is based upon the commercial product.

Unitree is now available through numerous suppliers, including CDC, CONVEX, DEC and IBM. Unfortunately, the current versions that are available in the market place do not fulfill many of the early promises of the company. Most importantly, there are significant performance problems in a number of areas, particularly NFS access, and there are a number of design limitations which prevent its use at a large site such as CERN.
Firstly, the number of tape volumes is limited to 10K, although some vendors have increased this to 100K. The CERN Tape Management System currently tracks well over half a million volumes. The meta and tape catalogues cannot be backed up online, requiring up to 3 hours per day when the system is unavailable. In addition, the performance of commands such as `ls` is unacceptable. One site quotes roughly one hour to perform an `ls -l` of one thousand files.

3.6 System Managed Storage

System Managed Storage is a response to a SHARE\textsuperscript{4} white paper of 1979. SMS was not announced until 1988, as the Data Facility Storage Management Subsystem. It was initially available under the MVS operating system only, but some components have since been made available under VM/CMS. Many of the problems cited in the white paper were clearly due to some of the less user-friendly features of IBM operating systems. Nevertheless, some extremely important features, such as device transparency, storage classes and transparent device conversion where high on the list of requested features.

3.7 The FATMEN review

All of the systems described above were reviewed by the FATMEN committee. Unfortunately, none of them appeared to satisfy our requirements, for the reasons cited below.

- CFS requires an MVS based server. The CERN MVS system was scheduled for termination at the end of June 1989. More importantly, CFS required a proprietary network such as Ultrarnet or Hyperchannel, although ftp access was possible via a gateway server. In addition, there was no CFS client for VM/CMS systems.
- The NCAR system again required proprietary network protocols. In addition, it had not been installed at a single site outside of NCAR, although it has since been acquired by one of the NASA sites.
- NASTore unfortunately runs only on Unix. We also needed to provided client support for VM/CMS and VAX/VMS systems.
- Unitree was not yet available. Again, it only targeted the Unix world.
- IBM's SMS looked interesting, but was IBM only.

In addition, all of the above products, with the obvious exception of SMS, only supported the STK silo. CERN had just started a joint project with IBM on their cartridge robot.

3.8 Recent developments

Many of the sites responsible for the development of the above products have continued to develop Mass Storage Systems. In particular, it is worth mentioning the recent developments at Los Alamos and the National Storage Laboratory hosted by Lawrence Livermore National Laboratory.

3.8.1 Los Alamos High Performance Data System (HPDS)

HPDS is a fourth generation storage system designed to meet the storage and access requirements of Grand Challenge problems. These applications typically run on supercomputers and massively parallel machines and generate gigabytes to terabytes of output. HPDS is designed to store up to petabytes of data, deliver data at tens of megabytes per second and handle files up to terabytes in size. Under HPDS, storage devices are directly attached to the network and data transfer passes directly from the storage device to the client, without passing through an intermediary server. Client access is based on an ftp-like interface called **Data Transfer Interface** (DTI). DTI differs from ftp in that it permits

\textsuperscript{4} The IBM users' group
partial file transfers and appends. Future enhancements include powerful scientific data management tools and meta data handling.

HPDS has a limited lifetime and will be superseded by HPSS, to be developed at the National Storage Laboratory.

3.8.2 The National Storage Laboratory

The National Storage Laboratory is located at Lawrence Livermore Laboratory in California. To some degree, it can be considered a proof of concept laboratory where new storage ideas are put into practice. There are two main areas of development, namely NSL Unitree and HPSS.

NSL Unitree is based on Unitree 1.7 with enhancements, which include third party data transfer, as in HPDS, and multiple dynamic hierarchies. The latter is a concept similar to storage classes introduced by SMS and is described further below.

3.9 Dynamic Hierarchies

The current storage paradigm handles a fixed hierarchy, which often consists of disk and tape, or perhaps disk, robotic tape and manual tape. However, this hierarchy does not exploit the features of different devices, such as solid state disk, RAID, 3480, 8mm, DAT, DD-2 etc. A more general approach is to build a multi-level hierarchy and define different migration paths for various types of data. For example, large, infrequently accessed data might migrate directly to DD-2, whereas more frequently referenced data might migrate to optical disk. In addition, this scheme should be flexible enough to permit the removal and addition of old or new media types. To remove a media type, e.g. 3420 tape, one would edit the appropriate configuration file and perhaps add a new type at the same time. The system would then ensure that the data were transparently migrated from the old media, which could then be removed from service.

3.10 Network File systems

3.10.1 NFS

The Network File System or NFS, originally developed by Sun, is almost certainly the best known distributed file system. NFS is based upon a stateless protocol, using remote procedure calls over UDP.

NFS servers must explicitly export those file systems that they wish to be network accessible. Clients must mount these file systems, or use the automounter, which automatically mounts file systems as needed and then dismounts them after a certain period of inactivity. The mount point may, and often does, differ from client to client, leading to considerable confusion.

NFS security is based upon Unix user and group IDs, which mean that it is only secure within a well-defined and controlled network.

NFS clients and servers exist for many systems, including Unix, VMS, VM, MVS and Novell (in some cases, such as VM/CMS, not all NFS functionality is supported. VM/CMS, for example, does not provide an NFS client).

3.10.2 The Andrew File System

The Andrew File System, or AFS, was developed at Carnegie-Mellon University, Pittsburgh. It is marketed by Transarc Corporation and has been selected by the Open Software Foundation (OSF) as the basis for its Distributed File System (DFS).

One of the important design considerations for AFS was that it be compatible with NFS. This is performed by the AFS-NFS exporter, which permits AFS file systems to be accessed by NFS clients through a gateway machine. For example, on VXCRNA the logical name AFSCERN points to a NFS mount of /afs/cern.ch/user.

From a user point of view, one of the most important features of AFS is the global name space. Thus, the file /afs/cern.ch/user/j/jamie/fatmen/fatmen.car
can be accessed transparently from the SSCL in Texas, CERN, or indeed any other node that can see the cern.ch cell.

Before describing what is meant by the term cell, it is worth mentioning an additional feature that is much appreciated in the wide area. This is the concept of local caching.

As can be seen by the following example, the first access from Texas to a file at CERN involves a small time delay. Subsequent accesses, however, are almost indistinguishable in terms of speed from those performed locally. Note that there is an improvement in access time even for local access, for the same reason.

```
zfatal:/afs/cern.ch/user/jamie/fatmen (543) time head -i fatmen.car +TITLE.
real 0m0.39s
user 0m0.01s
sys 0m0.03s

zfatal:/afs/cern.ch/user/jamie/fatmen (545) time head -i fatmen.car +TITLE.
real 0m0.06s
user 0m0.02s
sys 0m0.02s

synergy.ssc.gov$ time head -i /afs/cern.ch/user/j/jamie/fatmen/fatmen.car +TITLE.
real 0m7.36s
user 0m0.00s
sys 0m0.12s

synergy.ssc.gov$ time head -i /afs/cern.ch/user/j/jamie/fatmen/fatmen.car +TITLE.
real 0m0.05s
user 0m0.02s
sys 0m0.02s
```

**AFS cells** An AFS cell is a group of servers and clients in a single administrative domain. AFS cell names are typically those of the ftp domain, e.g. cern.ch.

**Operating system dependant pathnames** One obvious problem with a global naming scheme is the handling of operating system dependant code. The files that one would normally put in the bin subdirectory of one’s home directory are likely to be operating system specific, unless they are shell scripts.

This is handled in AFS by the use of @sys in the pathname. This is automatically translated by AFS into an operating system specific path. On my RS6000, this translates to rs.aix32. On an HP machine running HP/UX version 9, this would translate as hp700.ux90. Thus, by linking the bin directory to @sys/bin, the correct binaries will be automatically selected.

### 3.11 AFS and physics data

The global naming scheme of AFS would appear to be attractive for accessing physics data. Although AFS cannot itself improve the network bandwidth, local caching will certainly result in lower network load and better response time for repeated accesses to the same files than NFS. As such, it is well suited to accessing small amounts of test data in read only mode. It is not well suited to handling files which are frequently updated, as this would result in equally frequent cache updates. It also does not offer any improvement
over NFS for bulk data access. Both of these problems can be solved by using a library routine which bypasses the cache.

3.12 Integrating AFS with hierarchical storage systems

One feature that is lacking from standard AFS is hierarchical storage management. Two sites have done interesting work in this area, namely the Pittsburgh Supercomputer Center and the University of Michigan. The approaches taken are quite different and are described briefly below.

3.12.1 The Institutional File System (IFS)

The Institutional File System was developed at the University of Michigan [12]. It is based on AFS, over which it offers a number of enhancements. These include new server platforms, intermediate servers, new classes of clients and extensions to the authentication mechanism.

The main new server platform is the IBM 390 architecture, running under MVS, VM/CMS or AIX/370. This permits IFS to exploit the traditional strengths of a mainframe, namely high I/O capability, large disk capacity and main memory, high reliability and general mainframe utilities. The importance of many of these items was clearly greater when IFS was designed in the later 1980s.

A more important extension is the addition of hierarchical storage management, provided on the mainframe by IBM’s DFHSM. In addition, IFS have made a number of enhancements in the area of caching. The most significant of these is the concept of intermediate caching servers. IFS supports tens of thousands of workstations and PCs and it was felt unrealistic to serve such a high number systems using a single or small number of traditional AFS servers.

Further enhancements in the caching policy are planned, based on the following observations.
- After a file block has been accessed, the chances that it will shortly be reaccessed are high. Caching the block exploits temporal locality
- The chances that the subsequent block will be accessed is also high. Prefetching exploits this probability
- When a file on an archived volume is retrieved, other files from the same volume are frequently retrieved shortly after. Restoring all files from a tape exploits this. This does not necessarily hold for HEP. This feature was implemented in the VAX staging package used by FATMEN but did not help as anticipated at D0, FNAL
- Some blocks are more important than others. One should weight blocks appropriately in the flushing algorithm. e.g. directory blocks should be retained over data blocks
- Blocks should also be weighted in relation to their acquisition cost. e.g. blocks that were expensive to obtain, e.g. read from tape, should have higher priority than those that were cheap to obtain
- Similarly, some blocks are more expensive to flush, as they must be written to a slower storage device
- Once a user lists the contents of a directory, he frequently attempts to access a file in that directory shortly afterwards. Such hints should also be taken into account
- Some blocks within a file are more popular, e.g. head and tail blocks

3.12.2 Pittsburgh Supercomputing Center

Prior to migrating to AFS, the Pittsburgh Supercomputing Center used CFS to provide mass storage services. AFS has been extensively enhanced to meet their requirements for a mass storage system as described below [13].

Firstly, AFS has been enhanced to provide hierarchical storage management. For this purpose, AFS was modified to support multiple residencies. This permits a copy of
a file to exist in up to 32 places. This has the side effect of enhanced reliability. If one component of storage system is down, a request for a given file will be satisfied from highest priority storage system that is available. Data migration is performed between storage systems according to attributes in a database. These attributes include characteristics such as desired file size distributions amongst the different levels of the hierarchy. Media types can be replaced transparently by issuing only two commands.

For practical reasons, multiple AFS servers are required. Currently, each server can only support 52 GB of disk space. To avoid having to connect a robotic device of each type to every AFS server, the servers have been modified to communicate between themselves via RPC to provide shared storage services. Supported servers currently include Maximum Strategy RAID arrays, HP optical jukeboxes, Crays running Data Migration Facility and RS6000s running Unitree.

3.13 OSF/DFS

The OSF Distributed File System is based on AFS version 4.0. There are a number of differences between AFS and DFS, which include the following. DFS is currently only available for IBM RS6000 machines (both client and server). The RS6000 server may use either the standard AIX journaling file system or Episode. Episode is the standard file system for DFS and provides the basic functions that are required, e.g. ACLs. In addition, it is also log-based, which is important if one is to avoid length restarts after server crashes. DFS path names start /... rather than /afs. DFS and AFS can coexist, but are two separate file systems with two separate caches.

Is the global naming scheme of AFS or DFS with its caching capabilities and hierarchical enhancements such as those made at the University of Michigan the answer to our problems? Unfortunately, although extremely attractive for home directory style files, such a solution is probably not optimal for physics data files, for reasons that are discussed below.

3.14 Home directory file services

Home directory file services include backup/restore, migrate/recall and archive/retrieve. Surprisingly, these functions are often confused.

3.14.1 Backup and restore

Backup and restore can be further subdivided into disk (or filesystem) and file backup/restore. Disk backup is performed to permit disaster recovery. To a large degree, it can be avoided by using techniques such as RAID. Furthermore, it can be simplified by using dataless workstations. Firstly, one should ensure that the workstations are running from standard system disks. Any local tailoring that is required should be done using a script from a standard starting point. This permits one to backup only the master copy of the system disk and not the copy on each (remote) workstation.

File backup is performed to permit users to recover accidentally deleted or corrupted files. As such file restore must clearly be user driven. An analysis of restore requests on the central VAXcluster VX/CERN at CERN shows that the provision of version numbers in the VMS file system virtually eliminates the need for file restore. Unfortunately, Unix does not provide version numbers. However, one may achieve a similar effect by using code managers which keep copies of multiple versions of a program. In addition, techniques such as the trash-can, where deleted files are stored for a few hours, permit the recovery of files deleted by mistake. Finally, one should ensure that easily recreatable files, such as a.out, core, .o etc. are excluded from the file backup.

3.14.2 Migration and recall

File migration provides the user with virtually unlimited disk space. Unused files move from faster, more expensive media to slower, lower cost media. Recall must clearly
be transparent to applications and user programs, but both migration and recall should be user-driveable. This permits users to explicitly migrate files that they know will not be accessed for a long time, e.g. if they are about to leave on sabbatical and to recall files that will soon be accessed, e.g. when they return. If file migration is not provided, people often misuse file backup and archive facilities. It is clear that many of the concepts of SMS are required, whereby files are migrated to an appropriate level in the storage hierarchy, as determined by their storage class.

3.14.3 Archiving and retrieval

Archiving has a number of features that differentiate it from migration. The most significant is that archived files belong to a different name space. Some people argue that this is an advantage, although this can of course be achieved using a simple script built around the migrate command. This script would demand migrate the file and then mv it to a different filesystem or directory or rename it to a dot file, i.e. one that is not visible to a simple ls command. Thus, provided a good migration system is available, the need for archiving is considerably reduced. Some sites, such as the Rutherford Appleton Laboratory in the UK, have maintained a policy of removing archival systems in favour of migration systems.

3.15 Open Storage Manager

The Open Storage Manager is a set of three products from Lachman technology. The family of products are built around the IEEE MSS reference model and comprises
1. Transmigrator, a file migrator
2. Conservator, a storage system
3. Meditator, a distributed removable medium and library management package

3.15.1 Transmigrator

Transmigrator is a software package that provides file migration services for existing Unix file systems. Files may be migrated in any of three ways:
1. Explicitly, i.e. at the request of a user.
2. On demand, i.e. of there is a lack of disk space.
3. Periodically, e.g. from a cron job.

When a file is migrated out of the normal file system, a so-called stub-file remains. This stub-file contains the first data blocks of the file plus a bitfile-ID which points to the migrated copy. Obviously, there are thresholds on the minimum file size that a file must have before being eligible for migration. In addition, there are thresholds for the minimum age of a file. By default, these thresholds are set to 10KB and 1 day since last access.

Watermarks Transmigrator uses three watermarks. In addition to the traditional high watermark, at which point migration is initiated, and low watermark, at which point migration terminates, there is a so-called prestige watermark. This watermark is lower than the low watermark. After reaching the low watermark, transmigrator may optional migrate additional files down to the prestige mark. These files remain on disk in their entirety. However, should disk space be urgently required, it can now be released very rapidly simply by creating the stub-files - the data has already been migrated into the backing store.

Symbolic attributes Two symbolic attributes are associated to a migrated file. These are the storage group and migration path. These attributes permit different files to be migrated into different stores, and control the subsequent migration within those stores. This can be useful when the attributes or access patterns of certain files are known. Many of the files that are migrated will be log files, which are rarely, if ever, recalled. One may then migrate these files directly to a low level in the hierarchy onto low cost devices.
3.15.2 Conservator

The Conservator is a storage repository that is used by Transmigrator when migrating and recalling files. It is not required by Transmigrator, which can migrate to and from any mounted file system.

Conservator supports multiple internal hierarchies. That is, certain files may be migrated to one group of storage devices and others to a different, potentially overlapping group of devices.

3.15.3 Mediator

The mediator handles dismountable volumes and robotic libraries. In terms of the IEEE MSS RM, it provides the functions of both the PVL and PVR.

The mediator provides a number of timers to optimise operation. These are the wait-timer, hog-timer and release-timer.

When a volume is released, it is maintained online for wait-timer seconds. If a new request for the same volume arrives within this interval, it will be satisfied immediately.

To protect against a given user repeatedly remounting the same volume, the wait-timer is turned off after hog-timer seconds from the time when the volume was first released.

3.16 ADSTAR

As one approaches the Adstar building in San Jose, California, the meaning of the company's name is made clear - Advanced Storage and Retrieval. These last two words feel strangely reassuring.

ADSTAR is a component of IBM. It has offices in Tuscon, Arizona, which make tape products, and in San Jose, where disk products and storage software are produced. ADSTAR are currently represented at the PVR working group meetings.

The ADSTAR Distributed Storage Manager provides backup and archive services for local, NFS and AFS file systems. Space management (migration) is currently being added. ADSM clients are already available for MVS, VM/CMS, many Unix platforms and Novell. Additional clients are planned for other Unix platforms and VMS systems. The servers are also being ported to Unix platforms. ADSM supports many concepts from SMS, such as dynamic hierarchies and transparent media retirement.

In addition to command line and Motif interfaces, ADSM offers a library interface. This interface is potentially extremely interesting for the storage and management of HEP data.

3.17 A possible solution to physics data storage

One of the big problems of using AFS together with hierarchical storage management for physics data is that all data must pass through the file system. Not only is this inappropriate for certain files, such as rawdata files, but it also gives problems due to the sheer number of files involved. Currently, experiments have between \(10^5\) and \(10^6\) files. These numbers are already large for Unix file systems. Future experiments can expect up to \(10^9\) files, although the actual number is likely to be somewhat smaller. Today's file sizes are limited to roughly 180MB so that they fit conveniently on a 3480 cartridge. Tomorrow's media are likely to have capacities well in excess of 1GB, so that the actual growth in the number of files is likely to be somewhat reduced.

A feature that is frequently requested by Unitree users is the ability to bypass the disk cache for certain types of files - typically large data files such as those used in HEP. CFS read and wrote files larger than a certain threshold directly from/to tape - in 1979!

In addition to the support for multiple dynamic hierarchies and retirement of media, ADSM also permits meta-data to be associated with files. Thus, one could consider using the ADSM product to manage physics data in one of three ways:
1. By modifying the existing FATMEN system to use the ADSM API to store and retrieve data.
2. By replacing the existing FATMEN catalogue with the ADSM internal catalogue and a small amount of code based on the API.
3. By using a conventional Unix file system instead of the FATMEN catalogue, with metadata stored in dot files.
   In all cases, ADSM would take on the functions of our current Tape Management System, Staging systems and Robot control.

4. The IEEE MSS Reference Model

4.1 History

The IEEE MSS Reference Model grew out of the initial generic model first presented at the 6th IEEE MSS symposium. The model is now approaching version 5, and it is planned to freeze the model soon and begin work on standardising the various layers of the model. Unfortunately, there are still a number of areas of disagreement. However, it is likely that the model will mature bottom up, the first standard being for the PVR.

It is frequently pointed out that there are two sorts of standards: de facto, like Sun’s NFS, which is widely available and de jure, like OSI, which is essentially unused. We would clearly like a standard at least as interoperable and available as NFS.

4.2 The Storage System Standards Working Group

In 1989, the IEEE Computer Society Technical Committee on Mass Storage Systems and Technology created a working group, known as the Storage System Standards Working Group or SSSWG. The goal of the working group is to convert the model into a set of standards for submission to the IEEE and ANSI.

The immediate goal was the decomposition of the model into interoperable functional components that could be offered by vendors as commercial products. The long term goal was to define interoperable standards which define the software interfaces and protocols associated with each of the architectural elements of the model.

The members of the SSSWG come from vendors, such as DEC, HP, IBM, StorageTEK, Convex plus user sites, such as Lawrence Livermore National Laboratory, the University of Michigan and CERN. To acquire voting rights, at least 3 out of the previous 6 meetings must be attended. Up to now, all meetings have been held in the US but it is planned to start a series of workshops in Europe, the first of which will be held at CERN later this year.

4.3 Important architectural features

The following features figure prominently in the design of the model.

1. Location independence
2. User and system oriented file identifiers
3. Separation of control and data paths
4. Applicable to all sizes of storage systems

4.4 Overview of the components of the model

The model is built out of a number of functional components. These are as follows:

- **Movers** are responsible for copying data to and from storage media and requesting the transmission of data from source to sink.

- A **Physical Volume Repository** is responsible for storage removable media containers, such as tape cartridges or optical disk platters, and mounting these containers onto devices through robotic or human agents.

- A **Physical Volume Library** maps storage media, or physical volumes, to movers. The PVL provides a client interface that permits physical volumes to be mounted. The PVL maps these volumes to cartridges.
A **Storage Server** provides mechanisms to compose stores from discretely addressable storage spaces and translate access requests to these stores into a set of requests to the appropriate mover and/or PVL if the associated volume is not mounted.

### 4.5 A personal view of the model

I feel that it is important to use terms whose technical meaning is as close as possible to that of the English words used. In that respect, one would define a **Physical volume** as being the smallest entity that the **Physical Volume Repository** can address. On top of such objects, one might build **logical volumes**, which would be managed by the **logical volume library**. These definitions can be explained by a simple analogy with books. A **volume** may contain several books, alternatively a book may be spread across many volumes. In this analogy, the volume, which is the smallest thing that we can pick up in a book shop, is the **physical volume** of our model. The **logical volume** is the equivalent of a book.

Logical to physical mapping can be used to explain existing objects, such as those found on VAX/VMS systems. A **volume-set** is a bound set of disks that appear to higher levels as a single entity. In this respect, the **logical volume** is composed for two or more **physical volumes**. A **shadow-set** is also a bound set of one or more disks, however its appearance is completely different to higher layers! In this case, the data is replicated on the different physical volumes, giving reliability and performance improvements (at least on read). A further possibility is that of a **stripe set** or even of a striped, shadowed volume set.

In addition to the advantages expressed above, logical volumes also have benefits in terms of transparent migration from one physical volume to another and for caching purposes. At CERN, we have used single member shadow sets on VXCERNN for many years. When a hardware problem is suspected, a spare disk is added to the shadow set. Once the **catch up copy** has completed, the original disk can be removed and repaired. Thus we have performed transparent migration from one physical volume to another. The same technique could also be used to perform transparent media migration. The local volume, which today might map to a complete physical volume, may well be moved to much higher capacity media in a many that is completely transparent to the user. This feature will certainly be required for the LHC era. Other benefits include cached, striped logical volumes. High capacity media often fails to bring with it an appropriate increase in throughput. If one requires that a complete volume be processed in less than 100 seconds, to avoid problems with drive and volume contention, one finds that even striping is not sufficient to solve the problem. Logical volumes help by keeping the size of data that is to be moved at a manageable level. Finally, one could move logical volumes according to access patterns. Those that were frequently accessed together could be moved transparently so that they were on a single physical volume.

### 4.6 The official view

Unfortunately, although the IEEE model supports the concepts described above, the official names are **cartridge** for the smallest object that one can address and **physical volume** for our **logical volume**.

### 5 Future developments

A number of developments are foreseen to enhance FATMEN in the short and medium term. In the long term, it is hoped that the IEEE Mass Storage work will develop sufficiently so that standard solutions can be used.

#### 5.1 The short term future of FATMEN

The main short term changes concern usability. Originally, it was envisaged that the main use of FATMEN would be via the Fortran callable interface from production
programs. To facilitate access to data, a command `fmln` will be provided which will make access to a catalogued file, regardless of its location, as simple as creating a Unix link.

For example, before running a program that processes a data file on Fortran logical unit 1, one might make a link as follows:

```
ln -sf ~jamie/data/fxfile.dat fort.1
```

The same program could then access a file catalogued in FATMEN if the following command were first issued:

```
fmln //cern/l3/prod/data/ldre/cc02zkxu fort.1
```

Although data cataloguing is normally performed only by a small number of production managers, a similar command could be provided to simplify this task, as shown below:

```
fmcat //cern/l3/prod/data/ldre/cc02zkxu vid.[fseq] [options]
```

### 5.2 The medium term future of FATMEN

In the medium term, one could envisage Fortran 90 and/or C interfaces, which would use standard language features, such as derived data types of structs, rather than Zebra banks. More importantly, the area of data export could be improved, e.g. by the provision of an asynchronous data export service along the lines of the CHEOPS project but possibly using terrestrial networks.

In addition, a standard `event catalogue`, such as that used by Aleph, H1 or OPAL could be provided and integrated with FATMEN.

To facilitate the use of standard conforming products, as they emerge, the declared interfaces of the components of the FATMEN model could be made conformant to the IEEE Mass Storage Systems Reference Model. This would allow us to replace our existing Tape Management System with a commercial solution, for example.

### 5.3 The long term future of FATMEN

Unfortunately, it is unlikely that the IEEE model will ever solve all of our needs. For example, the `names server` component is currently outside the scope of the model. Other important areas that are not presently addressed include the handling of files with internal structure (the IEEE model treats a file as just a stream of bits) and the whole question of data management.

However, it is clear that the IEEE work will never address issues as HEP specific as Zebra FZ initialisation. Consequently, a (hopefully thin) layer of HEP specific code will always be required. Nevertheless, it is extremely important that the HEP community provides adequate input to the Storage System Standards Working Groups.

### 6 Summary

High Energy Physics has a data management problem that is at least the equal of that of any other field. Future experiments will present even more challenging problems and it is essential that we start trying to understand these problems now. As data management touches many fields, we should adopt a model on which to build. The logical choice would appear to be the IEEE reference model for Mass Storage Systems. This model already addresses many areas that we know are of critical importance, such as `dynamic hierarchies`, transparent transition from old to new technologies etc. Rather than work in a vacuum, we should work closely with the IEEE Storage System Standards Working
Group to ensure that any standards or products that appear will cope with the needs of HEP.

References


Working With Object Stores of Events Using ptool

Robert Grossman
University of Illinois at Chicago, Chicago, USA

Abstract

The purpose of these notes is to give an expository treatment of how to use a persistent object manager called ptool to analyze high energy physics data. Traditionally, the analysis of scientific data has been file based. On the other hand, scientific data often has a rich semantic complexity, which mirrors objects more closely than files. The goal of ptool was to replace file based analysis of scientific data with object based access in a scalable fashion. Ptool was designed to scale as the amount of data scales by interfacing object stores to hierarchical storage systems; and as the complexity of the query scales by providing the ability to query objects in parallel.

1 Introduction

The purpose of these notes is to give an expository treatment of how to use a persistent object manager called ptool to analyze high energy physics data. Traditionally, the analysis of scientific data has been file based. On the other hand, scientific data often has a rich semantic complexity, which mirrors objects more closely than files. The goal of ptool was to replace file based analysis of scientific data with object based access in a scalable fashion. Ptool was designed:

• to scale as the amount of data scales by interfacing object stores to hierarchical storage systems;
• to scale as the complexity of the query scales by providing the ability to query objects in parallel.

Another important design goal was to provide the ability to create, update and access persistent objects in a tool based framework [4] rather than a framework based upon object oriented databases. In particular, ptool was designed to provide high performance and low overhead. This was done by trading performance and overhead for functionality. Ptool only provides persistence for complex objects—any other functionality, such as concurrency or transactions must be layered on top of ptool.

The idea of using databases and object stores to analyze high energy physics data began with [18]. Ptool was initially developed for aeronautical applications [5], [8], [6]. A proposal for analyzing HEP data using object stores was described in [1], which led to the PASS Project. The PASS Project is a five year research and development project supported by the US Department of Energy under the High Performance Computing and Communications Program with the goal of developing new technology to analyze HEP data using database computing. The project is presently in its second year. As part of the PASS Project, ptool has been applied to the analysis of HEP data viewed as collections of persistent events [3] and [7].

The current notes are an expository treatment of the use of ptool to analyze HEP data. The research explained in these expository notes is the work of many individuals. This research is described in the reports [13] and [14]. These notes describe just one part

1) This research was supported in part by NASA grant NAG2-513, DOE grant DE-FG02-92ER25133, and NSF grants IRI 9224605 and CDA-9303433. For more information, contact Robert Grossman, Laboratory for Advanced Computing, m/c 249, University of Illinois at Chicago, 815 South Morgan Street, Chicago, IL 60607, grossman@math.uic.edu.
of the PASS Project. The PASS Project has also analyzed HEP data using both relational databases and object oriented databases, as well as analyzed the differences between these approaches [9].

These notes are organized as follows: Section 2 is a brief introduction to object stores; Section 3 describes the functional requirements for our problem; Section 4 describes the basic concepts of the system; Section 5 describes the physical data model; and Section 6 describes the architecture and implementation. There is also a rather lengthy appendix which contains program fragments with some basic examples. The purpose of the appendix is to give enough information so that someone familiar with UNIX can create, populate and access persistent stores of events.

Ptool was designed by the present author and Xiao Qin [10]. D. Valsamis and W. Xu also made important contributions to ptool. The eventstore [13] and [14] was designed and developed by the present author, D. Lifka, D. Malon, E. May, L. Price, X. Qin, and D. Valsamis.

The examples in these notes were run with ptool, version 0.6 release 2 on a Sun Sparcstation 1.

2 Object stores

We begin with a quick review of some basic definitions, which we discuss more fully below. An object is an abstract data type together with some functions, or methods, for creating, accessing, and modifying it. In the context of object stores, an element of the abstract data type is called an attribute of the object. An object is called persistent if it continues to exist once the process which creates it dies; otherwise, the object is called transient. An object manager is a process which can create and manage persistent collections of objects, called stores. An object oriented database is an object manager with additional functionality, such as transactions, back up, and recovery.

The underlying thesis of these notes is that for many applications file based access to scientific data should be replaced with object based access using an object manager. This provides several advantages:

- With object-based data access, scientific and engineering applications can easily exploit specialized application specific algorithms for storing and accessing data in order to gain higher performance. In other words, rather than accessing unstructured data using the file system, one accesses the relevant structured objects by querying a persistent object store.
- By using an object store instead of an object oriented database, one can usually obtain higher performance and greater scalability. This is because the additional functionality of an object oriented database involves trading performance for functionality. With a persistent object store, one can incrementally add functionality as required for a specific application.
- By using an object store, code for the application specific analysis of data can be modularly separated from more generic code to store and access data.

Although partly misleading, it is helpful to consider the analogies in Table 1 between file systems and object stores and the analogies in Table 2 between relational databases and object oriented databases.

3 Functional Requirements

Consider the following characteristics of the off-line analysis of HEP data viewed as object stores of events:

- Large numbers of complex objects must be accessed.
- Data analysis consists of numerically intensive queries, which are normally expressed in Fortran or a similar language.
- Although large numbers of objects are examined in a query, a typical query touches only a small percentage of the data in each object, say 0.1%. Different queries touch
<table>
<thead>
<tr>
<th>file system</th>
<th>object store</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>set of sets</td>
</tr>
<tr>
<td>file</td>
<td>set of objects</td>
</tr>
<tr>
<td>record</td>
<td>object</td>
</tr>
<tr>
<td>field</td>
<td>attribute</td>
</tr>
<tr>
<td>distributed file system</td>
<td>distributed object store</td>
</tr>
<tr>
<td>hierarchical file system</td>
<td>multi-level object store</td>
</tr>
</tbody>
</table>

Table 1: Some analogies between file systems and object stores.

<table>
<thead>
<tr>
<th>relational database</th>
<th>object oriented database</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>store of objects</td>
</tr>
<tr>
<td>table</td>
<td>set of objects</td>
</tr>
<tr>
<td>tuple</td>
<td>object</td>
</tr>
<tr>
<td>structured query language</td>
<td>object query language</td>
</tr>
</tbody>
</table>

Table 2: Some analogies between relational and object oriented databases.

different portions of the data.

The data is historical in the sense that it is written once and rarely updated. Other applications which share these characteristics include: the analysis of seismic data, the analysis of consumer marketing data, and anomaly detection. Collectively, these applications may be viewed as data mining applications. Each is looking for an application specific needle in an application specific haystack.

Ptool was designed for these types of applications. The intention was to support
1. access to large numbers of complex objects distributed on a variety of storage media;
2. and numerically intensive queries.

These requirements are noteworthy in what they omit: there is no direct support for transactions, back up, or recovery. The decision was made to trade performance for functionality in these areas, since the data is primarily historical.

4 Concepts

In this section, we describe a reference model for computing with distributed stores of objects. As throughout the notes, this is an expository treatment of research described elsewhere. In particular, this section is adapted from [11] and [17].

The reference model uses just three primary concepts:

*Objects.* The data is assumed to be organized into objects.

*Processes.* Computation consists of communicating processes which act upon objects.

*Nodes.* Objects and processes are distributed among nodes in a network, with two nodes being able to exchange objects in case there is a path connecting them.

In addition, the following three concepts, which are defined in terms of the basic concepts, are also important:

*Persistent Objects.* An object is called *persistent* if it exists independently of the process which creates it. Otherwise an object is called *transient.* In other words, transient objects basically reside in memory, while persistent objects also reside on disk or other permanent media.

*Brokers.* Processes may make requests of other processes, called brokers, to take certain actions on objects.

*Folios.* For efficiency, objects are collected into physical units called folios, which the system manages. Components of the system respond to distributed requests
for objects and folios. In turn, folios may themselves consist of smaller physical units, called subfolios or segments.

We now elaborate on these concepts.

As already mentioned above, an object is an abstract data type together with some functions, or methods, for creating, accessing, and modifying it. From the viewpoint of object stores, an element of the abstract data type is called an attribute of the object. Objects which share the same attributes and methods form classes. We assume that each persistent object is assigned a unique id, called a persistent object id and that each object belongs to one persistent store or store.

An object manager creates, updates, and accesses persistent objects. Object oriented databases, in addition to providing persistence, also provide additional functionality such as transactions, back up, and recovery.

It is convenient to organize objects into logical units called collections. Collections may contain the actual objects themselves or pointers to the objects. There are many types of collections. For example, unordered collections without duplicates are called sets; ordered collections are called lists. A collection itself is an object. A store is a top level collection which for this reason is handled somewhat differently by the object manager.

Since collections are themselves objects, functions may be applied to collections. If the result is another collection, the function is called a query. Notice that this definition of query includes queries that select objects for inclusion into the output collection, queries that compute new objects from the input objects, and queries which both select objects and compute new objects. When new objects are produced by a query, this additional data is sometimes called derived data.

Nodes are the abstractions of the different physical devices in a distributed computing environment. Nodes abstract physical memory, virtual memory, disks, tapes, workstations, disk arrays, clusters of workstations, etc. For example, brokers can move folios from node to node, abstracting the movement of folios from workstation to workstation, from tape to disk, and from disk to disk.

5 Physical Data Model

The object manager must ultimately pass objects to a storage manager in order for the objects to be persistent. There are several possibilities: the storage manager may be part of the object manager itself, the storage manager may be accessed through an API, the storage manager may be accessed through a file system and its API, or the storage manager may be accessed through a hierarchical storage system and its API. There are several versions of ptool available. Ptool32 accesses the storage system through a file system API, while ptool64 accesses the storage system through a hierarchical storage system API. Although the API's are essentially the same, providing an additional level of caching when interfacing to a hierarchical storage system provides better performance. For this reason, ptool32 uses single level caching as in Diagram 1, while ptool64 uses a multi-level caching as in Diagram 2.

The current version of ptool64 (version 0.6) interfaces to a hierarchical file system through the abstraction of a bitfile, as described in the IEEE Hierarchical Storage System Reference Model, version 4 [2]. It is a current area of research to design the most appropriate interface between object managers, file systems, and storage systems. Later versions of ptool plan to provide direct interfaces to storage systems, as in version 5 of Storage System Reference Model [20].

As mentioned, to increase performance objects are gathered into physical units called folios and subfolios or segments. Objects may span one or more segments, and even one or more folios. We assume that each folio has a folio id attached to it, and each segment has a segment id attached to it. Briefly in ptool64, there is a cache manager, called a segment cache manager, which manages segments and another, called a network cache manager or folio cache manager, which manages folios. The folio cache manager
Figure 1: The design of ptool32. Ptool32 provides uniform access to a persistent space of objects. The Persistent Object Manager provides persistence for complex objects by mapping physical extents of virtual memory called segments. When a segment that is not currently mapped is needed, the Persistent Object Manager generates a fault to the Segment Cache Manager. If the segment is available in the local segment cache, it is mapped into virtual memory. Otherwise a fault is generated, and the segment is obtained from the appropriate data file. This figure is from [10].
extracts folios from bitfiles, while the segment cache manager extracts segments from folios. Bitfiles are managed by the hierarchical storage system. See Diagram 2. In ptool32, there is simply a segment cache manager which interfaces directly to the file system, as illustrated in 1.

Persistent objects must be stored on some permanent media; because of this, each persistent object is associated with a physical address, or physical id. For example, a persistent object may have a virtual memory address or byte location within a file associated to it in this way. Note that this physical address may change. For example, this happens if the object is cached or migrated. In addition, each object has a logical object id associated to it. By assumption, this does not change, despite any changes to the object. There are several possibilities:

1. The pid is always the physical address just described.
2. The pid is always the logical id and a table is maintained between the physical addresses and the logical ids.
3. The physical address can be computed from the logical id, and perhaps some auxiliary information.
4. The pid is sometimes the physical address and sometimes the logical id; tables are maintained as necessary. The pid is said to swizzle between the two.

Ptool uses the third possibility.

6 Architecture and Implementation

In this section, we describe the architecture of ptool32 and ptool64. Ptool64 provides a 64 bit persistent address space, while ptool32 provides a 32 bit persistent address space. The main difference in architecture is that ptool64 uses multi-level caching and migration, while ptool32 uses single level caching and migration. In the current implementation (version 0.6), there are additional differences arising from the fact that the languages, compilers, and operating systems used do not support 64 bit pointers. A future release of ptool is planned which does exploit 64 bit systems. We now describe the various components of the system: the Persistent Object Manager, the Segment Cache Manager, and the Folio Cache Manager.

The Persistent Object Manager is responsible for creating, storing, and accessing complex persistent objects. The Persistent Object Manager is implemented using virtual memory mapping techniques [19]. The Persistent Object Manager uses the UNIX function mmap to map identified disk regions to virtual memory. When objects in this identified region are referenced, the virtual memory system is responsible for moving them in and out of memory. The Persistent Object Manager divides the portion of virtual memory it uses into several slots. The segments or subfolios managed by the Persistent Object Manager are moved into one of the slots when needed. If a referenced object is contained in a segment which is already mapped, then the referenced object is returned. If not, the Persistent Object Manager generates a fault to the Segment Cache Manager.

The Segment Cache Manager maintains a cache of segments. If the requested segment is available in this cache, it is returned to the Persistent Object Manager, which frees a slot of virtual memory it manages, and maps the segment into the slot. On the other hand, if the segment is not available, a fault is generated. In ptool32, this fault causes ptool32 to retrieve the required segment from the appropriate data file. See Figure 1.

In ptool64, an additional level of caching is used. The segment fault causes the Folio Cache Manager to search the cache of folios it maintains. If the segment is available in one of these folios, it is extracted and returned to the Segment Cache Manager. If not, the Folio Cache Manager generates a fault to the hierarchical storage system which obtains the required file and extracts the folio. See Figures 2, 3.

In applications, ptool32 is usually used on a single workstation with an attached local disk and one level of caching suffices. while ptool64 is usually used in a networked
Figure 2: The design of ptool64: View 1. Ptool64 provides uniform, scalable access to a persistent space of objects. The Persistent Object Manager provides persistence for complex objects by mapping physical extents of virtual memory called segments. When a segment that is not currently mapped is needed, the Persistent Object Manager generates a fault to the Segment Cache Manager. If the segment is available in the local segment cache, it is mapped into virtual memory. Otherwise a fault is generated and passed to the Folio Cache Manager. The segment may be available in the Folio Cache Manager’s segment cache; if not, a fault is generated to the hierarchical storage system to retrieve the corresponding physical collection of segments called a folio which contains the needed segment. Pre-emptive prefetching is used to improve performance. This figure is from [12].
Figure 3: The design of ptool64: View 2. The multi-level caching algorithm implemented in version 0.6 of ptool64 is designed to exploit the structure of a heterogeneous computing network with a high performance connection to hierarchical storage. The Network Cache Manager is assumed to transport folios over a high bandwidth channel between the hierarchical storage system and a node on the network. Segment Cache Managers run on other nodes and transport segments over the network at a lower bandwidth. We assume that the Persistent Object Managers which run on each node have a high bandwidth means of transporting objects between virtual memory and the local disk. This figure is from [12].
environment with local and remote disks and other storage media. Our experience has shown the advantages of multi-level caching in these environments [12].

A Persistent Objects and ptool32

In this section, we describe how to create and access persistent objects, following [10], using the example of events. For simplicity, we put each event directly in the store. More commonly, events would be gathered together into sets and the sets placed in the store, as we describe in the next section.

There are three main steps to create and access a store of events: first, a scheme for the store is designed, which defines the objects and their attributes; second, a store is created and populated with objects; and third, the store is queried for objects meeting specified criteria.

1. The first step is to define the schema for the store by defining the objects and their attributes. This is done by creating a file, in this case event.h, containing the classes defining the objects, and the member variables defining the attributes. See Figure 4. In this example, a Lepton object consists of a four vector and a charge, and an Event object consists of two Leptons, plus some additional data.

2. The second step is to populate the store. Figure 7 contains the code to do this. It begins with the statement #include "event.h" defining the schema. The next step is to create a store with the statement store a("PsiEvents");. This creates a store with the internal handle a, and the external name PsiEvents. If the store PsiEvents already exists, then this program would append objects to it. The store must also be closed with the statement: a.close().

To make an object persistent, the standard C++ declaration

```cpp
Event *e;
e = new Event;
```

is replaced by

```cpp
Event *e;
e = new(&a) Event;
```

In order to access the object later, an entry point must be created for it in the store with the statement:

```cpp
a.add(e);
```

Attributes for persistent objects and transient objects are accessed in the same way:

```cpp
e->vertex=(double)drand48();
```

assigns a random double to the vertex attribute of the Event object e.

Finally, note that if a persistent object contains pointers to other persistent objects, then these must also be made persistent, as with

```cpp
e->lepton1=11=new(&a) Lepton
```

But note that it is not necessary to create an entry point for these, since they are automatically accessed whenever their parent object is accessed.

3. The third step is to query the persistent store of objects. Figure 8 contains a simple program to do this. The schema is included at the beginning: #include "event.h". The store with the external name PsiEvents is opened with the internal handle b using the statement

```cpp
store b("PsiEvents");
```

and later closed with b.close().

To loop through all objects in a store is easy:

```cpp
Event *e;
for (e=(Event*)b.first(); b.more(); b=(Event*)b.next())
```
Note that this assumes that all objects in the store are of the same type. Also note that the objects must be explicitly cast into objects of the correct type.

To summarize: first, define the classes for the persistent objects; second, populate the store by persistently allocating instances of the class using

```c++
store a("PsiEvents");
Event *e;
e = new(&a) Event;
```

third, iterate over the objects in the store and make selections.
#ifndef __EVENT__
#define __EVENT__

#include <iostream.h>
#include "ptool32.h"

enum populationMode { randomPopulation = 0, sequentialPopulation = 1);
class Lepton;

class Event {
public:
   int runNumber;
   int eventNumber;
   double vertex;
   Lepton *lepton1;
   Lepton *lepton2;

   Event(populationMode, store*);
   friend ostream& operator << (ostream& os, Event& e);
};

class Lepton {
public:
   double p[4];
   double charge;

   Lepton(populationMode);
   friend ostream& operator << (ostream& os, Lepton& l);
};

#endif

Figure 4: This figure contains the schema for a very simple event object containing just a few attributes. With ptool, objects are defined using C++ classes and attributes are nothing more than data members of the corresponding classes. This is the simplest of three examples showing how ptool can be used to create eventstores. The first two examples use ptool32 and the third example uses ptool64.
#include <iostream.h>
#include "event.h"

extern "C" {
    void srand(int);
    long rand();
    long random();
    double drand48();
}

Event::Event(populationMode m, store *a) {
    if (m==randomPopulation) {
        Lepton *l1, *l2;
        runNumber=(int)rand()%10000;
        eventNumber=(int)rand()%10000;
        vertex=(double)drand48();

        lepton1=l1=new(a) Lepton(randomPopulation);
        lepton2=l2=new(a) Lepton(randomPopulation);
    }
};

Lepton::Lepton(populationMode m) {
    if (m==randomPopulation) {
        p[0]=(double)drand48();
        p[1]=(double)drand48();
        p[2]=(double)drand48();
        p[3]=(double)drand48();
        charge=(double)drand48();
    }
};

Figure 5: This figure defines the methods for the eventstore. With ptool, objects are defined using C++ classes and methods are nothing more than member functions of the corresponding classes. To simplify the code for this simple example, the values of the attributes are defined randomly.
ostream& operator << (ostream& os, Event& t) {
    os << "Event: \n";
    os << "run# = " << t.runNumber << "\t";
    os << "event# = " << t.eventNumber << "\t";
    os << "vertex = " << t.vertex << "\n";
    os << *(t.lepton1);
    os << *(t.lepton2);
    os << "\n";
    return os;
}

ostream& operator << (ostream& os, Lepton& l) {
    os << "Lepton: \n";
    os << "p0 = " << l.p[0] << "\t";
    os << "p1 = " << l.p[1] << "\t";
    os << "p2 = " << l.p[2] << "\t";
    os << "p3 = " << l.p[3] << "\n";
    os << "t \t charge = " << l.charge << "\n";
    return os;
}

#include <string.h>
#include <stdio.h>
#include "ptool32.h"
#include "event.h"

void main() {
    store a("PsiEvents");
    Event *tmp;

    for (int i=0; i<10; i++) {
        tmp = new(&a) Event(randomPopulation,&a);
        a.add(tmp);
    }
    a.close();
}

Figure 6: The methods for a simple eventstore created with ptool32 (continued).

Figure 7: This figure shows how to populate a simple eventstore containing 10 events using ptool32.
```cpp
#include <iostream.h>
#include "event.h"

extern "C" {
    void srand(int);
    long rand();
    long random();
    double drand48();
}

Event::Event(populationMode m, store *a) {
    if (m==randomPopulation) {
        Lepton *11, *12;
        runNumber=(int)rand()%10000;
        eventNumber=(int)rand()%10000;
        vertex=(double)drand48();

        lepton1=11=new(a) Lepton(randomPopulation);
        lepton2=12=new(a) Lepton(randomPopulation);
    }
};

Lepton::Lepton(populationMode m) {
    if (m==randomPopulation) {
        p[0]=(double)drand48();
        p[1]=(double)drand48();
        p[2]=(double)drand48();
        p[3]=(double)drand48();
        charge=(double)drand48();
    }
};
```

Figure 8: This figure shows how to access an eventstore using ptool32 by looping through all the events.
B Persistent Sets

Sets have two fundamental roles in object stores:
1. Sets organize collections of objects, just as directories organize collections of files. Sets may contain other sets of objects, just as directories may contain other directories of files.
2. Queries map one set of objects to another set of objects, by some combination of selecting objects and computing new objects from the input set. From this viewpoint, sets are analogous to tables in relational databases, since relational queries map one or more input tables to an output table.

There are a variety of different ways of constructing sets; in this section, we describe one of the most basic ways in order to introduce the idea. In practice, one would use more elaborate variants, or alternatives such as templates. To define our illustrative set class, one uses an auxiliary class

```cpp
class Link {
public:
    Link() { next = 0; }
    Link *Next() { return next; }

protected:
    Link *next;
};
```

Then to define a set of Events for example, one modifies the class for Event to inherit Link

```cpp
class Event : public Link {
};
```

A Set class can then be defined by simply creating pointers to the head of the set, to the current position in the set, and providing methods to add and delete elements from the set. Such a set class will allow one to create sets of any object which inherit Link. See Figures 9 and 10.

Given the classes Link and Set, persistent sets for the store

```cpp
store a("PsiEvents");
```

can be defined as follows:

1. Define a persistent set in the store a

   ```cpp
   Set *PsiSet;
   PsiSet = new(&a) Set;
   ```

   Note that the first statement declares that PsiSet is a pointer to a Set, while the second statement defines the variable. The definition of a variable allocates storage for it. In this case, the presence of the over loaded new indicates that the storage is persistent.

2. Define a persistent element of the set

   ```cpp
   Link* tmp;
   tmp = new(&a) Event();
   ```

   This gives us a persistent Event which, since it inherits the class Link, can be an element of a persistent set. At this point, although tmp is persistent, it is not part of any set. Furthermore, there is no way to reference it in the store, since no entry point for it has been created.

3. The statement

   ```cpp
   PsiSet->Add(tmp);
   ```

   adds the persistent tmp to the set PsiSet.
#ifndef __SET__
#define __SET__

#define print_set_element(objLink, genLink) \
    cout << *( (objLink *)genLink );

class Link {
    friend class Set;
public:
    Link() { next = 0; }  \
    Link *Next() { return next; }
protected:
    Link *next;
};

class Set {
public:
    Set() { head = tail = current = 0; length = 0; }
    Set& Add( Link* );
    Set& AddFirst( Link* );

    long Length() { return length; }
    Link* Next();
    Link* First() { return head; }
    Link* Last() { return tail; }

    void Discard();
    void reset() { current = head; }
protected:
    Link *head;
    Link *tail;
    Link *current;
    long length;
};
#endif

Figure 9: The class definition for a very basic container class.
```c
#include <iostream.h>
#include "set.h"

Set& Set::AddFirst( Link *newNode ) {
    length++;  
    newNode -> next = head;  
    head = newNode;  
    if ( !tail ) tail = current = newNode;

    return *this;
}

Set& Set::Add( Link *newNode ) {
    length++;  
    newNode -> next = 0;  
    if ( tail ) {
        tail -> next = newNode;  
        tail = newNode;  
        current = ( !current ) ? newNode : current;  
    }
    else
        head = tail = current = newNode;

    return *this;
}

void Set::Discard() {
    Link *todelete;

    while( todelete = Next() ) {
        delete todelete;
        head = current;
    }
    head = tail = current = 0;
    length = 0;
}

Link* Set::Next() {
    Link *ret = current;

    current = ( current ) ? current -> next : head;
    return ret;
}
```

Figure 10: The methods for a very basic container class. The container class is simply a linked list with pointers to the first, last and current elements.
#ifndef __EVENT__
define __EVENT__

#include <iostream.h>
#include "ptool32.h"
#include "set.h"

enum populationMode { randomPopulation = 0, sequentialPopulation = 1 };

class Lepton;
class Jet;

class Event : public Link {
public:
  int runNumber;
  int eventNumber;
  double vertex;
  Lepton *lepton1;
  Lepton *lepton2;
  Set *event_to_jet_set;
  Event(populationMode, store*);
  friend ostream& operator << (ostream& os, Event& l);
};

class Lepton {
public:
  double p[4];
  double charge;
  Lepton(populationMode);
  friend ostream& operator << (ostream& os, Lepton& l);
};

class Jet : public Link {
public:
  int jet_num;
  int ntrk;
  double phi_jet;
  double eta_jet;
  double pt;
  double mass_jet;
  Jet(populationMode);
  friend ostream& operator << (ostream&, Jet& );
};

#endif

Figure 11: The second version of an eventstore. This eventstore is also created using ptool32, but this time an event can contain one or more jets. This figure defines the classes.
#include <iostream.h>

#include "event.h"
#include "set.h"

extern "C" {
    void srand(int);
    long rand();
    long random();
    double drand48();
}

Event::Event(populationMode m, store *a) {
    if (m==randomPopulation) {
        Lepton *l1, *l2;
        runNumber=(int)rand()%10000;
        eventNumber=(int)rand()%10000;
        vertex=(double)drand48();

        lepton1=l1=new(a) Lepton(randomPopulation);
        lepton2=l2=new(a) Lepton(randomPopulation);

        Link *tmpjet;
        int numberJets = ((int)random()%3)+1;
        event_to_jet_set = new(a) Set;
        for (int i=0; i<numberJets; i++) {
            tmpjet = new(a) Jet(randomPopulation);
            event_to_jet_set->Add(tmpjet);
        }
    }
};

Lepton::Lepton(populationMode m) {
    if (m==randomPopulation) {
        p[0]=(double)drand48();
        p[1]=(double)drand48();
        p[2]=(double)drand48();
        p[3]=(double)drand48();
        charge=(double)drand48();
    }
};

Figure 12: The methods for the event class for the second version of the eventstore. Again, for simplicity, the attributes are given random values. In particular, each event is given 1, 2 or 3 jets, with the number of jets determined randomly.
```
Jet::Jet(populationMode m) {
    if (m==randomPopulation) {
        jet_num=(int)rand()%10;
        ntrk=(int)rand()%100;
        q_frac=(double)drand48();
        phi_jet=(double)drand48();
        eta_jet=(double)drand48();
        pt=(double)drand48();
        mass_jet=(double)drand48();
    }
}

ostream& operator << (ostream& os, Event& t) {
    os << "Event: \n";
    os << "run# = " << t.runNumber << "\t";
    os << "event# = " << t.eventNumber << "\t";
    os << "vertex = " << t.vertex << "\n";
    os << *(t.lepton1);
    os << *(t.lepton2);
    Jet *tmpjet;
    while (tmpjet = (Jet*) t.event_to_jet_set->Next())
        os << *tmpjet;
    os << "\n";
    return os;
}

ostream& operator << (ostream& os, Lepton& l) {
    os << "Lepton: \n";
    os << "p0 = " << l.p[0] << "\t";
    os << "p1 = " << l.p[1] << "\t";
    os << "p2 = " << l.p[2] << "\t";
    os << "p3 = " << l.p[3] << "\n";
    os << "t \t charge = " << l.charge << "\n";
    return os;
}

ostream& operator << (ostream& os, Jet& l) {
    os << "jet: \n";
    os << "jet_number = " << l.jet_num << "\t";
    os << "ntrk = " << l.ntrk << "\t";
    os << "q_frac = " << l.q_frac << "\t";
    os << "phi_jet= " << l.phi_jet << "\n";
    os << "t \t eta_jet = " << l.eta_jet << "\t";
    os << "pt = " << l.pt << "\t";
    os << "mass_jet = " << l.mass_jet << "\n";
    return os;
}
```

Figure 13: The methods for the second version of the eventstore (continued).
4. The statement
   
   a.add(PsiSet);

   adds an entry point for the persistent set PsiSet to the store a, allowing subsequent
   access to the persistent set PsiSet in the same way as for any other persistent object.

   In this way, we can populate stores with persistent sets of persistent objects. See
   Figure 14. Consider a store store b("PsiEvents") consisting of sets of Events. Accessing
   the persistent sets of persistent events consists of two steps:

   1. Since the persistent sets are themselves persistent objects, they can be accessing just
      as any other persistent objects:

      Set *PsiSet;
      for (PsiSet=(Set*)b.first();
      b.more(); PsiSet=(Set*)b.next())

   2. Given a persistent set, one can follow the pointers next from one Link to the next.
      The method Next() from the class Link does this automatically.

      Event *e;
      while (e=(Event*) PsiSet->Next())
        cout << *e;

   See Figure 15.
```c
#include <string.h>
#include <stdio.h>

#include "ptool32.h"
#include "event.h"
#include "set.h"

void main()
{
    store a("PsiEvents");

    Link *tmp;
    Set *PsiSet;

    PsiSet = new(&a) Set;
    for (int i=0; i<10; i++) {
        tmp = new(&a) Event(randomPopulation,&a);
        PsiSet->Add(tmp);
    }
    a.add(PsiSet);

    PsiSet = new(&a) Set;
    for (i=0; i<20; i++) {
        tmp = new(&a) Event(randomPopulation,&a);
        PsiSet->Add(tmp);
    }
    a.add(PsiSet);

    a.close();
}
```

Figure 14: In this version of the eventstore, ptool32 is used to populate an eventstore containing two sets of events. The first set contains 10 events and the second set contains 20 events.
#include <stdio.h>
#include "ptool32.h"

#include "set.h"
#include "event.h"

void main()
{
    Set *PsiSet;
    Event *e;
    store b("PsiEvents");

    int count = 1;
    for( PsiSet=(Set*)b.first(); b.more();
        PsiSet=(Set*)b.next()) {
        cout << "Set" << count++ << "\n";
        PsiSet -> reset();
        while( e =(Event*) PsiSet -> Next() ) {
            cout << *e;
        }
    }
    b.close();
}

---

Figure 15: Ptool32 is used to access an eventstore containing one or more sets of events by looping over all such sets. For each set accessed, each event in the set is accessed in turn.
C  Pointers to Persistent Objects and ptool64

Recall that object ids should be unique even across stores. For this reason, many applications require object ids that are 64 bits or even 96 bits long. This allows enough room to include an id for the store as well as for the object within the store. For example, the current implementation of ptool64 uses object ids of the form

\[
\begin{align*}
\text{int} & \text{ store_id;} & & \text{// 16 bits} \\
\text{int} & \text{ folio_id;} & & \text{// 8 bits} \\
\text{int} & \text{ segment_id;} & & \text{// 8 bits} \\
\text{int} & \text{ offset_id;} & & \text{// 16-21 bits} \\
& & & \text{// 11-16 bits reserved}
\end{align*}
\]

For compilers which support only 32 bit pointers, special effort must be used to work with pointers which are 64 bits long. For example, the statements

\[
\begin{align*}
\text{Event} & \text{ *e;} \\
e & = \text{ new Event;}
\end{align*}
\]

create a 32 bit pointer e. To define 64 bit pointers some other scheme must be used. There are many ways to do this.

One simple method is to define a special class consisting of 64 bit persistent pointers, say pptr. Pointers to objects of a given class can then inherit from the class pptr. For example, the following code fragment declares ppEvent to be a 64 bit pointer to persistent Events.

\[
\begin{align*}
\text{class ppEvent : public pptr} & \{ \\
& \text{ Event* operator->() } \\
& \text{ etc. } \\
& \};
\end{align*}
\]

Another problem is that with most compilers today, \texttt{new} returns a 32 bit pointer. Not all compilers support overloading \texttt{new} to return a different structure. Again, there are several ways of proceeding. One way is to return the 64 bit persistent pointer as an argument to \texttt{new}. In the following example a 64 bit persistent pointer \texttt{tmp} is first declared and then defined upon the return of the overloaded \texttt{new}.

\[
\begin{align*}
\text{ppEvent} & \text{ tmp;} \\
\text{new(&a, &tmp) Event(&a);}
\end{align*}
\]

Since this approach requires that each class have its own persistent pointer class, it is helpful to automate this definition. Since methods cannot return \textit{class definitions}, we use a precompiler directive to do this:

\[
\begin{align*}
& \#define \text{PPTR}(X,Y) \\
& \text{class } Y : \text{ public pptr} \{ \\
& \quad X* \text{ operator->(); } \\
& \quad \text{ etc. } \\
& \text{ Y\& operator=(pptr pp) } \\
& \quad \text{ etc. } \\
& \};
\end{align*}
\]

To define a persistent 64 bit pointer \texttt{tmp}, an instance of the class \texttt{ppEvent} which points to persistent \texttt{Events}, is now easy:

\[
\begin{align*}
\text{PPTR(Event,ppEvent);} \\
\text{ppEvent tmp;} \\
\text{new(&a, &tmp) Event(&a);} \\
\end{align*}
\]

To summarize the previous discussion, ptool64 works exactly like ptool32 except that one defines a class of persistent pointers for each class of persistent objects and uses these pointers to manipulate persistent objects rather than the standard 32 bit pointers supported by the compiler. Figures 16-22 contain the ptool64 version of the example in the last section.
```cpp
#include <iostream.h>
#include "ptool64.h"
#include "pset64.h"

enum populationMode { randomPopulation = 0, sequentialPopulation = 1 };

class Event;
class Lepton;
class Jet;

PPTR(Event, ppEvent);
PPTR(Lepton, ppLepton);
PPTR(Jet, ppJet);

class Event {
public:
    int runNumber;
    int eventNumber;
    double vertex;
    ppLepton lepton1;
    ppLepton lepton2;
    Set64 event_to_jet_set;

    Event() {};
    Event(populationMode, store*);
    Event(ppEvent&, store*);
    friend ostream& operator << (ostream& os, ppEvent& l);
};

class Lepton {
public:
    double p[4];
    double charge;

    Lepton() {};
    Lepton(populationMode);
    Lepton(ppLepton&);
    friend ostream& operator << (ostream& os, ppLepton& l);
};
```

Figure 16: This figure contains the class definitions for the third version of the event store. This version is created using ptool64, which requires that the persistent objects be accessed with 64 bit persistent pointers. These are defined for each class using the macro PPTR.
class Jet : public Link64 {
public:
    int jet_num;
    int ntrk;
    double q_frac;
    double phi_jet;
    double eta_jet;
    double pt;
    double mass_jet;

    Jet() {};
    Jet(populationMode);
    Jet(ppJet&);
    friend ostream& operator << (ostream&, ppJet& );
};
#endif

Figure 17: This figure contains additional class definitions. Since set of Jets will be needed, the class Jet inherits the class Link 64.
#include <iostream.h>
#include "event.h"
#include "pset64.h"

extern "C" {
    void srand(int);
    long rand();
    long random();
    double drand48();
}

Event::Event(populationMode m, store *a) {
    if (m==randomPopulation) {
        randomNumber=(int)rand()%10000;
        eventNumber=(int)rand()%10000;
        vertex=(double)drand48();

        new(a, &lepton1) Lepton(randomPopulation);
        new(a, &lepton2) Lepton(randomPopulation);

        pptr tmp;
        int numberJets = ((int)random()%3)+1;
        for (int i=0; i<numberJets; i++) {
            new(a, &tmp) Jet(randomPopulation);
            event_to_jet_set.Add(tmp);
        }
    }
}

Lepton::Lepton(populationMode m) {
    if (m==randomPopulation) {
        p[0]=(double)drand48();
        p[1]=(double)drand48();
        p[2]=(double)drand48();
        p[3]=(double)drand48();
        charge=(double)drand48();
    }
}

Figure 18: This figure and the next two contain the methods for the event class used for the third version of the eventstore, which uses ptool64. This is similar to the methods used for the second version, but somewhat more complicated since persistent objects cannot themselves be accessed, but must be accessed instead through 64 bit persistent pointers.
Jet::Jet(populationMode m) {
  if (m==randomPopulation) {
    jet_num=(int)rand()%10;
    ntrk=(int)rand()%100;
    q_frac=(double)drand48();
    phi_jet=(double)drand48();
    eta_jet=(double)drand48();
    pt=(double)drand48();
    mass_jet=(double)drand48();
  }
}

Event::Event(ppEvent& ppEv, store *a) {
  pptr iterator;
  runNumber=ppEv->runNumber;
  eventNumber=ppEv->eventNumber;
  vertex=ppEv->vertex;
  new(a, &lepton1) Lepton(ppEv->lepton1);
  new(a, &lepton2) Lepton(ppEv->lepton2);
  ppJet tmp; //temp handle new object
  ppJet ppjet; //temp handle for iterating set
  for (ppjet=ppEv->event_to_jet_set.First(iterator);
       ppEv->event_to_jet_set.More(iterator);
       ppjet=ppEv->event_to_jet_set.Next(iterator) ) {
    new(a, &tmp) Jet(ppjet);
    event_to_jet_set.Add(tmp);
  }
}

Lepton::Lepton(ppLepton& l) {
  p[0]=l->p[0];
  p[1]=l->p[1];
  p[3]=l->p[3];
  charge=l->charge;
}

Jet::Jet(ppJet& j) {
  jet_num=j->jet_num;
  ntrk=j->ntrk;
  q_frac=j->q_frac;
  phi_jet=j->phi_jet;
  eta_jet=j->eta_jet;
  pt=j->pt;
  mass_jet=j->mass_jet;
}

Figure 19: The methods for the third version of the eventstore (continued).
ostream& operator << (ostream& os, ppEvent& t) {
    os << "\n------Event------\n";
    os << "runNum=" << t->runNumber << "\t";
    os << "eventNum=" << t->eventNumber << "\t";
    os << "vertex=" << t->vertex;
    os << t->lepton1;
    os << t->lepton2;

    pptr ppjet;
    pptr iterator;

    for (ppjet=t->event_to_jet_set.First(iterator);
         t->event_to_jet_set.More(iterator);
         ppjet=t->event_to_jet_set.Next(iterator)) {
        os << ppjet;
        os << endl;
    }
    return os;
}

ostream& operator << (ostream& os, ppLepton& l) {
    os << "\nLepton: \n";
    os << "p0=" << l->p[0] << "\t";
    os << "p1=" << l->p[1] << "\t";
    os << "p2=" << l->p[2] << "\t";
    os << "p3=" << l->p[3] << "\t";
    os << "charge=" << l->charge;
    return os;
}

ostream& operator << (ostream& os, ppJet& l) {
    os << "\nJet: \n";
    os << "jet_number=" << l->jet_num << "\t";
    os << "ntrk=" << l->ntrk << "\t\t";
    os << "q_frac=" << l->q_frac << "\t";
    os << "phi_jet=" << l->phi_jet << "\n";
    os << "eta_jet=" << l->eta_jet << "\t";
    os << "pt=" << l->pt << "\t";
    os << "mass_jet=" << l->mass_jet;
    return os;
}

Figure 20: The methods for the third version of the eventstore (continued).
```c
#include <string.h>
#include <stdio.h>
#include "ptool64.h"
#include "pset64.h"
#include "event.h"

void main(int argc, char **argv)
{
    if (argc<2) {cout <<"USAGE : pop store_name\n"; exit(0);}

    store a(argv[1]);

    for (int i=0; i<10; i++) {
        pptr tmp;
        new(&a, &tmp) Event(randomPopulation, &a);
        a.add(tmp);
    }

    a.close();
}
```

Figure 21: An eventstore containing 10 persistent events is populated using ptool64. Notice that a persistent 64 bit pointer to each event is returned through an argument to new.

```c
#include <stdio.h>
#include "ptool64.h"
#include "pset64.h"
#include "event.h"

void main(int argc, char **argv)
{
    if (argc<2) {cout <<"USAGE : acc store_name\n"; exit(0);}

    store b(argv[1]);

    ppEvent ppEv;
    for (ppEv=b.first(); b.more(); ppEv=b.next()) {
        cout << ppEv;
    }

    b.close();
}
```

Figure 22: In this figure, ptool64 is used to loop through an eventstore of persistent events.
References


Level-1 Triggers for LHC Experiments

Eric Eisenhandler
Physics Dept., Queen Mary and Westfield College, University of London, London, UK

Abstract
Triggering at the CERN Large Hadron Collider will be far more difficult than at any previous particle accelerator. After briefly outlining the physics requirements, I describe the very severe time constraints and the unprecedented rate reduction needed at level-1. Algorithms for triggering on electromagnetic showers, jets and missing transverse energy are presented, followed by a discussion of muon triggers. I then review the present status and future plans for work on these subsystems. The emphasis is on electromagnetic calorimeter triggers, where we have successfully demonstrated the principles of a prototype level-1 calorimeter trigger system running at the full LHC bunch-crossing rate under test-beam conditions. Next comes a brief account of work on muon triggers. Finally, current work on system timing problems and the trigger central processor logic is discussed. This introduction to level-1 triggering at LHC is essentially a description of the work of the RD27 collaboration. The discussion here is more directed to ATLAS, but much of it could also be applicable in CMS.

1 What the level-1 trigger must achieve
In perhaps ten years time, experiments should start at the Large Hadron Collider (LHC), a proposed new machine to be built at CERN in the existing LEP tunnel. The operating conditions at LHC will be far more difficult than at any previous high-energy particle accelerator, and some of the most formidable problems will be in the trigger system, which must select extremely rare physics events from enormous backgrounds. This will be done using a multi-level scheme whose first stage is its most demanding and novel component. In this section I will start by briefly indicating the kinds of physics signals that must be searched for, and then show what rate reductions must be achieved by the level-1 trigger and how quickly this must be done.

1.1 Physics goals of LHC

The standard model of particle physics
At present, most of what we know about elementary particle physics is described by the so-called standard model, which works surprisingly well as far as it goes. In this picture, all known particles are either leptons (electrons, muons or taus, and their corresponding neutrinos), or are made up of quarks (u, d, s, c, b, t; only t remains to be discovered), or are carriers of the forces — electromagnetic ($\gamma$), weak ($W^{\pm}/Z^0$), and strong (gluon). The leptons and quarks are grouped into three families, and we know that there are no others.

Despite the success of the standard model, there are some things that it just doesn’t explain. Why do the particles have very different masses? (For example, the $\gamma$ is massless while the W and Z are the heaviest particles found so far, almost a hundred times more than a proton.) What is the mechanism whereby the forces seem likely to be unified at high energy but very different at the lower energies we have so far achieved? Why three families rather than just one, or any other number?

Many of the answers about the mechanism behind differences in mass and the forces might be known if we can find the hypothetical Higgs particle and measure its properties. Unfortunately, unlike the search for the W and Z ten years ago, there is no prediction of the
mass of the Higgs; from searches already done it is about as heavy or heavier than the W and Z. Moreover, the production rate (which depends on the mass) will be extremely low. By way of insurance, if the Higgs does not turn up with a mass of ~1 TeV or less, theorists predict that other interesting phenomena will be seen. We therefore need an accelerator with a combination of high energy and very intense beams.

The LHC

At CERN, it is proposed to build the Large Hadron Collider, which stores and collides two counter-rotating proton beams with an energy of about 7 TeV each. LHC would be built in the 27 km circumference tunnel of the existing LEP electron–positron collider, and the luminosity (beam intensity) would be an unprecedented \(1.7 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}\). (Luminosity times cross-section gives the event rate for a specified reaction.) There will probably be two general-purpose experiments at LHC — ATLAS and CMS. LHC’s energy is limited by its use of the LEP tunnel, while the recently-cancelled Superconducting Super Collider (SSC) in Texas would have collided beams of 20 TeV each. There is some trade-off between energy and intensity, and SSC would have had a luminosity of ‘only’ about \(1 \times 10^{33} \text{ cm}^{-2} \text{ s}^{-1}\). This would have made triggering and event reconstruction easier, for reasons described below.

Although the Higgs particle is LHC’s headline attraction, any move upwards into a new energy range can bring both expected results and surprises. Searches for new particles and effects will include supersymmetric partners of existing particles (SUSY) and hints of structure within quarks, as well as the genuinely unpredicted.

How do we find these new particles?

The Higgs is unstable and too short-lived to be detected directly, so it must be detected from its decay products. The easiest and cleanest way is to detect decays into purely leptonic final states — experience with the W and Z has been that it is much more difficult to identify decaying particles when they decay into quarks, which produce ‘jets’ of hadronic (strongly-interacting) particles such as pions. If the Higgs is heavy enough, some very clean but relatively rare final states to look for would be:

\[
H \rightarrow Z^0 Z^0 \rightarrow e^+e^- e^+e^- \text{ or } \mu^+\mu^- \mu^+\mu^- \text{ or } e^+e^- \mu^+\mu^{-}
\]

If the Higgs is relatively light, it can be identified when it decays into photons:

\[
H \rightarrow \gamma\gamma
\]

We can see already that we need a way to trigger on high-energy electrons, photons and muons. But to see other interesting physics, this is not enough. For example, to study the top quark, we might look for the following decay chain:

\[
t \rightarrow bW \rightarrow \text{jet} + e/\mu + \nu
\]

This indicates we should also try to trigger on jets of hadronic particles and on neutrinos.

How do we detect them?

A typical general-purpose experiment, such as ATLAS or CMS, tries to cover as much as possible of the solid angle around the beam-collision region and to classify and measure as many as possible of the particles that emerge from collisions. The experiment is built in layers, and the way it sorts out the particles is like a series of filters — this is indicated very schematically in fig. 1. The differences between charged and neutral particles, and signatures used to classify electrons, photons, hadrons and muons, are summarised in table 1, which describes the detectors in the order that they are encountered by particles moving outwards from the collision point.
Level-1 Triggers for LHC Experiments

Figure 1. A very schematic cross-section through a typical general-purpose experiment.

<table>
<thead>
<tr>
<th>Detector</th>
<th>Particle type</th>
<th>electron</th>
<th>photon</th>
<th>charged hadron</th>
<th>neutral hadron (e.g. n)</th>
<th>muon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Tracker</td>
<td></td>
<td>track</td>
<td>no track</td>
<td>track</td>
<td>no track</td>
<td>track</td>
</tr>
<tr>
<td></td>
<td>Records charged-particle tracks, measures momentum (magnetic field)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electromagnetic</td>
<td>narrow shower of full energy</td>
<td>narrow shower of full energy</td>
<td>start of broad shower</td>
<td>start of broad shower</td>
<td>small signal</td>
<td></td>
</tr>
<tr>
<td>Calorimeter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifies electrons and photons and measures their energies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hadronic Calorimeter</td>
<td>-</td>
<td>-</td>
<td>broad shower of ~full energy</td>
<td>broad shower of ~full energy</td>
<td>small signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifies hadrons and measures their energies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muon Detector</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>track</td>
</tr>
<tr>
<td></td>
<td>Identifies muons and measures their momentum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. How the detectors classify particles.

Neutrinos (and some of the hypothetical particles at the ends of SUSY decay chains) do not usually interact in particle detectors, so we must detect them by their absence — we try to measure everything else and see if energy is conserved. If there is a lot of missing energy, then it is possible that a non-interacting particle carried it off. However, since much of the energy in a collision goes unseen down the beam pipe, we only look at energy components transverse to the beam — what we seek is called **missing transverse energy**.

1.2 Speed, rate reduction and deadtime

**Physics vs. background**

The proton beams in LHC are bunched such that counter-rotating bunches collide every 25 ns (40 MHz). Although the cross-sections for producing interesting physics are very low, with detectable events as seldom as once every few hours, days or weeks, the proton–proton inelastic collision cross-section is very large, so that on average at full LHC luminosity each bunch-crossing produces 20 to 30 visible collisions. In other words, the total p–p interaction rate is about 1 GHz, which is unprecedented. Most of these interactions do not produce particles with very high transverse energy or missing transverse energy, so the interesting physics should still be visible above this unavoidable background. However, the implications of the high background for the detectors, triggering, and the event reconstruction are extremely serious. If the SSC had been built, its combination of higher energy and lower luminosity, with an average of only one or two interactions per bunch-crossing, might have been easier to handle.
Level-1 Triggers for LHC Experiments

In an LHC experiment, these 20–30 events produce on the order of 500 tracks in the central tracking system and ~1000 particles in the calorimeters. Only the muon detector, shielded by the mass of the inner detectors, has a relatively low rate from interactions (but may still have high rates from beam halo and neutron-induced random background). In order to be able to sort out this mess, LHC experiments will build the detectors so that they are subdivided into a huge number of small units, or cells, in order to ensure low ‘occupancy’ of individual detector elements and so minimise pile-up.

Trigger levels

The trigger system must reduce the 1 GHz interaction rate, bunched at 40 MHz, to a rate that can be recorded permanently for later analysis. This is probably limited to ~10 bunch crossings per second, which means an enormous reduction factor of $10^6$–$10^7$. If the trigger misses anything that is interesting it is lost forever, so despite this huge rejection factor it must be very efficient and introduce as little deadtime as possible. In ATLAS a three-level system is planned; CMS hopes to omit the second level. Either way, the level-1 trigger is the most difficult because it has the highest rejection factor and must work too quickly to reap the benefits of programmable devices like microprocessors and digital signal processors. It must also be flexible enough to allow, as much as possible, new and unanticipated triggers if hints of new physics appear.

To illustrate just how demanding LHC will be, let’s compare some of the parameters with previous machines. LEP, with its leisurely 11 µs (originally 22 µs) bunch-crossing interval and low trigger rates, is almost irrelevant. The only machine with a remotely comparable bunch-crossing rate to LHC’s 25 ns is HERA, the e–p collider at DESY, which has a 96 ns period. This sort of rate requires that event readout and triggering be pipelined, and in that respect it provides a good training ground for LHC. However, the interaction rate at HERA is orders of magnitude lower, so trigger selectivity does not have to be high. In fact, no one has ever run with many events per bunch crossing. The closest situation is at the former CERN and present Fermilab antiproton colliders, with rates ~1 event per bunch crossing to be handled by level-1 triggers. However the ~4 µs bunch-crossing time makes pipelining unnecessary since the level-1 triggers can do their job before the next bunch crossing. These colliders also have the current record for level-1 trigger rejection, roughly a factor of ~$10^3$. As will be seen below, at LHC this will have to rise to a factor of $10^4$–$10^5$. As for the detectors, the number of calorimeter channels per experiment at LHC will be ~$10^5$ and the number of central tracker channels will be ~$10^7$. At LEP or HERA these numbers are both ~$10^4$. Finally, comparing the total interaction rate with the ‘Nobel Prize’ physics signal, at the CERN Collider this was ~$10^{-9}$ but at LHC it may be ~$10^{-13}$!

System architecture

An overall diagram of the trigger and data acquisition system is shown in fig. 2. The 25 ns between bunch crossings is far too short to do any triggering, so in order to be able to examine the interactions that take place at each bunch crossing both the trigger and the data to read out from each detector are pipelined. This means that at level-1 there is no deadtime, so that no data is lost while the trigger decision to keep or reject the data is being made. Once the level-1 trigger has made its rate reduction, we can afford to let levels 2 and 3 take longer and if necessary produce deadtime (minimised at level-2 by multiple buffering). Fig. 2 indicates the hoped-for rate reductions, as well as showing which R & D collaborations are doing relevant work.

I will not describe levels 2 and 3, which have sufficient time to use programmable processors, but the ways they can gain on level-1 are that level-2 has access to more detailed
Information from the detectors (e.g. full detector channel granularity) and the electron trigger has access to tracker information. Level-3 combines information from all the detectors so it has a picture of the entire bunch crossing. Level-2 has ~1 ms to make its decision, and can set more demanding thresholds than level-1. It will probably be divided into local processing (using information from level-1 telling it where to look; these are called ‘regions of interest’) and global calculations based on an entire detector.

![Diagram of Level-1 Triggers for LHC Experiments]

**Figure 2.** Trigger and data acquisition system architecture.

Detector data is read out after a level-1 trigger. This typically takes on the order of 10 μs for one bunch-crossing worth of data from the detectors, which means that the bunch-crossing readout rate allowed past level-1 is limited to 10–100 kHz. This is what determines the rejection rate of the level-1 trigger. The other important parameter is how much time the trigger has to do its job, and this is determined by how long the readout pipelines on the detectors are. Pipelines are expensive, so the trigger is made as fast as possible. On the other hand, it would be a major calamity if an experiment is built and then it is found that the pipelines on all the detector channels are too short, so it is important to do the calculations very carefully and to build in a safety margin. The present consensus is to have pipelines about 2 μs long, which means they must store data from about 80 bunch crossings.

Having pipelines 2 μs long does not mean that the level-1 trigger has all of that time to do its job. Trigger input signals must be collected from the detectors and the trigger decision (yes or no) broadcast to the readout system, and all this can take a surprisingly long time. In fact, the level-1 trigger must do its work in only a few hundred nanoseconds! That is why it must use hard-wired processor logic rather than microprocessors or digital signal processors. However, as I will show, this does not mean the trigger is completely inflexible. Programmable thresholds and other options do allow a degree of versatility, even though the algorithms themselves are wired in.

**Timing and synchronisation problems**

The 25 ns interval between bunch crossings is of the same order as the time it takes to collect signals across detector elements or to send them down cables or optical fibres. In fact, for an experiment like ATLAS, which is roughly 20–30 m in its overall dimensions, at the instant
when the proton bunches collide the particles produced by the previous bunch crossing are only about half-way out through the experiment, and those from the bunch crossing before that are only just leaving the outermost corners of the muon system. This means that putting together the data from different detectors, or even different parts of a detector, presents a huge synchronisation problem. It is all too easy to combine data from different bunch crossings, and this would be disastrous. Later I will describe some work in this area.

**Front-end electronics; analogue vs. digital**

The front-end detector electronics will be mainly located on the detectors, which means that it must be compact and use as little power as possible. The signals must be brought out using a compact medium — optical fibres have advantages over copper cable. Inaccessibility means it must be reliable, have good fault detection and if possible have redundancy built in. For electronics near the inside of the experiment, radiation hardness is required.

One of the issues in meeting these requirements is whether the front-end should be analogue or digital. Many of the arguments here apply to the trigger logic as well, even though we hope that it will not be mounted on the experiment itself. Some important points are:

**Analogue**
- Low power dissipation and compact electronics
- Fewer connections needed, but must be high quality
- Analogue pipeline or storage very difficult to make with wide enough dynamic range for calorimeter
- Problems with noise pickup and cross-talk

**Digital**
- Higher power consumption
- More connections needed, but need not be high quality
- Digitisation with sufficient dynamic range (16 bits) and precision (10 bits) for calorimeter very difficult at rates ~40 MHz
- However, digitisation for trigger purposes (8–9 bits) is straightforward
- More versatile, since easier to fan-out signals to logic
- Predictable — algorithm can be exactly matched in software, so easier to check operation
- Easy to test under computer control
- Easy to correct signals (see RAM lookup table discussion below)
- Fewer problems with noise pickup and cross-talk

The level-1 trigger work described here uses entirely digital solutions. The past experience of most people in RD27 on a number of large experiments has been with digital rather than analogue triggers, and we have benefitted from the advantages listed. Our approach is therefore to digitise the signals, at least for trigger purposes, as soon as we can.

2 **Detectors and algorithms**

The overall organisation of the level-1 trigger is shown schematically in fig. 3. Each of the triggers, shown by squares in the diagram, will be discussed in turn. However, there will be a strong emphasis placed on electron and photon triggers. This is because it is probably the most difficult calorimeter trigger, and represents by far the most serious rate problem. Therefore, that is where the most work has been done up until now.
2.1 Electromagnetic showers (electrons and photons)

We need to trigger on electrons and photons having transverse energies of 50 GeV or less. There is a huge background rate, mainly from pions ($\pi^\pm$ and $\pi^0$) in jets. The distinctive features of electron or photon showers in the calorimeters are that the showers are relatively narrow and do not penetrate to the hadron calorimeter. The electrons or photons we seek also tend to be isolated, unlike particles in jets, so we can ask that there be little calorimeter activity nearby.

Electron and photon trigger algorithm

The electron and photon trigger algorithm that we favour is illustrated in fig. 4. It is based on simplified information from the calorimeters — the trigger cells are sums over the entire depth of the e.m. calorimeter or the hadronic calorimeter, and calorimeter cells feeding them are summed laterally to cover an area of 0.1 radians in azimuthal angle $\phi$ by 0.1 units of pseudorapidity ($\eta = -\ln \tan \theta/2$ where $\theta$ is the polar angle). This would result in about 4000 trigger channels in the e.m. calorimeter and a similar number in the hadronic calorimeter. As we will see shortly, this reduced granularity of channels is good enough for our purposes.

Figure 3. Block diagram of the first-level trigger system.

Figure 4. The electron/photon trigger algorithm.

The algorithm has two parts. The first sets a threshold on energy in a cluster, defined as the sum of two adjacent trigger cells either vertically or horizontally. Using only one cell would result in very imprecise thresholds because showers tend to spread between cells; on the other hand using four cells does not produce much improvement — the evidence will be shown below. The second part of the algorithm looks at the summed energy in a ring of 12 trigger cells. If this is above a fairly low threshold then the shower seen is probably part of a jet, so
Level-1 Triggers for LHC Experiments

reject it. We also look to see if there is energy above a low threshold in the hadronic calorimeter cells just behind the 16 e.m. cells, and if that is true the shower is probably not electromagnetic. Particularly in forming the e.m. and hadronic isolation sums, which combine a fairly large number of cells, we must beware of noise pickup and low-energy signals from the pile-up of multiple events in one beam crossing. To do this, we exclude any signal in a single trigger cell that is below a rather low $E_T$ threshold, e.g. 1 GeV.

How well does the e.m. algorithm perform?

The electron/photon algorithm just described must reduce the event rate to an acceptable value, and it must perform efficiently so that we do not lose many interesting events or have to make large corrections to the data. Its performance has been studied using a Monte Carlo simulation of the ATLAS experiment. The rate rejection is shown in fig. 5 as a function of the cluster threshold used. The upper points show the jet background rate that must be reduced, and below it is a curve showing the effect of a simple cluster threshold without using any isolation criteria. Broadly speaking, this reduces the rate by about an order of magnitude, but this is not quite enough to be acceptable for level-2 (see fig. 2). Applying isolation in the ring of e.m. cells and in the hadronic cells gains a further order of magnitude, giving rates that would be acceptable to level-2 at transverse energy thresholds of 40–50 GeV. The lower points show the rate for production of individual $\pi^0$s — the electron/photon trigger has difficulty distinguishing these, since they decay to two photons (though isolation helps), so there is no point in devising an algorithm that does very much better than this.

![Figure 5. Trigger rate vs. cluster threshold at $|\eta| < 2.5$ with and without isolation, and for jets and $\pi^0$s. (‘Threshold’ corresponds to 95% efficiency for electrons.)](image1)

![Figure 6. Efficiency vs. cluster threshold for 50 GeV electrons at $|\eta| = 0$, for various algorithms.](image2)

The efficiency of the algorithm is shown in fig. 6. What is shown is its performance for electrons of transverse energy 50 GeV, as a function of the cluster threshold used. If the threshold is very low we should always trigger on the electrons, but if the threshold is around 50 GeV we would expect to be less efficient. The sharpness of the edge is a measure of how well an algorithm performs. On the graph the three obvious possibilities are shown: one cell, two cells, and four cells. Because the shower is not often fully contained in one cell, the one-cell curve has a very ‘soft’ edge, with efficiency reduced even for a threshold of around 30 GeV. The two-cell and four-cell results are much sharper, and in fact are nearly identical. This means that there is little to gain in efficiency by going to a four-cell algorithm. And
since the jet background rate is roughly proportional to the area of calorimeter used in the threshold, the four-cell algorithm would have rate reduction about twice as bad as the two-cell algorithm.

2.2 Jets

In order to trigger on jets, we simply propose to apply a threshold to the total transverse energy summed over the full e.m. and hadronic calorimeter depth in a window that roughly matches the size of jets, for example 0.8 radians in $\phi$ by 0.8 units of pseudorapidity. The windows must overlap so that the energy of a jet is always picked up with little leakage, and so a ‘sliding’ 2 x 2 scheme will be used. The basic cells used are four times the size of the e.m. cells. This trigger is rather straightforward to implement.

2.3 Missing transverse energy

The electron/photon and jet triggers are done using localised regions of the calorimeters. To look for missing transverse energy we must form a global sum of the x and y components of energy everywhere in the calorimeter (the beam direction is z), so here we could face timing problems. Each trigger input signal is converted to components based on its position in the experiment using RAM lookup tables (see below):

$$E_{T_x} = E \sin \theta \cos \phi; \ E_{T_y} = E \sin \theta \sin \phi$$

We then form the square of the total transverse energy:

$$\left( \sum E_{T_x} \right)^2 + \left( \sum E_{T_y} \right)^2$$

and compare it to a threshold. If this is high it indicates the possibility of an undetected high-energy particle such as a neutrino.

2.4 Muons

The principle of the muon detectors is that almost all other types of particle except neutrinos (which are not normally detected) are absorbed by the great mass of material that must be traversed to reach them. High energy muons emerge relatively unaffected by bending in the central magnetic field and multiple coulomb scattering in the material, so if their tracks are detected they should point back to somewhere in or near the region where the beams collide. Therefore, muon triggers try to rapidly reconstruct tracks in the muon detector and decide whether they have angles and positions consistent with a high energy muon coming from the collision and only slightly bent. The degree of bending allowed effectively sets a threshold on momentum or transverse-momentum. This is sketched conceptually in fig. 7. The trigger looks for correlated hits in two or more well-separated layers of the muon detector. (In ATLAS, the present idea is to have two double layers at the centre of the muon toroid and a triple layer at the outside. The trigger requirement and threshold depend on luminosity.)

![Figure 7. Sketch of the principle of muon triggering. The acceptance cone is shaded.](image)
The main backgrounds, which are worst for transverse momenta below about 6 GeV/c, are due to decays of pions and kaons inside the experiment, hadrons occasionally leaking through all the material, beam halo, cosmic rays, and random background due to neutrons. The problems in triggering at an acceptable rate depend on the threshold: according to simulations, the Higgs should not be too hard (threshold of perhaps 20 GeV/c), but in order to do b-quark physics a threshold of around 6 GeV/c transverse energy is needed and that seems to be much more difficult.

A very serious problem is the enormous size of the muon detectors, since they extend to the very outside of the experiment, covering an area of about a thousand square metres. There will clearly be problems with synchronising the signals to a few nanoseconds due to both the variable flight time of the muons and signal propagation delays.

3 Implementation: R & D work and test results

3.1 Calorimeter trigger (bit-parallel approach)

Initial stages: trigger ADCs and RAM lookup tables

Before the calorimeter signals are put through the trigger algorithm, they will first be digitised and treated using RAM lookup tables. This is illustrated in fig. 8. The latches after each stage allow pipelining: while one bunch-crossing’s data is being treated in the RAM, the next bunch crossing is being digitised. Simulation indicates that digitisation to 8 or at most 9 bits is good enough; if necessary a non-linear digitisation can extend the dynamic range but is probably not necessary. It is not yet decided whether to use dedicated trigger ADCs or to have them integrated into digital front-end readout logic, e.g. using chips developed by the FERMI collaboration. In any case, the flash-ADC performance needed is readily available.

![Figure 8. Initial stages of the calorimeter logic: ADC and RAM.](image)

![Figure 9. Conversion of calorimeter energy to ADC signal, and its treatment using RAM lookup table.](image)

The use of RAM lookup tables solves several problems quickly and simultaneously, and is an old trick (used by some of us in UA1 around 1980). The incoming data is the address to a RAM whose contents have the desired ‘answer’ for a given calorimeter signal, as illustrated in fig. 9. This allows subtraction of the ADC pedestals, small corrections in calibration...
between channels, and application of a noise or pile-up threshold. But most important, the incoming energy signal can be converted to transverse energy, which is what we need for most triggers, or to the x and y components of transverse energy needed for the missing transverse energy trigger.

**Custom chip (ASIC)**

We have designed and built a custom integrated circuit (ASIC, or application-specific integrated circuit) to carry out the electromagnetic cluster-finding algorithm shown in fig. 4 at speeds up to 67 MHz (the original LHC specification) using 8-bit input signals. Each chip does the job for one reference cell, but to do that it must look at the 8-bit data from 16 trigger cells. Therefore, each trigger cell must be fed to many ASICs, resulting in a huge interconnection problem. In this prototype chip the hadronic calorimeter data are not processed, so the isolation is done only on the 12-cell e.m. ‘ring’. The cluster-finding logic is duplicated, with two pairs of programmable cluster and isolation thresholds. The energy sum of all 16 input cells is then formed, for use in future jet and missing-\(E_T\) logic.

The ASIC is a 0.8 \(\mu\)m CMOS gate array from Fujitsu, packaged as a 179-pin ceramic pin-grid array. The algorithm is implemented as a sequence of pipelined arithmetic stages, with the 12-bit total energy sum emerging after a latency of 6 clock cycles, and the cluster-found flags after 7 clock cycles. Tests at frequencies up to 70 MHz have been successful.

A few words on our experience with ASICs may be useful. ASICs allow the use of complex functions, customised exactly as you want, in a single integrated circuit. This can produce very compact designs, packing what might have needed an entire printed-circuit board onto one chip. This results in systems that are smaller and less complex, with fewer boards, crates and interconnections, and this should be more reliable. However, you do not always save money! ASIC technology is most cost-effective when large numbers of chips of each design are needed (e.g. in consumer electronics). The cost of producing an ASIC for small production runs is dominated by non-recurrent engineering charges (NRE), which are typically of the order of \(10^4 - 10^5\) SF. If there is a design error, however tiny (!), another prototype run costs another NRE charge, so the total cost of getting a working ASIC can have a large measure of unpredictability. This means that design must be done very carefully and simulated very thoroughly. Despite this, trivial mistakes still happen — it's a bit like reading a document on a screen, but only seeing the obvious mistakes when it has been printed out.

**Cluster-finding module**

In order to learn about the real-life performance of a trigger system using the ASICs, we built a small demonstrator system using a large (9U [40 cm] \(\times\) 40 cm) printed-circuit cluster-finding module (CFM) holding nine of the ASICs. This can fully process a 3 \(\times\) 3 area of calorimeter reference cells, but to do so a total of 36 trigger cells must be examined. A block diagram in shown in fig. 10. This module also has RAMs at the input, to allow readout of the incoming ADC information, and FIFOs at the output, to allow readout of the results from the ASICs. The RAM at the input can also inject stored data patterns, for testing.

The module needs a lot of I/O pins! A 96-way connector at the rear connects to a data bus, used for loading and reading registers (e.g. control, thresholds), while a massive 400-pin Teradyne connector (320 active pins plus 80 grounds) is used mainly for the 288 trigger-cell input bits (in high-speed ECL) as well as clock connections.
Prototype calorimeter trigger processor system

The prototype system is shown schematically in fig. 11, as it was used in beam tests with RD3 (see below). After adding both laterally and in depth to produce a suitable granularity, calorimeter signals are digitised in three 12-channel 8-bit flash-ADC (FADC) modules and then passed to the cluster-finding module containing the nine ASICs. In this first-prototype system we have not used RAM lookup tables to subtract pedestals, etc.

Figure 11. Trigger demonstrator system, as used with RD3.

The cluster-finding module and FADCs are housed in a trigger crate, controlled and read out by an interface module in a separate VME crate. A module in the trigger crate distributes system clock signals to the FADCs and the CFM. On each clock cycle, digital data from the 36 FADC channels are transferred in parallel to the CFM and injected into a pipeline running at the system clock frequency. Trigger bits resulting from the trigger cluster-finding algorithm emerge seven clock cycles later from the ASICs and are available as prompt front-panel outputs.

Copies of the incoming FADC information, output trigger hits and ASIC energy sums are saved in the CFM’s high-speed memories, allowing up to 256 time slices to be recorded. During operation the memories scroll continuously at the system clock frequency, so that at
any instant the memories contain a history of the preceding 256 FADC samples and the corresponding trigger algorithm results from the nine ASICs.

The trigger thresholds are set up by an Apple Macintosh via the VME crate. This machine is also used to run various test programs on the trigger and clock module hardware.

**Beam tests with RD3 and RD33**

In November 1992, April–May 1993, and November 1993 we recorded data in a CERN SPS test beam with the RD3 prototype liquid-argon Accordion calorimeter. This particular technology has now been selected for the electromagnetic calorimeter in ATLAS. A system-clock frequency of 40 MHz, corresponding to the 25 ns LHC bunch-crossing period, was used for most data collection although some data were also taken at the original design frequency of 67 MHz. The phase of the event trigger with respect to the asynchronous system clock was recorded using a CAMAC time-to-digital converter. We recorded data with different combinations of beam energy, position and particle type, and several cluster and isolation thresholds in the trigger. Beam positions near the centres of trigger cells and near the edges and corners were used.

On receipt of a beam trigger signal, the clock module counts a further programmable number of system clock cycles before generating a stop signal to freeze the CFM memories. The trigger signal is also sent to a VME-based data acquisition system (CERN Spider, using the OS9 operating system), which selectively reads out and records the memory contents of the CFM, and performs on-line analysis on a sample of the data. For much of the data-taking, we combined data with RD3 by linking our VME system with theirs and sending data to be recorded on their data cartridges.

We also recorded test beam data at CERN in June and November 1993 with the RD33 prototype TGT liquid argon calorimeter, the analysis of which is still in progress. These tests used a similar setup, except that RD33 supplied flash-ADC modules built for RD27, and the data acquisition used a Macintosh.

It used to be said that even in the distant future high-energy physicists will be using a language called Fortran (even if it is quite different from Fortran-77). Since this is a School of Computing, it is perhaps worth remarking that, at least in the online area, this is now untrue. Our DAQ and test software were all written in C, and people in the group are now experimenting with object programming and C++. I have the strong impression that similar things are happening in many other experiments.

**Analysis of test-beam data from the RD3 calorimeter**

I will give here only a brief survey of our results from RD3, with the goal of showing that the demonstrator system essentially works.

The Accordion calorimeter pulse is long compared to the bunch-crossing period, so the signal from a single shower is sampled over several clock cycles. Bipolar shaping is used. A typical pulse sampled at the LHC bunch-crossing frequency of 40 MHz is shown in fig. 12. Using the FADC digitisations prior to the arrival of the pulse, pedestal values and rms widths corresponded typically to 26 GeV and 0.6 GeV respectively, and were stable over periods of several days. The observation of low rms values is important because the noise levels seen in unoccupied channels have a considerable impact on the performance of the trigger, especially in the isolation veto. The FADCs were calibrated using electron beam data of several energies, and were cross-calibrated against the RD3 high-precision ADCs. The measured sensitivity is close to our design goal of 1 GeV/count.
**Level-1 Triggers for LHC Experiments**

![Graph showing FADC counts over time](image)

*Figure 12. Accordion calorimeter pulse sampled at 40 MHz.*

Before looking at the trigger logic, we can use the FADC input data to check how well our Monte Carlo simulations represent the situation. The trigger logic can be simulated easily in software. Threshold curves for 50 GeV electrons are plotted in fig. 13, using both the data and the Monte Carlo expectation. The top graph is for two-cell cluster windows and the bottom graph for one-cell windows. In both cases there is reasonable agreement between the data and the simulation, and as expected the two-cell window performs far better. Since the beam spot used was quite small (about the size of one calorimeter cell), the effect of hits at random positions in the trigger cell was simulated by mixing together runs at the centres, edges and corners of the cells.

![Graphs showing efficiency vs. threshold](image)

*Figure 13. Efficiency vs. threshold, for two-cell algorithm and single-cell algorithm.*

![Graph showing ASIC energy-sum output vs. expected value](image)

*Figure 14. ASIC energy-sum output vs. expected value.*

Recording both the FADC input data and the ASIC outputs enables us to compare the results of the ASIC-embedded algorithm with its software counterpart. Fig. 14 shows the correlation between the energy sum of all ASIC input channels and the prediction. The results are, in fact, perfect — any spread seen is simply due to the bin-widths used. Of course, this is just what we should expect from a digital system! In this graph and the results shown later, we have corrected for one small problem that was present in our early running with RD3: about 20% of the time the energy sum appeared one bunch-crossing late in the ASIC’s memories. This problem turned out to be a subtle timing effect on the CFM caused by readout software and has now been fixed; it was the only real error we encountered.

In order to test the two types of thresholding operations of the e.m. cluster-finding algorithm separately, the isolation requirement was disabled in many data-taking runs. Fig. 15 shows the effect of the cluster threshold, set to 60 counts, just above the pedestal sum for two channels. All events that should fire the trigger do so, and all events that should not don’t.
Fig. 16 shows the effect of the isolation threshold, set close to the expected 12-channel pedestal sum. Choosing events in which one of the cluster windows should be above the cluster threshold, one can compare the isolation sums in events for which the trigger hit was or was not present. Once again the results are perfect: every event with energy in the isolation region above threshold is vetoed, but when the energy in the isolation region is below threshold the event is never vetoed.

**Bunch-crossing identification**

As we saw in fig. 12, the calorimeter pulse is several bunch-crossings wide. For a pipelined level-1 trigger processor only the sampling corresponding to the peak of the pulse should be used by the cluster-finding algorithm for a particular crossing. All other samplings must be set to zero or they will contribute to triggers associated with earlier or later crossings.

We are therefore exploring techniques for bunch-crossing identification (BCID), using digital signal processing algorithms which we have simulated in software and exercised on our test-beam data. Algorithms studied include a peak-finding algorithm, a zero-crossing identifier, a constant-fraction discriminator and a deconvolution filter. Very encouraging results have already been obtained, down to quite small deposited energies. Even simple peak-finding (i.e. look for a bin higher than its two neighbours) identified the correct bunch-crossing 99% of the time for as little as 4 GeV energy deposited in one trigger cell.

A module using logic based on RAM lookup tables to allow the use of different algorithms is currently being tested. In the final ATLAS trigger system the selected BCID algorithm, implemented in hardware (programmable logic), would form part of the ADC system.

**Next steps with the calorimeter trigger**

The ~8000-channel calorimeter trigger has a formidable data-processing task — it's a parallel system that has to handle 8000 bytes × 40 MHz = 320 Gbytes/s. With nine ASICs, the current CFM requires over 320 I/O connections (plus control and power) to fully process a 3 × 3 area of e.m. trigger cells, and adding the hadronic calorimeter will make things even more difficult. As stated earlier, to cover all of ATLAS will require ~4000 e.m. trigger cells and possibly ~4000 hadronic cells. To do this we would need ~450 CFMs. With FADCs and the rest of the logic, this might result in a system with ~50 crates. This is huge! Techniques must be found to reduce its size by processing more trigger channels on each module.

Simply increasing the number of trigger channels processed per ASIC soon leads to I/O bottlenecks both at the ASIC and module level. For example, an ASIC built to process a 4 × 4
trigger cell array from both the ATLAS e.m. and hadronic calorimeters would require over 800 I/O pins alone. And a CFM containing only four such ASICs would need ~2000 I/O backplane connections, which is clearly not feasible.

We believe we have found an attractive solution, based on data sparsification and high-speed serialisation techniques. The major change is to transmit each 8-bit trigger channel in serial mode instead of parallel. Going at eight times the 40 MHz bunch-crossing rate seems very difficult, so we propose instead to transmit at 160 MHz. This is only possible if the data are sparsified using zero suppression, i.e. not sending signals unless there is data above pedestal. However, the data streams from different channels would then be asynchronous, so the signals must be tagged with a bunch-crossing number before being sent, and resynchronised using tag-matching on arrival inside the cluster-finding ASICs.

There are several key areas that require further study before a real processor can be designed in detail and constructed. We will therefore build a phase-2 trigger demonstrator system to enable us to resolve these issues. Our goal will be to show that a full-scale trigger system could then be built for ATLAS, even using present-day technology. The main aims are:

- To test our data sparsification scheme.
- To verify that bunch-crossing tagging and matching can be reliably accomplished.
- To include logic to evaluate various bunch-crossing identification algorithms.
- To assess the feasibility of using high-speed optical links between the front-end digitisation electronics and the level-1 processors.

Second-generation cluster-finding ASIC

To attack these issues we are designing a new high-speed multi-function ASIC (fig. 17), whose purpose will be to operate explicitly as a test bench. Our e.m. clustering algorithm has been adequately proven by the existing demonstrator system, and we believe that increasing the ASIC channel density to reduce the overall size of the system is a relatively straightforward task, assuming that the data sparsification scheme removes the I/O problem. Thus, in order to minimise the cost, the new ASIC will still process only one trigger channel.

The critical functions that this ASIC will test will be located in the following sections:

- **TRANSMIT block:** receive parallel data, perform zero suppression, tag, perform parallel-to-serial conversion and transmit serial data. (This would be part of the ADC system in ATLAS.)
- **RECEIVE block:** receive serial data, perform serial-to-parallel conversion, buffer (using FIFOs) and tag-match.
- **ALGORITHM block:** accept parallel data from the RECEIVE blocks and perform cluster processing for a $4 \times 4$ area, as in the current prototype system.

The ASIC will require about 63 inputs and 51 outputs, but by using bi-directional pins and sharing pins the I/O pin count could be considerably reduced. We will implement the ASIC in a 0.7 μm, 30 mm$^2$ CMOS cell-based array.

Second-generation demonstrator system

These ASICs will be used in a new demonstrator system, which will of course be tested on calorimeter data in a test-beam. We will keep the cost down by utilising some of the existing infrastructure. Existing FADCs may also be re-used. The system will use four new module and interface designs, as shown in fig. 18.
Level-1 Triggers for LHC Experiments

Each 12-channel transmitter module will pre-process the incoming FADC data to identify the relevant bunch-crossing, using a BCID algorithm running in field-programmable gate arrays. Look-up tables for pedestal subtraction will be implemented here. The resulting data will then transfer into three ASICs configured in their Transmit (T) mode, which will perform data sparsification and serialisation. The serial data from each transmitter module will then pass into an optical transmitter interface, which will inject them into 50 m optical fibre links, each running at 160 Mbit/s. These 36 serial links will feed the optical receiver interface, which will convert them back to electrical signals.

Figure 17. Phase-2 demonstrator ASIC.

Using nine ASICs configured in their Receive-Algorithm (RA) mode, the cluster processing module will accept the 36 asynchronous channels of serial data at 160 Mbit/s and rebuild the synchronous parallel data required by the cluster-finding algorithm. As in the current system, there would be input and output memories on all channels for data capture.

A possible ATLAS level-1 calorimeter trigger

To end the calorimeter trigger discussion, I will briefly show some ideas, sketched in fig. 19, for a working system. It has as its core a 0.5 μm CMOS ASIC that fully processes 4 × 4
trigger cells. This ASIC would receive 98 serial bit-streams of asynchronous zero-suppressed data from a $7 \times 7$ area of e.m. and hadronic calorimeters, and would provide eight sets of programmable threshold values. It would produce hit flags and transverse-energy sums, region-of-interest information for the level-2 trigger, and include pipeline memories for input data and results. Four such ASICs might be mounted on a cluster processing module.

Figure 18. Phase-2 demonstrator trigger processor system.
The FADCs and e.m. cluster processing might occupy four double-depth back-to-back crates, with jet cluster-finding and counting in a fifth crate, and missing-$E_T$ logic and e.m. cluster counting in a sixth crate. In addition to the e.m. cluster-finding ASIC, there might be ASICs for adding energy, finding jets in sliding $2 \times 2$ windows, and counting clusters by finding 'corners'. Thus the full level-1 calorimeter trigger processor for ATLAS might need as few as six crates.

3.2 Calorimeter trigger (bit-serial approach)

The calorimeter trigger work described so far has taken a relatively conventional approach, and has tried to show that the job could be done using present-day technology. At the same time, other members of RD27 have been pursuing a more radical option that may be adopted as an alternative, or the best of the two approaches might be combined.

The basic idea is to do all the operations in the trigger, such as addition and comparison with thresholds, in serial rather than in parallel. At first this seems a bit strange — if for example the operations were clocked at 160 MHz, i.e. 4 bits per LHC bunch-crossing period, then it is obvious that one trigger processor cannot keep up with the data. Instead, several trigger processors work in parallel, much like the topology of the high-level processor 'farms' already used for final-level triggering and event reconstruction in many present-day experiments. This is illustrated schematically in fig. 20. (Of course, if the speed is higher the need for parallel processors disappears. This may be possible.)

The logic in each processor is much simpler than in the bit-parallel processor, since it only operates on one bit at a time, and this should allow it to work much faster. Even taking account of the multiple processors needed, the overall system is claimed to have a lower total gate count. The complications, however, are the needs for processor switching logic and for the synchronisation of a bit-wise pipeline.

In order to gain experience and allow a fair evaluation of this option, a demonstrator system has been proposed and construction has started.
3.3 Muon triggers

The muon detector technologies for ATLAS and CMS are not yet decided. It seems probable that separate devices will be used for triggering (which requires high speed for bunch-crossing identification) and offline measurement — an attractive option for the trigger is resistive plate chambers (RPCs). These might come in strips a few cm wide by 1–2 m long.

The trigger would use hits in one trigger detector plane as a guide to where to look in a second plane (fig. 7 shows the concept). The width of the search region defines the momentum threshold. The muon trigger can be done locally, in perhaps ~1000 regions each corresponding to a level-2 trigger ‘region of interest’.

One way to do the ‘road-finding’ is to use a programmable coincidence matrix, illustrated in fig. 21. Each x,y cell can be set to give an output if both inputs are satisfied, or it can be switched off. This method has been used in experiment WA92, using RPCs and commercial programmable logic with a delay time of 96 ns. It is also used in the H1 forward muon trigger, which uses drift chambers and a $32 \times 32$ programmable matrix ASIC with a delay time of 48 ns. The H1 ASIC approach would be usable if a bigger and faster ASIC were developed. A $32 \times 64$ matrix with four thresholds and running at LHC clock speed has been looked at, and will be built soon (the NRE cost of the chip is considerable).

In the meantime, and in order to gain experience of RPCs, a simpler trigger demonstrator has been built and is being tested in conjunction with RD5. This uses a limited matrix based on four gallium arsenide crosspoint switches. Each switch can make 32 programmable connections between inputs and outputs. Using four switches, each strip in the first plane can be mapped to four strips in the second plane. A diagram is shown in fig. 22. The problem with this system is that using discrete switches for the final system is ruled out because there would have to be too much logic and the cost would be far too high.
3.4 Central trigger processor

The central trigger processor (see fig. 3) must collect results from all the different subtrigger processors, equalise their timing (both bunch-crossing number and phase), and then do combinatorial logic. This is because we will need compound triggers such as $ee$, $\mu\mu$, $e\mu$, jet*ET$_{miss}$, etc. In addition, to handle very high-rate triggers (e.g. for monitoring) some sort of scale-down facility will be very useful. And this is the place to monitor trigger rates.

The central trigger processor sends out crucial control signals to the front-end electronics on the detectors, and to the level-2 trigger system. Conceptually-similar devices are already used, for example in H1, but of course at LHC it must work very much faster.

The logic could be done using commercially-available chips, and RAM lookup tables for the combinatorial logic. However for reliability, speed and compactness three ASICs are being designed: a prescaler to scale down the input rates, a scaler to monitor rates, and a variable-length pipeline to adjust the bunch-crossing number of input signals to allow synchronisation. A schematic diagram showing the principles of the system is given in fig. 23.
3.5 Timing and synchronisation

I have already mentioned that synchronising control signals and collecting data from the detectors could easily become a nightmare, since the bunch-crossing time is of the same order as the time delays across detectors and the transmission time down cables or fibres. This is already a serious problem at HERA, where the machine period is 96 ns.

The problem begins with the need to distribute the master clock signal, with sub-nanosecond stability, to more than 10 million detector channels. Work has therefore begun on a system using laser transmitters and an optical-fibre distribution network. A big worry here is when to adopt a specific technology, since this area is developing very rapidly. At present a number of options are being studied in conjunction with RD12: a single high-power laser vs. many low-power ones; monomode vs. multimode optical fibres; choice of wavelength — 830 nm vs. 1300 nm; different kinds of couplers, detectors, etc.; development of a timing-receiver ASIC. Much progress has already been made and there is a much fuller discussion the RD27 1993 Status Report.

4 Conclusions

Although level-1 triggering at LHC will be extremely demanding, progress is being made on the main problems. A satisfactory calorimeter algorithm for electron and photon triggering has been both simulated and tested in beams. It seems possible to build a trigger based on this algorithm using even today's hardware, but more work is under way to study methods for reducing the size and interconnection problems of this trigger. Work on muon triggers is also progressing well, with some new test beam results. Central trigger processor logic is being
designed, and a concerted attack is being mounted on the problems of timing distribution and synchronisation.

In order to build a team to produce a real system for LHC more people will be needed, so if you find this unique area on the hardware/software frontier interesting please contact us!

Acknowledgements

The work described was done as part of RD27, and I would like to thank all members of the collaboration. I am particularly grateful to Nick Ellis, Alan Watson, John Garvey, Richard Carney, Tony Gillman, Norman Gee, and Viraj Perera for their help and encouragement in preparing these lectures. I would also like to thank the organisers of the School, notably Rinus Verkerk, Ingrid Barnett and Tony Hey, who provided an excellent atmosphere in an attractive, hospitable venue.

Suggested reading

Since these lectures were pedagogic, I have chosen to give a relatively short list of readable general references rather than specific but hard-to-find citations throughout the text. For those who want the full technical detail, there are now 19 RD27 notes available.


Visualization of Scientific Data for High Energy Physics.
PAW++, KUIP, PIAF: General-Purpose Portable Software Tools for Data Analysis and Presentation.

René Brun, Olivier Couet, Nicole Crevel, Michel Goossens, Alfred Nathaniel,
Fons Rademakers and Carlo E. Vandoni

CERN, CN Division, CH-1211 Geneva 23, Switzerland

Abstract
Visualization of scientific data, although a fashionable term in the world of computer graphics, is not a new invention, but is hundreds of years old. With the advent of computer graphics the visualization of scientific data has now become a well understood and widely used technology, with hundreds of applications in diverse fields. Over the last twenty years, CERN has played a leading role as the focus for development of packages and software libraries to solve problems related to High Energy Physics (HEP), a function which is considered essential by the entire HEP community. This paper discusses one of the key developments, which took place in the area of the visualization of scientific data, namely PAW, which stands for Physics Analysis Workstation. It is essentially an interactive system which includes many different software tools, strongly oriented towards data analysis and presentation. Some of these tools have been available in different forms and with different human interfaces for several years but they have now been integrated within a coherent framework.

1 PAW, the basic system
1.1 A short history
Nowadays, personal workstations equipped with high-resolution bitmap displays, a speed of several tens of MIPS, with at least 32 Mbytes of main memory and 1 Gbyte of local disk space (e.g., DECStaion, HP-700, IBM RS6000, Sun Sparc and SGI workstations) are widely available at an affordable price for individual users. Anticipating this evolution, and to prepare for fully exploiting the functionality of these workstations, in 1986 the PAW project was launched at CERN. The first public release of the system was made at the beginning of 1988. Today PAW runs on most of the computer systems used in HEP (Mainframes, Workstations, PC’s) but its full functionality is best exploited on personal workstations. In addition to its powerful data analysis facility, particular emphasis has been put on the quality of the user interface and of the graphical presentation.

1.2 What is PAW?
PAW [1] is an interactive utility for visualizing experimental data on a computer graphics display. It may be run in batch mode if desired for very large and time-consuming data analyses; typically, however, the user will decide on an analysis procedure interactively before running a batch job.

PAW combines a handful of CERN Program Library packages that may also be used individually in software that processes and displays data. The purpose of PAW is to provide a set of common analysis and display procedures, to supply a flexible way to invoke these procedures, and to allow user customization where necessary.

PAW's strong point is that it provides access to many facilities in the CERN library. One of its weaknesses is that these libraries were not designed from scratch to work together, so that a PAW user must eventually become somewhat familiar with many dissimilar subsystems in order to make effective use of some of PAW's more complex capabilities. As PAW evolves in the direction of more sophisticated interactive graphics interfaces and object-oriented interaction styles, the hope is that such limitations will gradually become less visible to the user.

PAW is most effective when it is run on a powerful workstation with substantial memory and rapid access to a large amount of disk storage. If the network traffic can be tolerated, PAW can be run remotely
over the network from a large, multiuser client machine to more economical servers such as an X-terminal. In case such facilities are unavailable, substantial effort has been made to ensure that PAW can be used also in non-interactive or batch mode from mainframes or minicomputers using character terminals.

1.3 Fundamental objects of PAW

PAW is implicitly based on a family of fundamental objects, as shown in Fig. 1.

![Diagram of PAW's fundamental objects](image)

**Figure 1: PAW's fundamental “data” objects**

Each PAW command performs an action that either produces another object or produces a “side-effect” such as a printed message or graphics display that is not saved anywhere as a data structure. Some commands do both, and some may or may not produce a PAW data structure depending on the settings of global PAW parameters. In this section, we describe the basic objects that the user needs to keep in mind when dealing with PAW. The Motif interactive graphics interface displays distinct icons for most of the object types listed below.

- **Ntuples.** An Ntuple is the basic type of data used in PAW. It consists of a list of identical data structures, one for each event. Typically, an Ntuple is made available to PAW by opening a ZEBRA/RZ file; this file, as created by HBOOK, contains one or more Ntuples and possibly also ZEBRA logical directories, which may store a hierarchy of Ntuples.
- **Chains.** A set of HBOOK files linked together, with each file containing Ntuples with an identical structure. This allows for logical Ntuples of arbitrary size. when creating histograms and plotting variables.

- **Cuts.** A cut is a function of Ntuple variables. Cuts are used to select subsets of events in an Ntuple when creating histograms and plotting variables.

- **Masks.** Masks are separate files that are logically identical to a set of boolean variables added on the end of an Ntuple's data structure. A mask is constructed using the Boolean result of applying a cut to an event set. A mask is useful only for efficiency; the effect of a mask is identical to that of the cut that produced it.

- **1D Histograms.** A histogram is the basic statistical analysis tool of PAW. Histograms are created ("booked") by choosing the basic characteristics of their bins, variables, and perhaps customized display parameters; numbers are entered into the histogram bins from an Ntuple (the histogram is "filled") by selecting the desired events, weights, and variable transformations to be used while counts are accumulated in the bins.

- **2D Histograms.** 2D (and higher-dimensional) histograms are logical generalizations of 1D histograms. 2D histograms, for example, are viewable as the result of counting the points in a the sections of a rectangular grid overlaid on a scatter plot of two variables. Higher-dimensional histograms can also be fitted.

- **Vectors.** PAW provides the facilities to store vectors of integer or real data. These vectors, or rather arrays with up to 3 index dimensions, can be manipulated with a set of dedicated commands. Furthermore they are interfaced to the array manipulation package SIGMA and to the Fortran interpreter COMIS. They provide a convenient and easy way to analyse small data sets stored in ASCII files.

- **Styles.** A "style" is a set of variables that control the appearance of PAW plots. Commands of the form `IGSET parameter value` determine fundamental characteristics of lines, axis format, text, and so on. Commands of the form `OPTION attribute` choose particular plotting options such as logarithmic/linear, bar-chart/scatter-plot, and statistics display. Commands of the form `SET parameter value` control a vast set of numerical format parameters used to control plotting. A "style" can be constructed by the user in the form of a macro file that resets all parameters back to their defaults and then sets the desired customizations. A style can also be defined via the style panel in the Motif Interface.

- **Metafile.** In normal interactive usage, images created on the screen correspond to no persistent data structure. If one wishes to create a savable graphics object, the user writes a metafile; as a graphics image is being drawn, each command is then saved in a text file in coded form that allows the image to be duplicated by other systems. PostScript format metafiles are especially useful because they can be directly printed on most printers; furthermore, the printed quality of graphics objects, such as fonts can be of much higher quality than the original screen image.

- **Pictures.** Metafiles describing very complex graphics objects can be extremely lengthy, and therefore inefficient in terms of storage and the time needed to redraw the image. A picture is an exact copy of the screen image, and so its storage and redisplay time are independent of complexity. Pictures are also intensively used for object picking in the Motif version of PAW.

- **ZEBRA(RZ) Logical Directories.** In a single PAW session, the user may work simultaneously with many Ntuples, histograms, and hierarchies of Ntuple and histograms. However, this is not accomplished using the native operating system's file handler. Instead, the user works with a set of objects that are similar to a file system, but are instead managed by the ZEBRA RZ package.
A single operating system file created by RZ can contain an entire hierarchy of ZEBRA logical directories; furthermore, sections of internal memory can also be organized as ZEBRA logical directories to receive newly-created PAW objects that are not written to files. A set of commands CDIR, LDIR, and MDIR are the basic utilities for walking through a set of ZEBRA logical directories of PAW objects. Each set of directories contained in an actual file corresponds to a logical unit number, and the root of the tree is usually of the form //LUNx; the PAW objects and logical directories stored in internal memory have the root //PAWC.

- **Operating System File Directories.** Many different ZEBRA files, some with logically equivalent Ntuples and histograms, can be arranged in the user's operating system file directories. Thus one must also keep clearly in mind the operating system file directories and their correspondence to the ZEBRA logical directories containing data that one wishes to work with. In many ways, the operating system file system is also a type of "object" that forms an essential part of the user's mental picture of the system.

- **Macros.** A macro is a set of command lines stored in a file, which can be created or modified with any text editor. In addition to all the PAW commands, special macro flow control statements are also available.

### 1.4 The component subsystems of PAW

The PAW system combines different tools and packages, which can also be used independently and some of which have already a long history behind them (e.g. HBOOK and HPLOT, SIGMA, COMIS, MINUIT). Figure 2 shows the various components of PAW.

#### 1.4.1 KUIP - The user interface package

The purpose of KUIP [2] (Kit for a User Interface Package) is to handle the dialogue between the user and the application program (PAW in our case). It parses the commands input into the system, verifies them for correctness and then hands over control to the relevant action routines.

Commands are grouped in a tree structure and they can be **abbreviated** to their shortest unambiguous form. If an ambiguous command is typed, then KUIP responds by showing all the possibilities. **Aliases** allow the user to abbreviate part or the whole of commonly used command and parameters. A sequence of PAW commands can be stored in a text file and, combined with flow control statements, form a powerful **macro** facility. With the help of **parameters**, whose values can be passed to the macros, general and adaptable task solving procedures can be developed.

#### 1.4.2 HBOOK and HPLOT - The histograming and plotting packages

HBOOK [3] and its graphics interface HPLOT [4] are libraries of Fortran 77 callable subroutines which have been in use for many years. They provide the following functionality:

- One- and two-dimensional histograms and Ntuples
- Projections and slices of two-dimensional histograms and Ntuples
- Complete control (input and output) of the histogram contents
- Operations and comparison of histograms
- Minimization and parameterization tools
- Random number generation
- Histograms and Ntuples structured in memory (directories)
- Histograms and Ntuples saved onto direct access ZEBRA files
- Wide range of graphics options:
  - Contour histograms, bar chart, shaded histograms, error bars, colour
  - Smoothed curves and surfaces
  - Scatter, lego, contour and surface plots
  - Automatic windowing
  - Graphics input
1.4.3 HIGZ - The graphics interface package

HIGZ [4], a High level Interface to Graphics and ZEBRA, is a layer between application programs (e.g., PAW/HPLOT) and basic graphics packages (e.g., X11). This package provides:

- Basic graphics functions: basic primitives, attributes, space definition.
- Higher-level macro-primitives (arcs, axes, boxes, pie-charts, Feynman diagrams, tables).
- Data structure management using an interface to the ZEBRA system.
- Full transportability of the picture data base.
- Easy manipulation of the picture elements (object picking, interactive picture editing).
- Compactness of the data to be transported and accessibility of the pictures in direct access mode.
- Independence of the underlying basic graphics package. Presently HIGZ is interfaced with several GKS packages, X-Window (X11), PHIGS, Mac, PC’s graphic systems, GL (Silicon Graphics), GIDDM (IBM), GPR (Apollo) as well as with the DI3000 system.
1.4.4 ZEBRA - The data structure management system

The data structure management package ZEBRA [5] was developed at CERN in order to overcome the lack of dynamic data structure facilities in Fortran 77. It implements the dynamic creation and modification of data structures at execution time and their transport to and from external media on the same or different computers, memory to memory, to disk or over the network, at an insignificant cost in terms of execution-time overheads.

ZEBRA manages any type of structure, but specifically supports linear structures (lists) and trees. ZEBRA input/output is either of a sequential or direct access type. Two data representations, native (no data conversion when transferred to/from the external medium) and exchange (a conversion to an interchange format is made), allow data to be transported between computers of the same and of different architectures. The direct access package RZ can be used to manage hierarchical data bases. In PAW this facility is exploited to store histograms, Ntuples and pictures in a hierarchical direct access directory structure.

1.4.5 MINUIT - Function minimization and error analysis

MINUIT [6] is a tool to find the minima of a multi-parameter function and analyse the shape around a minimum. It can be used for statistical analysis of curve fitting, working on a $\chi^2$ or log-likelihood function, to compute the best fit parameter values, their uncertainties and correlations. Guidance can be provided in order to find the correct solution, parameters can be kept fixed and data points can be easily added or removed from the fit. An interactive Motif based interface is in preparation.

1.4.6 COMIS - A Fortran 77 interpreter

The COMIS interpreter [7] allows the user to execute interactively a set of Fortran 77 routines in interpretive mode. The interpreter implements a large subset of the complete Fortran 77 language. It is an extremely important tool because it allows the user to specify his own complex data analysis procedures, for example selection criteria or a minimization function.

1.4.7 SIGMA - The array manipulation language

A scientific computing programming language SIGMA [1], which was designed essentially for mathematicians and theoretical physicists and has been in use at CERN for over 20 years, has been integrated into PAW. Its main characteristics are:

- The basic data units are scalars and one or more dimensional rectangular arrays, which are automatically handled.
- The computational operators resemble those of Fortran 90.

1.5 The functionality of PAW

PAW permits a wide variety of tasks relevant to analyzing and understanding physical data, which are typically statistical distributions of measured events. Below we list what are probably the most frequent and best-adapted applications of PAW; the list is not intended to be exhaustive.

1.5.1 Plot a vector of data fields for a list of events

A set of raw data is typically processed by the user’s own software to give a set of physical quantities, such as momenta, energies, particle identities, and so on, for each event. When this digested data is saved on a file as an Ntuple, it may be read and manipulated directly from PAW. Options for plotting Ntuples include the following:

- One variable. If a plot of one variable from the data set is requested, a histogram showing the statistical distribution of the values from all the events is automatically created. Individual events are not plotted, but appear only as a contribution to the corresponding histogram bin. A variety of features allow the user to slice and project a 2D scatter plot and make a 1D histogram from the resulting projection.
- **Two or three variables.** If a plot of two or three variables from the data set is requested, no histogram is created, but a 2D or 3D scatter plot showing a point or marker for each distinct event is produced. Various ways to show the resulting surface are available.

- **Four variables.** If a plot of four variables is requested, a 3D scatter plot of the first three variables is produced, and a colour map is assigned to the fourth variable; the displayed colour of the individual data points in the 3D scatter plot indicates the approximate value of the fourth variable.

- **More than four variables.** More than four variables can be plotted but it is up to the user to customize the system in order to assign the additional variables to graphics attributes like the size or the shape (type) of the markers.

- **Vector functions of variables.** PAW allows the user to define arbitrary vector functions of the original variables in an Ntuple, and to plot those instead of the bare variables. Thus one can easily plot something like $\sqrt{P_x^2 + P_y^2}$ if $P_x$ and $P_y$ are original variables in the data without having to add a new data field to the Ntuple at the time of its creation.

- **Event weights.** PAW allows the user to include a multiplicative statistical bias for each event which is a scalar function of the available variables. This permits the user to correct for known statistical biases in the data when making histograms of event distributions.

- **Selection functions (cuts).** PAW does not require the user of every event in the data set. Several methods are provided to define functions of the variables themselves that pick out subsets of the events to be included in a plot.

- **Histogram presentation options.** Virtually every aspect of the appearance of a histogram can be controlled by the user. Axis labels, tick marks, titles, colours, fonts, and so on, are specified by a large family of options. A particular set of options may be thought of as a "style" for presenting the data in a histogram; "styles" a formal part of PAW to aid the user in making graphics that have a standard pleasing appearance.

### 1.5.2 Fit a function to a histogram

Once a histogram is defined, the user may fit the resulting shape with one of a family of standard functions, or with a custom-designed function. The parameters of the fit are returned in user-accessible form. Fitted functions of one variable may be attached to a 1D histogram and plotted with it.

The fitting process in PAW is normally carried out by the HBOOK and MINUIT libraries. To use this package effectively, users must typically supply data within reasonable numerical ranges and give reasonable initial conditions for the fit before passing the task to the automated procedure.

### 1.5.3 Annotate and print graphics

A typical objective of a PAW user is to examine, manipulate, and display the properties of a body of experimental data, and then to prepare a graph of the results for use in a report, presentation, or publication. PAW includes for convenience a family of graphics primitives and procedures that may be used to annotate and customize graphics for such purposes. In addition, any graphics display presented on the screen can be converted to a PostScript file for black-and-white or colour printing, or for direct inclusion in a manuscript.

### 1.6 A user’s view of PAW

In order to take advantage of PAW, the user must first have an understanding of its basic structure. Below we explain the fundamental ways in which PAW and the user interact.

#### 1.6.1 Initialization

PAW may be invoked in a variety of ways, depending on the user’s specific computer system; these are described in the following chapter. As PAW starts, it prompts the user to select an interaction mode (or non-interactive mode) and window size and type (if interactive). The available window sizes and
positions are specified in the user file "higz_windows.dat". User-specific initializations are specified in the file "pawlogon.kmacu".

1.6.2 Command mode interface

The most basic interface is the KUIP "command mode" interface. KUIP provides a basic syntax for commands that are parsed and passed on to the PAW application routines to perform specific tasks. Among the basic features of KUIP with which the user interacts are the following:

- **Command entry.** Any unique partially entered command is interpreted as a fully entered command. KUIP responds to an ambiguous command by listing the possible alternatives. Individual lines can be edited in place using control keystrokes, either with an emacs-like key-binding as used by ksh and some other Unix shells, or with a binding as in VMS command line editing.

- **Parameters.** Parameters are entered after the basic command on the same line and are separated by spaces. If a parameter has embedded blanks, it must be specified between quotes. An exclamation point (!) can be used as a placeholder to keep the default value for a parameter if only a later parameter is being changed. If an underscore (_) is the last character on a line, the command may be continued on the next line; no spaces are allowed in the middle of continued parameter fields.

- **On-Line Assistance.** The "usage" and "help" commands can be used to get a short or verbose description of parameters and features of any command.

- **Command History.** The command history is kept in memory during an interactive session and is saved in a disk file at the end of the session. The command history file can be recovered and used to reconstruct a set of actions carried out interactively.

- **Aliases.** Allow the abbreviation of partial or complete command sequences.

1.6.3 KUIP/Motif interface

If the user's workstation supports the OSF/Motif windowing system, PAW can be started in the KUIP/Motif mode: the executable module to be run in that case is called PAW++. However, a small text panel and a command history panel keep track of individual actions, and permit entry and recall of typed commands similar to the command mode interface. The basic features of this interface are:

- **Pull-Down Menu "Commands".** Each PAW command (that can be given as input) has a corresponding item in a hierarchical pull-down menu (entry "Commands"). Commands that require arguments cause a parameter-entry dialog box to appear; when the arguments are entered and command execution requested (button "OK" or "Execute"), the command is executed as though typed from the command mode interface.

- **Action Panel(s).** A user may have a family of frequently executed macros or commands assigned to specific buttons on the action panel(s). These panels are totally user definable.

- **Object Browser.** All the objects known in PAW (Histograms, Ntuples, Vectors, etc...) can be manipulated via icons and pull-down menus in the "Object Browser". This is in many ways similar to the well-known browsers in the PC/MAC utilities or the visual tools on some workstations.

- **Direct Graphics Interaction.** One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object.

1.6.4 Graphics output window

The graphics image produced by PAW commands, regardless of the command interface, appears on a separate graphics output window. The actual size and position of this window on the screen is controlled by a list of numbers of the form x-upper-left y-upper-left x-width y-height in the user file higz_windows.dat. The width and height of the drawing area within this window are subject to additional user control, and the user can specify "zones," which are essentially ways of dividing the window into panes to allow simultaneous display of more than one plot. Some facilities are available in the current version of PAW to use the mouse to retrieve data such as the height of a histogram bin.
1.7 PAW development environment

The development of PAW involved the integration of a number of packages written by different people at different times and in different places. Therefore, several key goals had to be met and number of criteria and rules observed. Below we comment on the more points we took into consideration.

1.7.1 Portability

Computers used at CERN and in the collaborating Laboratories are anything but homogeneous. In practice, it is neither possible nor desirable to impose standards for the hardware platforms. Moreover, an analysis and/or data presentation package must run equally well in batch processing on the mainframe, interactively, via dumb terminals, in time-sharing on the mainframe, and interactively on the workstations. Therefore portability is the key to a successful implementation of software packages. There are several aspects of portability that should be taken into account: portability of the source code, support of the code, interface with the operating system, interface with the local data structure and storage, etc. This general need led CERN quite a long time ago to develop also portable software tools, e.g., source code management systems, data structures management systems, mathematical and utility libraries, etc., which are either completely portable or machine-independent.

1.7.2 Modularity and reusability

Often a package is not written by a single person, but is a collaboration between many different developers, working in Laboratories in CERN’s Member States, or beyond. Therefore, packages must have a modular structure, in order to permit an orderly and independent development of different modules by different authors. This raises problems of standard coding rules and practices, and on interfaces between the modules.

1.7.3 Backward compatibility

This is one of the most difficult goals to achieve. In fact, once a package has been released for production, there is the tendency, or even a systematic habit of many users to build around it their own specialized systems. The withdrawal of even minor features in an improved version of the system will inevitably provoke a strong negative reaction from such users. Notwithstanding that fact, the development of new, more modern, and powerful packages cannot and should not be stopped by making backward compatibility the main aim. Indeed, sometimes backward compatibility has to be sacrificed if it is incompatible with the desired improvements.

1.7.4 Design for maintenance

It is commonly recognized that maintenance of software products is often more expensive in terms of time and manpower than the actual development. It is always worthwhile to invest effort during the design phase in order to plan for facilities to ease the future maintenance of the system.

1.8 A few simple examples

The following pages present a few simple examples of the use of PAW.

- Figure 3 shows how to create and draw vectors.
- Figure 4 is about how to use vectors with COMIS and explains how to fit data points with error bars.
- Figure 5 contains pictures of one- and two-dimensional trigonometric functions.
- Figures 6 and 7 display various ways of presenting data, in one or more dimensions.
- Figures 8 and 9 are Feynman diagrams with fermion, gluon and photon lines or loops.
Figure 3: Vector operations, part 1
Vectors and COMIS

The declaration VECToR may be used inside a COMIS routine to address a KUIP vector. If the vector does not exist, it is created with the specifications provided by the declared dimension.

```
PAW > VECTOR/CREATE x(10) R 1 2 3 4 5 6 7 8 9 10
PAW > CALL VECT.F
PAW > VECTOR/WRITE x ! 10(1x,f3.0)
   1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
PAW > VECTOR/WRITE y ! 10(1x,f4.0)
   1. 4. 9. 16. 25. 36. 49. 64. 81. 100.
```

```
SUBROUTINE VECT
  VECToR X,Y(10)
  DO I=1,10
    Y(I) = X(I)*X(I)
  ENDDO
END
```

Fitting Vectors - Errors

```
PAW > APPLICATION SIGMA
SIGMA > alpha = array(24,0#2*PI)
SIGMA > sina = sin(alpha)+rndm(alpha)*0.3
SIGMA > errx = array(24,0.2#0.2)
SIGMA > erry = errx+rndm(errx)*0.1
SIGMA > EXIT
PAW > VECTOR/FIT alpha sina erry P3
PAW > VECTOR/FIT alpha(1:12) sina erry G S
PAW > VECTOR/CREATE par(1) r 10
PAW > VECTOR/FIT alpha sina erry SINFIT.F S 1 par
PAW > HPLoT/ERRORS alpha sina erry errx erry 24
```

```
function sinfit(x)
common /pawpar/ par(1)
sinfit=par(1)*sin(x)
end
```

```
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
<th>ERROR</th>
<th>SENS</th>
<th>DERIVATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1.0309</td>
<td>0.75578E-01</td>
<td>0.95588E-02</td>
<td>2.55594E-01</td>
</tr>
<tr>
<td>CHISQUARE</td>
<td>0.4981E+00</td>
<td>HPPR = 24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 4: Vector operations, part 2
Figure 5: Trigonometric functions in one and two dimensions.
Figure 6: Data presentations
Contour in linear and Log scale

Figure 7: Multi-dimensional data presentations

NTUPLE/PLOT with 3 variables
* c-c system
Arline 13 8 10 8 .3
Arline 10 10 13 10 .3
Arline 10 8 10 10 .3
* proton
Arline 4 5 8.5 5 .3
Arline 4 5.5 8.5 5.5 .3
Arline 4 6 8.5 6 .3
Line 8.5 6 13 4.5
Line 8.5 5 13 5
Line 8.5 5.5 13 5.5
* gluon
Helix 10 8 8.5 6 .3 7 30
* lepton
Arline 4 13 8 12 .3
Arline 8 12 13 13 .3
* photon
Helix 8 12 10 10 .1 4 0
* vertex
Fpoint 8 12 .1
Fpoint 10 10 .1
Fpoint 10 8 .1
Fpoint 8.5 6 .1
Itx 12.5 10.1 'c'
Line 12.55 8.45 12.7 8.45
Itx 12.5 8.1 'c'
Itx 12.5 13.1 'e^-1'
Itx 4.5 13.1 'e^-1'
Itx 4.5 6.2 'P'
Itx 9.3 11.1 '[g]'
Itx 9.5 6.8 'g'

* photon loop
arline 2 3 4 3 .2
archelix 4 3 10 3 .5 11 30 3.01
fpoint 4 3 .1
fpoint 10 3 .1
archelix 10 3 4 3 .5 11 30 3.01
arline 10 3 12 3 .2

* gluon loop
arline 2 11 4 11 .2
archelix 4 11 10 11 .15 6 0 3.01
fpoint 4 11 .1
fpoint 10 11 .1
archelix 10 11 4 11 .15 6 0 3.01
arline 10 11 12 11 .2

Figure 9: Feynman diagram with gluon and photon loops.

Figure 8: Feynman diagram with fermion, gluon and photon lines.
2 A detailed look at KUIP

KUIP represents a general-purpose User Interface Management System. The basis of KUIP is the so-called Command Definition File (CDF) containing a description of the command parameters, on-line help information, and how the commands are grouped into menus. Since menus can be linked to other menus the application commands are represented by an inverted tree in analogy to a Unix file system. The KUIP Compiler (KUIPC) converts the CDF into routines which have to be compiled and linked with the interactive application.

The interaction between the user and the application is either by typing command lines or selecting command from alphanumeric or graphical menus. In menu mode the user can traverse the command tree and select commands for execution.

2.1 Basic KUIP—Generalities

In this section we describe the main features of KUIP that are available for all the possible types of interaction (or dialogue styles) and especially the command line interface. This description will be made firstly from the “user’s point of view” and then, with more details, from the “application programmer’s point of view”.

2.1.1 Command structure

KUIP has a command tree like the Unix file system and the general syntax of a command line is a command path optionally followed by an argument list.

```
<table>
<thead>
<tr>
<th>Menus</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>directories</td>
<td>files</td>
</tr>
</tbody>
</table>
```

```
/KUIP/ALIAS/CREATE

menu submenu command

menu path
```

Abbreviations for command path /MENU/SUBMENU/COMMAND

- Command names are case-insensitive.
- Leading part of the menu path can be omitted.
- Trailing parts of menu and command names can be omitted, down to an unambiguous minimum, e.g., M/S/C SUB/COM COMM
- Rules to resolve ambiguities with other commands: /MENU/NEXTMENU/COMMA
  /MENU/COMMENT
  /OTHER_MENU/COMPUTE
  - Prefer exact matches, i.e., COMMA ⇒ /MENU/NEXTMENU/COMMA
  - Prefer lesser number of menu levels, i.e., COMM ⇒ /MENU/COMMENT
  - Prefer commands from SET/ROOT /OTHER menu, i.e., COM ⇒ /OTHER_MENU/COMPUTE

Command arguments

- Argument values are separated by blanks; strings containing blanks must be quoted; quotes must be duplicated:
  ```
  Mine > mycmd Hello 1 n
  Hello
  Mine > mycmd 'Hello World!' 1 n
  Hello World!
  Mine > mycmd 'How’re you?' 1 U
  HOW’RE YOU?
  ```
- Missing mandatory arguments are prompted for (CR re-uses value from previous command execution); CDF default values are used for missing optional arguments; exclamation mark is template for default value; double exclamation mark is template for last used value:
  ```
  Mine > mycmd
  Message text (CR=How’re you?)
  How’re you?
  ```
Mine > mycmd hello ! u
HELLO Mine >
mycmd hello 2 !
HELLO HELLO

- Arguments can be specified in a different order by using (abbreviated) parameter name=value; options can be shortened to -value:
Mine > mycmd count=2 l mess=HELLO
hello
hello
Mine > mycmd -l HELLO
hello

2.1.2 Macro interface language

A macro is a set of command lines stored in a file, which can be created and modified with any text editor. The command EXEC invokes the macro and allows for two ways of specifying the macro name:

PAW > EXEC abc | Execute first (or unnamed) macro of file abc.kumac
PAW > EXEC abc#m | Execute macro m of file abc.kumac

The first form executes the first macro contained in file while the second form selects the macro named macro. The default extension for file is ".kumac". In addition to all available KUIP commands the special "macro statements" in table 1 are valid only inside macros (except for EXEC and APPLICATION, which are valid both inside and outside).

Note that the statement keywords are fixed. Aliasing such as "ALIAS/CREATE jump GOTO" is not allowed.

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO mname var1=val1 ...</td>
<td>begin macro mname</td>
</tr>
<tr>
<td>EXEC mname var1 var2=val2 ...</td>
<td>execute macro mname</td>
</tr>
<tr>
<td>RETURN</td>
<td>end of a macro</td>
</tr>
<tr>
<td>READ var</td>
<td>read variable value from keyboard</td>
</tr>
<tr>
<td>SHIFT</td>
<td>control parameters list</td>
</tr>
<tr>
<td>label:</td>
<td>label (must terminate with a colon)</td>
</tr>
<tr>
<td>GOTO label</td>
<td>jump to label</td>
</tr>
<tr>
<td>ON ERROR GOTO label</td>
<td>resume at label on error condition</td>
</tr>
<tr>
<td>OF ERROR</td>
<td>temporarily deactivate the ON ERROR GOTO handling</td>
</tr>
<tr>
<td>ON ERROR</td>
<td>reactivate the latest ON ERROR GOTO handling</td>
</tr>
<tr>
<td>IF boolean_expr GOTO label</td>
<td>conditional statement</td>
</tr>
<tr>
<td>IF-THEN, ELSEIF, ELSE, ENDIFF</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>CASE, ENDCASE</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>WHILE-DO, ENDOWHILE</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>REPEAT, UNTIL</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>DO, ENDDO</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>FOR, ENDFOR</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>BREAKL</td>
<td>Macro flow control</td>
</tr>
<tr>
<td>EXitm</td>
<td>Macro termination</td>
</tr>
<tr>
<td>APPLICATION command marker</td>
<td>Inline text passed to application command</td>
</tr>
<tr>
<td>name = expression</td>
<td>Variable assignment</td>
</tr>
</tbody>
</table>

Table 1: List of statements possible inside KUIP macros
2.1.3 Additional features

- Built-in commands, e.g. /KUIP/EDIT to edit a file and /KUIP/SHELL to send a command to the operating system without leaving the application.
- Aliasing for commands and argument values:
  \texttt{Mine > alias/create hello 'mycmd Hello' -c}
  \texttt{Mine > hello -u}
  \texttt{HELLO}
  \texttt{Mine > alias/create greetings "'Hello World!'"}
  \texttt{Mine > mycmd greetings}
  \texttt{Hello World!}
- Command lines can be recalled, edited, and resubmitted. Typed-in commands are recorded in file \texttt{LAST.KUMAC}.

2.1.4 The Command Definition File (CDF)

From the application programmer's point of view the key to building a user interface with KUIP is to write the Command Definition File (CDF) which defines the available commands and their grouping into menus. It is a text file which has to be passed through the KUIP compiler (KUIPC). The output of KUIPC is either Fortran 77 or C source code\footnote{For the Motif interface (see section 2.2) the output of KUIPC must be a C source code procedure.} for a subroutine (we call it later the "definition routine") which the application main program has to call at initialization time in order to store the CDF information in memory.

In the Command Definition File (CDF) the programmer can specify:

- command names and their supporting menu;
- parameters: name, type (character, integer, real), prompt strings, default value, range, \ldots;
- guidance text for online help and manual in \LaTeX{} format;
- an action routine to perform the intended operation.

The action routine, which has to be provided by the application writer, can be coded in Fortran 77 (default) or in C. In the latter case the name of the routine has to be ended with "\%C" in the CDF declaration.

CDF example for a command /MYMENU/MYCMD

\begin{verbatim}
>Name MYDEF
>Menu MYMENU
>Guidance
This is the menu containing my own commands.

>Command MYCMD
>Action MYACT
>Parameters
MESS\%AGE 'Message text' C
  +
  COUNT 'Repetition count' I D=1 R=1:10
  CASE 'Case conversion' C Option D=N
    -N No conversion
    -U convert to Uppercase
    -L convert to Lowercase
    -C Capitalize (lowercase except first character in uppercase)

>Guidance
Print a message text.
Message text printed COUNT times with possible case conversion.
\end{verbatim}
Action routine for /MYMENU/MYCMD

SUBROUTINE MYACT
CHARACTER MSG*80, CASE*1
CALL KUGETS(MSG,LMSG) ! Get message text
IF(LMSG.EQ.0) LMSG=1
CALL KUGETI(NMIXES) ! Get repetition count
CALL KUGETC(CASE,LCASE) ! Get case conversion option

*----------- Apply case conversion
IF(CASE.EQ.'U') THEN
  CALL CLTUU(MSG(1:LMSG))
ELSEIF(CASE.EQ.'L') THEN
  CALL CUTUL(MSG(1:LMSG))
ELSEIF(CASE.EQ.'C') THEN
  CALL CLTUU(MSG(1:1))
  CALL CUTUL(MSG(2:LMSG))
ENDIF
DO 10 I=1,NTMIX
  PRINT *,MSG(1:LMSG)
10    CONTINUE
END

The action routine can retrieve the command arguments from the "KUGETx" routines (according to the type) and perform the appropriate actions. It can be coded in Fortran 77 or C.

2.1.5  Minimal main program for a KUIP application

PROGRAM MYAPPL
*---------- my own KUIP application
PARAMETER (NWPAW=20000)
PARAMETER (NWKUIP=5000)
COMMON/PWAC/PAN(NWPAW)
CALL MZEBRA(-3) ! Initialize ZEBRA
CALL MZPAW(NWPAW,' ') ! and PAW
CALL KUINIT(NWKUIP) ! Initialize KUIP
CALL MYDEF ! Definition of my own commands
           ! (from CDF translation)
CALL KUECC('SET/PROMPT 'Mine >''') ! This is my command line prompt
PRINT *,"*** Welcome to my application ***" ! And now pass control to KUIP
END

2.1.6  Building the KUIP application mine

- Translate CDF with KUIP compiler into C (or Fortran 77) source code:
  /cern/pro/bin/kuipc mydef.cdf mydef.c

- Compile source files and link with CERN libraries:
  cc -c mydef.c
  f77 -o mine myappl.f mydef.o myact.f -L/cern/pro/lib -lpacklib -lkernlib

- Run (command mode interface):

  $ mine
  *** Welcome to my application ***
  Mine > help mycmd

  * MYMENU/MYCMD MESSAGE [ COUNT CASE ]

  MESSAGE C 'Message text'
  COUNT   I 'Repetition count' D=1 R=1:10
  CASE    C 'Case conversion' D='N'
Possible CASE values are:

N No conversion
U convert to Uppercase
L convert to Lowercase
C Capitalize (lowercase except first character in uppercase)

Print a message text. The message text is printed COUNT times with possible case conversion.

Mine > mycmd hello 2 u
HELLO
HELLO
Mine > quit

2.2 The KUIP/Motif interface

A KUIP-based application can be adapted with minimal effort to the powerful graphical user interface provided by Motif, which is available on almost all workstations. By using the KUIP/Motif Interface one automatically gets access to:

- an executive window (or terminal emulator) for typing and editing commands and scrolling through the output (kxtterm);
- panels for entering command arguments and help texts generated from existing CDF specifications;
- user definable panels for executing frequently used command sequences;
- multiple instances of a browser for application definable objects (“browser interface”);
- optional HIGZ graphics window(s) with object identification;
- hooks for binding Motif code to buttons or pulldown menus (see PAW++).

It is supported on Unix and VMS workstations with X11R4 and Motif1.1 or higher.

In order to use these new features one does not have to learn programming with Motif.

The design goals of the KUIP/Motif interface are:

- Any application already using the basic KUIP command line interface can provide a graphical user interface with Motif “look-and-feel”. One does not have to get involved in Motif programming!
- The necessary changes are only in the initialization phase. Therefore the application needs only one library which can be loaded with different versions of the main program in order to produce a command line or a Motif interface, e.g., PAW and PAW++.
- Basic KUIP and KUIP/Motif co-exist in the same PACKLIB library. The command line interface does not have to load any Motif code.
- For applications which require high level graphics, users can have access to the X11 version of HIGZ (e.g., PAW++). Alternatively, if the application does not use graphics, KUIP/Motif does not load any HIGZ code.
- KUIP provides its own terminal window program kxtterm with Emacs-style command editing facilities. If kxtterm cannot be found in the search path, the terminal window (from which the application was started) will be used for input and output.
- An application can easily incorporate its own objects into the browser by adding class definitions to the CDF and providing simple routines for scanning the object lists.
- One can distribute the Motif version of the application as a single executable module (apart from kxtterm). Additional resource files are not mandatory.
2.2.1 The browser interface

The KUIP/Motif Browser is a key tool of this user interface: it allows the user to display and manipulate a tree structure of objects which are defined either by KUIP itself (commands, files, macros, etc.) or by the application (e.g. in PAW++: Zebra and HBOOK files, Chains, etc.). The objects contained in the currently selected directory can be displayed in various forms: big icons, small icons, text only, etc. It is possible to perform actions on these objects or the directories themselves by accessing popup menus directly attached to them: this behavior of the browser gives access to a “direct object manipulation” user interface by opposition to the usual “command mode interface”. Adding application specific objects into the browser is mainly done through the KUIP CDF.

Description of the “Main Browser” Window

For any application based on KUIP/Motif, one browser will be automatically created and displayed: it is called the “Main Browser”. Later on, it is possible to “clone” this browser (by pressing the corresponding button at the bottom/right) when it is in a certain state. This gives the user the possibility to have several instances of the browser window, and look at the same time to different kind of objects.

A “browser window” is composed of (Fig. 10):

- A two lines text/label area (⑥ et ⑦).
- The middle part of the browser is divided into two scrollable windows: the “FileList” or “Browsable window” ⑧ at the left and the “DirList” or “Object window” ⑨ at the right.
- Two lines of information at the bottom (⑥ et ⑦), plus a “Clone” ⑩ and a “Close” ⑪ buttons.

The middle (and main) part of the browser is divided into two scrollable windows on the left and right sides (Fig. 10):

The left hand “FileList” or “Browsable window” ⑧ shows the list of all the currently connected browsables. A “browsable” is simply a container of objects and is defined with the “>Browse” directive in the CDF. The browsables “Commands”, “Files” and “Macros” are built-in inside KUIP itself and are always displayed. Each application can add to this list its own definitions for any kind of browsables. Some browsables can also be attached at run time by selecting the corresponding “Open” entry in the menu “File”. Pressing the right mouse button in this window shows a popup menu with all the possible actions which have been defined for this browsable.

Selecting one item (or browsable) in this window with the left mouse button executes by default the “List” action (first entry of the popup menu): it displays the content of the browsable in the right hand window (“dirList” or “Object window”)

Note that the first entry of the popup menu of actions for one browsable is always “List” and that the last entry is always “Help” : it should give information concerning the selected browsable.

The right hand “DirList” or “Object window” ⑨ shows the content of the currently selected browsable for the selected path.

For instance, when one selects the browsable “Macro” (built-in inside KUIP), one gets all the KUIP macro files and sub-directories which are contained in the selected directory.

Objects are selected by clicking on them with the left mouse button. Pressing the right mouse button pops up a menu of possible operations depending on the object type ⑩.

An item in a popup menu is selected by pointing at the corresponding line and releasing the right mouse button. Double clicking with the left mouse button is equivalent to selecting the first menu item.

Each menu item executes a command sequence where the name of the selected object is filled into the appropriate place. By default the command is executed immediately whenever possible. (The commands executed can be seen by selecting “Echo Commands” in the “Options” menu of the “Executive Window.”)

In case some mandatory parameters are missing the corresponding “Command Argument Panel” is displayed, and the remaining arguments have to be filled in. The command is executed then by pressing the “OK” or “Execute” button. (Note that if it is not the last one in the sequence of commands bound to the menu item, the application is blocked until the “OK” or “Cancel” button is pressed.)
Figure 10: KUIP/Motif "Main Browser" Window

All application specific definitions for the entities accessible through the browser (objects, browsables and action menus) have to be made in the "Command Definition File" (CDF) with a simple syntax.

The two lines text/label area at the top displays information about (Fig. 10):

- the current path (or directory) for the selected browsable \( \mathbf{1} \) (entry "Path:"). The directory can be changed by pointing at the tail of the wanted sub-path and clicking the left mouse button. Clicking a second time on the same path segment performs the directory change and updates the "DirList" window with the list of objects.
- the number of objects of all different classes defined for selected browsable in current directory \( \mathbf{2} \).

The two lines of information at the bottom are filled with (Fig. 10):

- a short description of the browsable which is currently selected \( \mathbf{3} \) (entry "File:"),
- a short description of the object which is selected in the "object window" for a given browsable \( \mathbf{4} \).
Below follows a description of the different Browser menus:

**File**

This menu can be filled by the application with menu entries (buttons) which give access to the commands that can be used to connect or de-connect a new browsable at run time (e.g. in PAW++ the commands to open or close ZEBRA/RZ files).

These buttons/menu entries are automatically generated from the definition of the action menus for the browsables made in the CDF. The last entry of this menu is always "Exit", to exit from the application, e.g., the "File" menu in the PAW++ "Main Browser":

- **Exit**...  Exit from the application.

**Options**

- **Raise Window**  "cascade button" with the list of all opened windows. Selecting one of this window will popup the window on top of the others.
- **Command Argument Panel**  selecting this entry will prompt the user for a command name. If the command is valid then the corresponding "Command Argument Panel" with the list and description of all parameters will be displayed. If the command is ambiguous (e.g. command "list") the user will be proposed a list of all the possible commands. He can then select one and the corresponding "Command Argument Panel" will be displayed. If the command does not exist an error message is displayed.

**Commands**

This menu gives access to the complete tree of commands defined by KUIP and the application in the form of a pulldown menu. When a terminal item (command) in this menu is selected then the corresponding "Command Argument Panel" is displayed. The functionality of this menu is quite similar to the browsable "Commands" (this is just a matter of taste wether the user prefer to access commands through this pulldown menu or through the "Commands" browser).
Help

On Ktest
On Ktest Resources
On Kup Resources
On Browser
On Panel
On System Functions

Help specific to the application (has to be written in the CDF (Command Definition File)).

On {Appl.} Resources
Help specific to the application resources (has to be written in the CDF (Command Definition File)). Resources control the appearance and behavior of an application.

On Kup Resources
List the X resources available to any KUIP/Motif-based application.

On Browser
Help on the KUIP/Motif Browser interface ("Main Browser").

On Panel
Help on the KUIP/Motif "PANEL interface".

On System Functions
List KUIP all internal system functions currently available.

View

This menu allows to change the way objects are displayed or selected.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Small Icons</th>
<th>display objects with normal size icons and names (default).</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Icons</td>
<td>display objects with small icons and names.</td>
<td></td>
</tr>
<tr>
<td>Titles</td>
<td>display objects without icons, but names and small titles.</td>
<td></td>
</tr>
<tr>
<td>Select All</td>
<td>display objects without icons, but long titles.</td>
<td></td>
</tr>
<tr>
<td>Filter...</td>
<td>select all the objects.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ask for a filter to be put on object names.</td>
<td></td>
</tr>
</tbody>
</table>

2.2.2 kxterm: the KUIP terminal emulator (or "Executive Window")

This terminal emulator combines the best features from the (now defunct) Apollo DM pads (Input Pad and Transcript Pad, automatic file backup of Transcript Pad, string search in pads, etc.) and the Korn shell emacs-style command line editing and command line recall mechanism.

Description and behaviour

kxterm (or what we call the "Executive Window" in a KUIP-based application) is composed of three main parts (Fig. 11):

- A "menu bar" with the menu entries "File" 🎤, "Edit" 🎵, "View" 🎮, "Options" 🎤, and "Help" 🎤.
- A Transcript Pad 🎤 which contains any kind of output coming from KUIP or from the application.
- An Input Pad 🎤 which is an editable "scrolled widow" where the user can type commands.

Commands are typed in the input pad behind the application prompt. Via the toggle buttons 🎤 labeled "H" the Input Pad and/or Transcript Pad can be placed in hold mode. In hold mode one can paste or type a number of commands into the Input Pad and edit them without sending the commands to the application. Releasing the hold button will causes Kxterm to submit all lines, up to the line containing the cursor, to the application. To submit the lines below the cursor, just move the cursor down. In this way one can still edit the lines just before they are being submitted to the application.

Commands can be edited in the Input Pad using emacs-like key sequences. The Transcript Pad shows the executed commands and command output. When in hold mode the Transcript Pad does not scroll to make the new text visible.

Every time the current directory is changed, the Current working directory indicator 🎤 is updated. The current working directory is the one which is currently selected in the "Main Browser".
2.2.3 CDF directives specific to the Motif interface

This part is intended to application programmers who want to customize their application for the KUIP/Motif interface, and, in particular add their own classes of object into the browser interface.

Browser concepts and definitions

The browser allows to traverse a hierarchical directory structure of objects. Operations on these objects can be chosen from popup menus which depend on the object type (or "class").

To incorporate one’s own application specific objects into the browser one must:

- describe in the CDF the object types ("classes") and the containers for these objects ("browsable"),
- write "small" routines to scan through the list of objects, and eventually the list of browsables,
- describe in the CDF the action menus for classes of objects and browsables.
Classes of objects

"Classes" of objects can be any kind of entities handled by an application. For example:

- HIGZ pictures and HBOOK data (1d-histograms, 2d-histograms, N-tuples, chains and directories) are "Classes" defined in PAW++;
- in GEANT++ the classes are the data structures: volumes, materials, particles, ...;
- in KUIP/Motif the commands themselves have been defined as classes, as well as the different type of files (read, read-write, executable, and macros ...).

Object classes are defined in the CDF with the directive:

```
>Class class-name menu-title [ big-icon small-icon ]
```

Example: >Class 1d '1d histogram' big_1d small_1d

This specifies the class name ("1d"), a title text for menus, and optionally the icons used by the browser. The >Class directive can be followed by two sets of action menus. The first applies to objects selected inside the browser (see, e.g., section 2.3.3), while the second set is for objects identified inside a HIGZ graphics window. The two sets of action menus must be separated with a blank line starting with the character "+".

Browsables

All objects are part of container objects ("browsables") constituting the top level directory, e.g. ZEBRA/RZ files.Browsable classes are defined with the CDF directive

```
>Browse class-name menu-title scan-objects [ scan-browsables ]
```

Example: >Browse PAWC 'Histograms and Pictures in memory' PBHIST PBLUN

This specifies the browsable class-name ("PAWC"), a title text for menus, and the application routine to scan the objects contained in a directory (PBHIST).

Like the >Class directive, the >Browse directive can be followed by two sets of action menus. The first applies to the browsables themselves, e.g., how to create a new object or delete all objects in the browsable. There exist two kinds of browsable classes: single instance and multiple instance. For a multiple instance class the application has to provide a second scanning routine (PBLUN) from which the names of currently connected browsables of a given class can be retrieved. The second set of action menu allows one to specify the necessary actions to connect new browsables, e.g., how to open a file in which the objects are stored. As in the case of the definition of class objects described above, the two sets of menus must be separated by a blank line starting with the character "+".

If only one instance of a browsable class exists, for example the command tree, there is no need for a scan-browsables routine\(^2\) (and thus for the second set of action menus). Single instance browsables are always present in the browser.

Directories

A tree structure of objects can easily be achieved by defining a special class for subdirectories of a browsable. The corresponding CDF directive is the same as for simple object classes, ">Class", except that there must a "/" in front of the class name. KUIP/Motif imposes only the following conventions:

- The first item in the action menu for a directory always means: "change to this subdirectory".
- The new directory path is formed by concatenating the old path, a "/", and the name of the directory object.
- The routine used for scanning the objects in the top level directory must be able to return the objects in any subdirectory with a given path.

Going upwards in the directory hierarchy is done by selecting a substring of the current path displayed in the browser window (see Fig. 10).

\(^2\)This routine can, however, be used for returning the value of possible meta-variables used in the definition of the action-menu.
Action menus

Class specific menus for objects or browsables are derived from sequences of action definitions

\begin{multicols}{4}
\begin{itemize}
\item \texttt{menu-text [ special-flags ] [ action-string ] [ action-routine ]}
\end{itemize}
\end{multicols}

following the \texttt{>Class} or \texttt{>Browse} directive (see section 2.3.3 for an example). The menu pops up when pressing the right mouse button, and the selected action is performed when the button is released. A double click with the left mouse button executes the first menu item.

The first parameter \texttt{special-flags} is not used at present, and should be replaced by a dot (.), which acts as a placeholder. The action can be specified as a string of commands (\texttt{action-string}), or a routine (\texttt{action-routine}) to be called.

Inside the definition of \texttt{action-string} constructs of the form "\texttt{[var]}" can insert identifications of the selected object. Presently, the following “meta-variables” are recognized:

- \texttt{[this]} is replaced by the object name, e.g. histogram “10”.
- \texttt{[that]} is replaced by the short description text returned by the \texttt{scan-objects} routine and can thus be used as an alias name.
- \texttt{[name]} is replaced by the name of the browsable, e.g. “LUN21” for an ZEBRA/RZ file opened on unit 21.
- \texttt{[root]} is replaced by the path of the top level directory, e.g. “//LUN21”.
- \texttt{[path]} is replaced by the complete path to the directory in which the object is contained, e.g. “//LUN21/MYDIR”. Initially, \texttt{[path]} is set to \texttt{[root]}.

When the action is specified as a routine, the second parameter \texttt{action-string} should be replaced by a dot (.), acting as a placeholder. The action routine can be coded in C or Fortran 77. For C coding, the name of the action routine must be ended with “\texttt{KC}” in the CDF declaration. In Fortran 77 the pre-defined calling sequence depends on which entity the action menu applies. In C it is the “usual” syntax for a “callback” routine with Motif.

The replacement values for \texttt{[name]}, \texttt{[root]} and, if necessary, for additional variables have to be returned by the \texttt{scan-browsables} routine. For single instance browsables \texttt{[name]} is substituted by the class name and the field \texttt{menu-title} of the \texttt{>Browse} directive is used as the definition of \texttt{[root]}.

By default the commands in the action string are executed immediately, provided that all mandatory arguments are present. If any mandatory argument is missing the corresponding command panel comes up where they can be filled in. This default behaviour can be modified in several ways:

- Pressing the CTRL-key when popping up the menu forces the command panel even if all mandatory arguments are specified.
- Putting a “-” in front of the command name forces the command panel as well.
- Putting a “+” in front of the command name never produces a command panel, independently of the state of the CTRL-key.

Customization

- \textit{Interaction with HIGZ graphics}. KUIP/Motif can be interfaced to the X11 version of the HIGZ graphics package for an application which requires high level graphics. In order for the two packages to co-operate properly, applications using HIGZ have to add the directive \texttt{>Graphics} in the CDF and call \texttt{KUINIT} before calling \texttt{IGNIT}.

HIGZ allows one to store the name and the class for every displayed object by calling the routine \texttt{IGPID}. From this information KUIP/Motif can identify the object at the mouse pointer position and pop up the second set of action menus defined with the \texttt{>Class} directive. This mechanism is extensively used in the Motif versions of GEANT [8] (Fig. 12) and PAW (Figs. 13 and 14).
- **Buttons.** An application can create its own buttons and pulldown menus either in the Executive window (xterm) or the main browser window using the CDF directive

```
>Button menu-title button-label action-routine mnemonic accelerator accelerator-text [flag]
```

- **menu-title** is the title of the pulldown menu where the button has to be put. If it does not exist, a new pulldown menu with this title is created.
- **button-label** is the text label of the menu entry to be created.
- **action-routine** is the associated action automatically invoked for a “Button Press” event. It can be a string of commands or a routine coded in C or Fortran 77 (in C the name of the routine has to be ended with “%C” in the CDF declaration).
- **mnemonic** is a mnemonic definition for the label.
- **accelerator** and **accelerator-text** are accelerator definition for the label.
- if flag = “BR” menus/buttons are created in the main browser (instead of the executive window, by default).

- **Icons.** The application can define its own icons to represent objects in the browser. Each class definition allows one to specify the names of “big” and “small” icons which are looked up in the table of available icon bitmaps. The same icon is used for both sizes only one of them is defined. If none is defined then a default icon is used. The >Class directive refers to icons by name, e.g. “sm_myicon”.

To create a new icon bitmap one can use the X11 standard bitmap editor, e.g., to get a 20 × 20 pixel icon called “small_id” one can type: bitmap small_id.by 20x20. Then the output file small_id.by containing “#define small_id_width 20…” simply needs to be inserted into the CDF following the directive >Icon_bitmaps, for instance.

```
>Icon_bitmaps
#define sm_myicon_width 18
#define sm_myicon_height 22
static char sm_myicon_bits[] = {
  0xff, 0xff, 0x03, 0x01, 0x00, 0x02, 0x81, 0x07, ...
  0x21, 0x10, 0x02, 0x21, 0x10, 0x02, 0x21, 0x10, ...
  0x41, 0x08, 0x02, 0x81, 0x04, 0x02, 0x01, 0x03, ...
  0x81, 0x04, 0x02, 0x41, 0x08, 0x02, 0x41, 0x06, ...
  0x21, 0x10, 0x02, 0x21, 0x10, 0x02, 0x41, 0x08, ...
  0x01, 0x00, 0x02, 0xff, 0xff, 0x03}
#define big_myicon_width 36
...
```

KUIPC compiles the bitmaps into the application, i.e., there is no need for separate bitmap files at run-time. Alternatively bitmap files can be specified as resources in .Xdefaults overriding CDF bitmaps of the same name.

- **Hooks.** If one wants to get involved in Motif programming one is free to do so. KUIP/Motif provides books that an application can link its own Motif code in. The following CDF directive defines two routines fallback-proc and widget-proc, which are called at the initialization phase.

```
>Motif_customize fallback-proc widget-proc
```

The first routine has to return the application specific fallbacks while the second routine is called for every top level widget (Executive window, browser instance, graphics window) created by KUIP/Motif.
2.3 Example of KUIP/Motif application code

2.3.1 Skeletons for the scan-objects routine (next_object)

This routine can be coded in Fortran 77 or in C.

Fortran 77 coding

In the first call the object name is set to blank and then the routine is called by the browser repeatedly until the returned object name is blank.

```fortran
SUBROUTINE next_object(
  + browsable_name, browsable_class, current_path,
  + object_name, object_class, short_text, long_text
) CHARACTER(*) browsable_name, ..., long_text

SAVE next_counter

IF( object_name .EQ. ' ' ) THEN
*** first object is requested
  next_counter = 1

*** identify from browsable_name, browsable_class
*** and current_path which objects are requested

ELSE
*** next object is requested
  next_counter = next_counter + 1
ENDIF
*** find object with number next_counter

IF( no_more_objects_left ) THEN
*** stop scanning process
  object_name = ''
ELSE
*** mandatory return values
  object_name = ...
  object_class = ...
*** optional return values
  short_text = ...
  long_text = ...
ENDIF
END
```

C coding

In the first call the object position is set to 0 and then the routine is called repeatedly until the returned object description (char *obj_desc) is a NULL pointer.

```c
char **next_object( brobj_name, brcls_name, path, n )
char *bobj_name; /* browsable name (displayed in the browser) */
char *brcls_name; /* browsable class (from the "Browse" directive) */
char *path; /* current path for the directory */
int n; /* object position (0 the first time) */

static char *obj_desc[4];

... obj_desc[0] = NULL; /* name */
obj_desc[1] = NULL; /* class */
obj_desc[2] = NULL; /* alias */
obj_desc[3] = NULL; /* text */

if( n == 0 )
  /* Initialization */
...
obj_desc[0] = .../* object name or identifier */
obj_desc[1] = .../* class name */
obj_desc[2] = .../* short text description of the object */
obj_desc[3] = .../* long text description of the object */
return obj_desc;

2.3.2 Skeletons for the scan-browsables routine (next.browsable)

A scan-browsables routine must be supplied for browsable classes if more than one browsable of that type can exist, or if non-default meta-variables are needed.

Again, this routine can be coded in Fortran 77 or in C.

Fortran 77 coding

SUBROUTINE next_browsable(
   + browsable_name, browsable_class, meta_variables 
   CHARACTER(*) browsable_name, ..., meta_variables
   ...
   SAVE next_counter
   
   IF( browsable_name .EQ. ' ' ) THEN
      next_counter = 1
   ELSE
      next_counter = next_counter + 1
   ENDIF
   *---- find next_counter browsable of requested class
   ...
   IF( no_more_browsables_left ) THEN
      browsable_name = ' ' 
   ELSE
      browsable_name = ...
      meta_variables = ...
   *---- e.g. 'name=LUN1 root=//LUN1 unit=1 file=myfile.hbook'
   ENDIF
   END

C coding

char **kmbsdi( class_name, first )
   char *class_name;
   int first;

   static char *path_desc[2];
   ...
   path_desc[0] = NULL;
   path_desc[1] = NULL;

   if (first)
      /* Initialization */
      ...

   path_desc[0] = ...; /* browsable name */
   path_desc[1] = ...; /* meta-variable */
   /* e.g. : sprintf(path_desc[1], "root=Ns file=List of KUIP macros",..."); */
   return path_desc;

In both cases the "meta-variable" is a string filled with a set of keywords and their value for the corresponding browsable. All keywords have a predefined meaning (see the description of the action menus in section 2.2.3), e.g., [name] is the text for the class window, [root] is the start-up value of [path], and [file] is the text displayed at the bottom of the browser (see Fig. 10).
2.3.3 A KUIP/Motif toy example—Create a browsable “Alphabet” class

Defining the CDF

Let us define the browsable class “Alphabet” to contain the object classes “Letter” and “Digit”. The letter and digit objects should be contained in separate directories. The action menus should allow one to print the Latin letters or their Greek equivalent and to print the digits with Arabic or Roman numerals.

> Name ALFDEF
> Browse Alphabet ‘Letters and Digits’ NXALFA
> List

By convention selecting the first menu item or single-clicking the left button calls scan-objects to obtain the objects in the current directory.

> Class Letter ‘Roman and Greek letters’ sm_letter big_letter
Roman . ‘message [this]’
Greek . ‘message [that]’
Translate . ‘message Letter [this] is [that] in Greek’

> Class Digit ‘Arabic and Roman digits’ sm_digit big_digit
Roman . ‘message [that]’
Arabic . ‘message [that]’
Translate . ‘message Digit [this] is [that] in Roman’
Hello . ‘mycmd Hello [this]’
Hello... . ‘-mycmd Hello [this]’

> Class /LetDig ‘Letters and Digits’ sm_letdig big_letdig
> List

Directory objects are denoted by a “/” preceding the class name. The first menu item or double-clicking the left button makes it the current directory and calls scan-objects.

The scan-objects routine for “Alphabet”

```
SUBROUTINE NXALFA(BRNAME, BRCLAS, BRPATH, OBNAME, OBCLAS, STEXT, LTEXT)
  CHARACTER*(*) BRNAME, BRCLAS, BRPATH, OBNAME, OBCLAS, STEXT, LTEXT
  PARAMETER (MLET=5, MDIG=9, MLD=2)
  CHARACTER LLATIN(MLET)=9, LGREEK(MLET)=8,
  CHARACTER DROMAN(MDIG)=4, LETDIG(MLD)=8
  DATA LGREEK//’Alpha’,’Beta’,’Gamma’,’Delta’,’Epsilon’/
  DATA LLATIN//’A’,’B’,’C’,’D’,’E’/
  DATA DROMAN//’I’,’II’,’III’,’IV’,’V’,’VI’,’VII’,’VIII’,’IX’/
  DATA LETDIG//’Letters’,’Digits’/
  SAVE NEXT
  IF(OBNAME.EQ., ’’) THEN
    NEXT=1
  ELSE
    OBNAME=’’,
    NEXT=NEXT+1
  ENDIF
  IF(BRPATH.EQ., ’’) THEN
    IF(NEXT.LE.MLD) THEN
      OBNAME=LETDIG(NEXT)
      OBCLAS='LetDig'
      STEXT='Directory of Letters'
      LTEXT='Roman and Greek uppercase letters'
    ENDIF
    ELSEIF(BRPATH.EQ., ’/Letters’) THEN
      IF(NEXT.LE.MLET) THEN
        OBNAME=LLATIN(NEXT)
        OBCLAS='Letter'
```
Modifications to the main program

To use the Motif version of our example very few changes are necessary to the user’s main program (compare with the code shown in section 2.1.5).

```c
# ifdef MOTIF
    CALL KUWHAM('Mine++') ! Give control to KUIP (Motif)
#else
    CALL KUWHAT ! Give control to KUIP (non-Motif)
#endif
END
```

The character string given to KUINIM or KUWHAM is used for window titles and to lookup resources in .Xdefaults.

Building the KUIP/Motif application mine++

- Translate CDFs with KUIPC (the Motif specific information is only supported in C output mode):
  ```
  /cern/pro/bin/kuipc mydef.cdf mydef.c
  /cern/pro/bin/kuipc alpha.cdf alpha.c
  ```

- Extract KUIP/Motif version of the main program, e.g.
  ```
  /lib/cpp -P -DMOTIF myappl.src myappl++.f
  ```

- Compile source files and link with CERN and Motif libraries (paths are system dependent):
  ```
  cc -c mydef.c alpha.c
  f77 -o mine++ myappl++.f mydef.o myact.f alpha.o nxalfa.f
  -L/cern/pro/lib -lpacklib -lkernlib -L/usr/lib/X11(?) -lXm -lXt -lX11 -lFW(?)
  ```

- Run (using the Motif interface):
  ```
  mine++
  ```

2.4 GEANT++: a CERN application interfaced to KUIP/Motif

The GEANT program, written at CERN, is an example of an application interfaced to KUIP/Motif and Figure 12 will give a better idea of the general look and feel of a Motif graphical user interface available as a KUIP-based application.
Figure 12: Display showing various panels for GEANT++

- The upper left corner is the "Executive Window" (xterm) with the top menu bar and input/output pad.
- The "Command Panel" window on top of the executive window is generated from the CDF definition of the command /CONTROL/PLMAT. This panel allows the user to interactively execute the corresponding command arguments and to control the command execution. (The "OK" button is to execute the command and delete the panel, "Execute" is to execute the command and keep the panel, "Reset" to reset parameters values to default ones, "Cancel" to delete the panel without executing the command and "Help" to access the command help.)
- The lower left window shows the GEANT++ Main Browser. Different classes of objects (specific to GEANT) have been defined in the CDF and appear as a list in the very left part of the browser: volumes (VOLU), materials (MATE), particles (PART), ... The browsable class MATE (materials) has been selected and its content is displayed in the scrollable area at the right side. The CO2 object in that area is activated and a menu of possible actions is displayed (when pressing the right mouse button).
- The graphics window (interfaced to HIIGZ/X11) at the top right shows the results for all physical mechanisms in the material, selected via the command panel /CONTROL/PLMAT described above. By using the HIIGZ mechanism to structure the graphics, it is possible to pop up a menu of actions for the object identified at the mouse pointer position.
- At the lower right and under the graphics window two user defined panel of commands are displayed. The content of these two panels has been defined with the KUIP command "/KUIP/SET_SHOW/PANEL", which allows one to define command sequences which are executed when the corresponding button is pressed. Panels can also be edited/modified interactively and save into a macro file ("Save Panel1" button). A panel can be reloaded by pressing the "Panel1" button in the Executive Window and entering the corresponding macro file name.
3 PAW++

PAW++ [9] is a version of PAW which used the KUIP/Motif interface and is optimized for OSF/MOTIF. This makes the full and rich command set of PAW more easily accessible to the average user, since simple point and click operations are sufficient to execute commands that were previously only used by experts. Presently, it is released on Unix workstations and VAX/VMS.

PAW++ has, in addition to the conventional command line and macro types of interface, the following dialogue modes:

- **Pull Down menus**
  They are useful to understand the command structure of the PAW system.

- **Command panels**
  They give a “panel representation” of the commands.

- **Object Browser**
  This is in many ways similar to the well-known browsers in the PC/MAC utilities or the visual tools on some workstations.

- **Direct graphics**
  One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object. For example, by clicking with the right mouse button on a histogram, one can make directly a Gaussian fit, a smoothing etc. Pop-up menus are available by clicking on the graphics window to automatically produce PostScript, Encapsulated PostScript, \LaTeX\ files or print the picture on your local printer.

- **Histogram style panel**
  Buttons are available to change histogram attributes, colours, line styles, fonts, and axes representation. 2-D histograms can be rotated interactively. Zooming and rebinning can be performed interactively in real time.

- **Ntuple viewer**
  Just click on the Ntuple column name to histogram the column.

PAW++ starts up with three windows on the screen:

- **The “PAW++ Executive window”** includes a menu bar, a Transcript pad, a current working directory indicator and an Input pad.

- **The “PAW++ Graphics 1”** window displays the graphics output from HIGZ/X11. Objects, e.g. histograms, displayed in the graphics window can be manipulated by pointing at them, pressing the right mouse button and selecting an operation from the popup menu. Pointing at the edge of the graphics window (between displayed object and window border) brings up a general popup menu. Up to 4 additional graphics window can be opened by selecting “Open New Window” from this menu.

- **The “PAW++ main browser”** displays all browsable classes and connected HBOOK files. Up to 4 additional browsers can be opened via the “View” menu of the “PAW++ Executive window” or via the “Clone” button on the browsers. For more information on the browsers see the “Help” menus.

Figures 13 and 14 show the PAW++/Motif screen with the Executive window and the one of the Ntuple viewer.

There also exists a utility in ZEBRA (DZDOC), which allows one to view structural elements, like divisions, banks, etc. Figure 15 is an example of the Motif interface for ZEBRA, showing in the lower right hand corner the description of the data structure.
- The upper left corner is the PAW++ Executive window, with its Input pad at the bottom and the Transcript pad at the top.

- The PAW++ Browser, where the various entities (pictures, 1-D and 2-D histograms and Nuples) are all defined with their own symbol, is shown bottom left. A “pop-up” menu has been activated for the chosen 1-D histogram. Several actions like Plot, Smooth, Fit etc... can be performed via this menu.

- The graphics window is seen top right. A 1-D view of the data points and two 2-D views (a Surface-plot and a colored contour plot) are shown. On the 1-D view, two 1-D histograms are superimposed. The results of a “smoothing” type of fit to the data points is also drawn. Information about the data and the fit can be found in the inserted window.

- The histogram style panel at the lower right allows graphics attributes of the histogram to be controlled.

Figure 13: The PAW++/Motif screen with the Executive window
- The upper left corner shows the Ntuple viewer. The left window shows the name of the various variables, characterizing the selected Ntuple. Other windows and press-buttons specify which combinations of the various variables and which events have to be treated (plotted, scanned, ...).

- The lower left contains the PAW++ Browser, with this time an Ntuple selected. A double on a Ntuple icon open automatically the Ntuple viewer on the active Ntuple.

- The graphics window is seen top right and shows a 3-D view of the combination of three variables, whose cuts are specified with the cut editor (see below).

- Direct graphics interactions with Ntuple data are possible. Just by clicking on a point in the graphics window, the event description is displayed in the PAW++ locate window.

- The cut editor panel, shown at the lower right, allows various combinations of cuts to be specified and applied.

Figure 14: The PAW++/Motif screen with the Ntuple viewer
1 A ZEBRA bank structure

2 ZEBRA stores and RZfiles

3 Browser display of ZEBRA banks

4 Popup menu with actions for class: Banks

Figure 15: Example of the DZDOC Motif interface with ZEBRA
4 Ntuples

To introduce the concept of Ntuples, let us consider that we want to analyze a Data Summary Tape (DST), which contains 100000 events. Each event consists of NVAR variables (say 200) for which we would like to make some distributions according to various selection criteria. One possibility is to create and fill 200 histograms on an event-by-event basis while reading the DST. An alternative solution, particularly interesting during interactive data analysis with the data presentation system PAW, is to create one Ntuple. Instead of histogramming the 200 variables directly, and thereby losing the exact values of the variables for each event and their correlations, the variables are first stored in an Ntuple (one can of course fill histograms at the same time!). An Ntuple is thus a distribution of points in a space of NVAR dimensions, or, can also be considered as a table where the NVAR variables are the columns, and each event is a row. The storage requirement is proportional to the number of columns in one event and can become significant for large event samples. An Ntuple can thus be regarded as an interesting way of storing a DST. The left hand side of Fig. 16 shows this schematically.

![Diagram of Ntuple]

Figure 16: The basic ideas behind Ntuples

Once the events are stored in this form, it becomes easy, to make one- or more-dimensional projections of any of the NVAR variables of the events and to change the selection mechanisms, or the binning and so on. Moreover, before running the system on a large number of events, the selection mechanisms can be rapidly tested on a small sample.
4.1 CWN and RWN – Two kinds of Ntuples

In a *Row-Wise-Ntuple* (RWN) the elements of each row, usually corresponding to an individual event, are stored contiguously in an HBOOK RZ file. This storage method is similar to that of a conventional DST, where events are stored sequentially and it is particularly suited for small Ntuples (up to a few Mbytes), with only a few columns. One can even use an RWN for larger Ntuples (up to about 20 Mbytes) when almost all columns are referenced in the query commands. A RWN should not be used if there are more than about 100 columns, or when queries reference only a small number of columns. A RWN can contain only floating-point data. It is created with HBOOKN and filled with HFN. Routines HGN, HGNF are used to retrieve information about one row.

![Schematic structure of a RWN Ntuple](image)

$x_1, y_1, z_1, t_1$, etc. represent the columns of event $i$.

Figure 17: Schematic structure of a RWN Ntuple

Figure 17 shows schematically how a RWN is laid out in memory, row after row. The buffer size in memory NWBUFF is specified as the primary allocation parameter NWBUFF of the HBOOKN routine. Of course, one must have reserved sufficient space in the /PAWC/ common when calling the HBOOK initialization routine HLIMIT. The lower line shows how the information is written to an RZ file. The length of the input/output buffer LRECL is specified as an argument of the routine HROPN. It is evident that, if one has a small Ntuple and a lot of memory, one can fit the complete Ntuple in memory, thus speeding up the Ntuple operations.

In a *Column-Wise-Ntuple* (CWN) the elements of each column are stored sequentially. Data in such an Ntuple can be accessed in a much more flexible and powerful manner than for a RWN. The CWN storage mechanism, as shown schematically in Fig. 18, has been designed to substantially improve access time and facilitate compression of the data, thereby permitting much larger event samples (several hundreds of Mbytes) to be interactively processed, e.g. using PAW. Substantial gains in processing time can be obtained, especially if queries reference only a few columns. A CWN is not limited to floating point data, but can contain all basic data types (real, integer, unsigned integer, logical or character). A CWN is created with routines HBNT, HBNAM or HBNAMC and filled with HFNT and HFNTB. Information about one row/block/column can be retrieved with routines HGNT, HGNTB, HGNTV and HGNTF.

4.2 Row-Wise-Ntuples (RWN)

4.2.1 Booking and filling a RWN

```c
CALL HBOOKN (ID, CHTITL, NVAR, CHRZPA, NWBUFF, TAGS)
```

Routine HBOOKN books a RWN in memory or with automatic overflow on an RZ file. Only single
$x_i, y_i, z_i, t_i$, etc. represent the columns of event $i$.

Figure 18: Schematic structure of a CWN Ntuple

The buffer size for each of the columns NWBUFF was set equal to the record length LRECL (defined with routine HROPN). A CWN requires a large value for the length of the common /PAWC/, in any case larger than the number of columns times the value NWBUFF, i.e. NWPAWC$\times$NWBUFF$\times$NCOL. Important performance improvements can, however, be expected by setting the buffer size NWBUFF equal to a multiple of the record length LRECL (via routine HBSET).

Precision floating point numbers (REAL*4 on 32 bit machines) can be stored, no data compression is provided. An RWN is filled by a call to routine HFN. An example is shown below:

```fortran
PARAMETER (NCOL=5) CHARACTER*2 TAGS(NCOL)
REAL X(NCOL)
DATA TAGS/'Px','Py','Pz','Qz','Nu'/
*
CALL HBOOKN(10,'A small RWN Ntuple',
+ NCOL,' ',1000,TAGS)
DO 100 I=1,1000
   READ(1,'(SE11.5)') X
   CALL HFN(I,X)
100 CONTINUE
```

4.2.2 Memory resident Ntuples

When a RWN is booked, HBOOK reserves space in memory in which to store the data, as specified by the NWBUFF argument of routine HBOOKN. As the RWN is filled by a call to HFN, this space will be successively used until full. At that time, a new memory allocation will be made and the process continues.

If this data is then written to disk, it appears as one enormous logical record. In addition, it most cases the last block of data will not be completely filled thus wasting space.

If one tries to re-process this RWN at a later date, there must be enough space in memory to store the entire Ntuple. This often gives rise to problems and so the following alternative method is recommended.

4.2.3 Disk resident Ntuples

Prior to booking the RWN, a new HBOOK RZ file is created using HROPN. The top directory name of this file is passed to routine HBOOKN when the Ntuple is booked.
Filling proceeds as before but now, when the buffer in memory is full, it is written to the HBOOK file and then reused for the next elements of the Ntuple. This normally results in a more efficient utilisation of the memory, both when the Ntuple is created and when it is re-processed.

When the Ntuple is filled, routine HFN will automatically write the buffer to the directory in the RZ file, which was specified in the call to HBOOK3. If the current directory (CD) is different, HFN will save and later automatically restore the CD. Figure 19 shows various ways to create and store Ntuples.

![Diagram showing the process of creating and storing Ntuples]

Figure 19: Different ways of creating Ntuples

4.3 Column-Wise-Ntuples (CWN)

A CWN supports the storage of the following data types: floating point numbers (REAL*4 and REAL*8), integers, bit patterns (unsigned integers), Booleans and character strings.

4.3.1 Data compression

Floating point numbers, integers and bit patterns can be packed by specifying a range of values or by explicitly specifying the number of bits that should be used to store the data. Booleans are always stored using one bit. Unused trailing array elements will not be stored when an array depends on an index variable. In that case only as many array elements will be stored as specified by the index variable.

For example, the array definition NHITS(NTRACK) defines NHITS to depend on the index variable NTRACK. When NTRACK is 16, the elements NHITS(1..16) are stored, when NTRACK is 3, only elements NHITS(1..3) are stored, etc.
4.3.2 Storage model

Column-wise storage allows direct access to any column in the Ntuple. Histograming one column from a 300 column CWN requires reading only 1/300 of the total data set. However, this storage scheme requires one memory buffer per column as opposed to only one buffer in total for an RWN. By default the buffer length is 1024 words, in which case a 100 column Ntuple requires 409600 bytes of buffer space. In general, performance increases with buffer size. Therefore, one should tune the buffer size (using routine HBSET) as a function of the number of columns and the amount of available memory. Highest efficiency is obtained when setting the buffer size equal to the record length of the RZ HBOOK file (as specified in the call to HROPEN). A further advantage of column wise storage is that a CWN can easily be extended with one or more new columns.

Columns are logically grouped into blocks (physically, however, all columns are independent). Blocks allow users to extend a CWN with private columns or to group relevant columns together. New blocks can even be defined after a CWN has been filled. The newly created blocks can be filled using routine HFNTB. For example, a given experiment might define a number of standard Ntuples. These would be booked in a section of the code that would not normally be touched by an individual physicist. However, with a CWN a user may easily add one or more blocks of information as required for their particular analysis.

Note that arrays are treated as a single column. This means that a CWN will behave like a RWN, with the addition of data typing and compression, if only one array of NVAR elements is declared. This is not recommended as one thereby loses the direct column access capabilities of a CWN.

4.3.3 Performance

Accessing a relatively small number of the total number of defined columns results in a huge increase in performance compared to a RWN. However, reading a complete CWN will take slightly longer than reading a RWN due to the overhead introduced by the type checking and compression mechanisms and because the data is not stored sequentially on disk. The performance increase with a CWN will most clearly show up when using PAW, where one typically histograms one column with cuts on a few other columns. The advantages of having different data types and data compression generally outweigh the performance penalty incurred when reading a complete CWN.

4.3.4 Optimizing event loops

PAW is able, before starting the loop over the events, to find out which are the columns of the CWN which are actually referenced in any given query (command line selection expression or COMIS routine). Only the columns which are referenced will be read from the file.

When analyzing Ntuple data without PAW, e.g. in a batch job, it is the user's responsibility to find out and specify which are the columns referenced via the routines HGNTV or HGNTB (if all variables in a block are needed).

This optimization can be very important. Assume, for example, that a 100 Mbyte file contains an Ntuple consisting of 300 columns, and that 3 columns are referenced. Then without optimization the complete 100 Mbyte file will have to be read, while, by specifying the columns used, only 1 Mbyte will be input.

4.3.5 Booking a CWN

The booking is performed in two stages. Firstly, a call to HBNT is made to declare the Ntuple identifier and title. Secondly, one or more calls are made to HBNAME or HBNAMC to describe the variables that are to be stored in the Ntuple. Routine HBNAMEC is used to define CHARACTER variables, while all other variable types are defined with routine HBNAME.

```
CALL HBNT (ID,CHTITL,CHOPT)
```

The CWN will be stored in the current HBOOK directory. The variables to be stored in the Ntuple will be specified with routine HBNAME or HBNAMC described below.

When the CWN is to be filled with HFNT, the memory buffers associated with each column will be written to the file and directory corresponding to the current working directory when HBNT was called.
Remember that when routine HROPEN is called, the current working directory is automatically set to the top directory of that file. It is therefore convenient to call HBNT immediately after HROPEN. If this was not the case, routine HCDIR must be called prior to HBNT to set the current working directory. When the Ntuple has been filled (via calls to HFNT) the resident buffers in memory as well as the Ntuple header must be written to the file with a call to HROUT. Before calling HROUT, the current working directory must be set to the current directory when HBNT was called.

\[
\text{CALL HBSET (CHOPT, IVAL, IERR*)}
\]

This routine sets global Ntuple options. At present only the option 'BSIZE' is supported, which sets the size of the buffer in memory for each Ntuple column equal to IVAL words (default 1024 words). If the total memory in common /PAWC/, allocated via HLIMIT, is not sufficient to accommodate all the column buffers of the given size, then HBNT will automatically reduce the buffer size in such a way that all buffers can fit into memory. It is strongly recommended to allocate enough memory to /PAWC/ in such a way that each column buffer is at least equal to the block size of the file. A simple rule of thumb in the case of no data compression is to have NWPAC\*NCOL\*LREC, where NWPAC is the total number of words allocated by HLIMIT, LREC is the block size of the file in machine words as given in the call to HROPEN, and NCOL is the number of columns.

Describing the columns of a CWN

\[
\text{CALL HBNAME (ID, CHBLOK, VARIABLE, CHFORM)}
\]

\[
\text{CALL HBNAME (ID, CHBLOK, VARIABLE, CHFORM)}
\]

These routines describe the variables to be stored in a CWN (non-character and character variables, respectively).

With CHFORM the variables, their type, size and, possibly, range (or packing bits) can all be specified at the same time. Note however that variable names should be unique, even when they are in different blocks. In the simplest case CHFORM corresponds to a COMMON declaration, e.g.

\[
\text{COMMON /CEXAM/ IEVENT, ISWIT(10), IFINIT(20), NEVENT, NRNDM(2)}
\]

can be described by the following CHFORM:

\[
\text{CHFORM = 'IEVENT, ISWIT(10), IFINIT(20), NEVENT, NRNDM(2)'}
\]

In this case Fortran type conventions are followed, default sizes are taken and no packing is done. Note that to get a nice one-to-one correspondence between the COMMON and the CHFORM statements the dimension of the variables are specified in the COMMON and not in a DIMENSION statement.

The default type and size of a variable can be overridden by extending the variable name with :<t>*<s>:

<table>
<thead>
<tr>
<th>&lt;t&gt;</th>
<th>type of variable</th>
<th>&lt;s&gt; values</th>
<th>default</th>
<th>routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>floating-point</td>
<td>4, 8</td>
<td>4</td>
<td>HBNAME</td>
</tr>
<tr>
<td>I</td>
<td>integer</td>
<td>4, 8</td>
<td>4</td>
<td>HBNAME</td>
</tr>
<tr>
<td>U</td>
<td>unsigned integer</td>
<td>4, 8</td>
<td>4</td>
<td>HBNAME</td>
</tr>
<tr>
<td>L</td>
<td>logical</td>
<td>4</td>
<td>4</td>
<td>HBNAME</td>
</tr>
<tr>
<td>C</td>
<td>character</td>
<td>[4≤s≤32] (multiple of 4)</td>
<td>4</td>
<td>HBNAME</td>
</tr>
</tbody>
</table>

When the range of a type U, I or R variable is known, its storage size (number of packing bits) may be added behind the :<t>*<s> specification using :<b> for types U and I and :<b>:[<min>,<max>] for type R.

When :<b>... is not specified a variable is stored using the number of bytes given by <s> or the default. In case the default type and size of a variable should be used, the packing bits can be specified as ::<b>...; <b> must be in the range 1≤b≤8.<s>. Automatic bit packing will happen, for type U and I, when a range is specified like: ICNT [-100, 100]. In this case ICNT will be packed in 8 bits (7 bits for the integer part and 1 bit for the sign). In case CNT is an integer ranging from -100 to 100 it could be
specified either like CNT[-100, 100]:I or like CNT: I: : [-100, 100]. Logical variables will always be stored in 1 bit. All variables must be aligned on a word boundary and character variables must have a length modulo 4. The maximum length of the variable name is 32 characters.

Variable-length Ntuple rows and looping over array components are also supported to optimize Ntuple storage and Ntuple plotting. Variable row length can occur when arrays in the Ntuple depend on an index variable.

---

**Example of a variable row length CWN definition**

```plaintext
PARAMETER (MAXTRK = 100, MAXHIT = 300)
COMMON /CMTRK/, NTRACK, NEITHS(MAXTRK),
+ PX(MAXTRK), PY(MAXTRK), PZ(MAXTRK)
CALL HBNAME(ID, 'VARBLOK2', NTRACK,
+ 'NTRACK[0,100], NEITHS(NTRACK)[0,300],'
+ 'AX(NTRACK), PY(NTRACK), PZ(NTRACK)')
```

---

In this example the number of elements to store in one Ntuple row depends on the number of tracks, NTRACK. The call to HBNAME declares NTRACK to be an index variable and that the size of the Ntuple row depends on the value of this index variable. The range of an index variable is specified using [l<r>, <u>r], where <l>r is the lower and <u>r the upper limit of the arrays using this index variable. In the above example the lower limit of NTRACK is 0 and the upper limit is 100 (= MAXTRK). While filling a CWN HBOOK can also easily test for array out-of-bound errors since it knows the range of NTRACK. Only the last dimension of a multi-dimensional array may be variable and the index variable must be specified in the block where it is used. Array lower bounds must, just like the lower range of the index variable, be 0.

HBNAME may be called more than once per block as long as no data has been stored in the block. New blocks can be added to an Ntuple at any time, even after filling has started, whereas existing blocks may only be extended before filling has started.

### 4.4 Ntuple variables and selection functions

Operands and expressions used as selection criteria in Ntuple operations are of the following types:

#### Numerical
Real and integer numbers, e.g. 23.877, 23, -2.E3, 0

#### Character string
This is a string of characters (plus the wild characters '*' and '?' see “Other features” below) written in the following form: 'AbCd' (upper and lower case, within quotes)

#### Bit strings (unsigned integers)
These are numbers written in hexadecimal, octal or binary form enclosed in single quotes.
- Hexadecimal numbers are written in the form Z'xxxx', where x is a hexadecimal digit in the range 0 - F.
- Octal numbers are written in the form 0'oooo', where o is an octal digit in the range 0 - 7.
- Binary numbers are written in the form B'bbbb', where b is either 0 or 1.

#### Logical
Logicals are written in the form TRUE (true) and FALSE (false), i.e., upper or lower case without quotes. This implies that the names TRUE and FALSE are not permitted as Ntuple column identifiers.

#### Cuts
A cut is identified by a '$' followed by an integer (written in the form $nn, where 0<n<100). A legal Cut expression may include any kind of expression, other cuts, masks or COMIS functions. The result of a cut is a real number (0. or 1. in case of a boolean or a graphical cut), so that the result of a boolean cut is mapped to the real numbers 0. and 1. so that it is possible to operate arithmetically on results of cuts.
Ntuple cuts definition

The NTUPLE/CUTS command

NTUPLE/CUTS CUTID ( OPTION FNAME )
CUTID Cut identifier
OPTION Options
FNAME File name

Define the CUTID with the format int. rin is an integer between 1 and 99.
This cut can then be used in subsequent commands NTUPLE/Plot, PROJECT.

Ntuple Drawing

The NTUPLE/Plot command

NTUPLE/Plot IDN [ UMFUNC NEVENT IFIRST NUPD OPTION IDH ]
IDN Ntuple identifier
UMFUNC Selection function
NEVENT Number of events
IFIRST First event
NUPD Frequency to update histogram
OPTION Options
IDH Identifier of histogram to fill

Figure 20: Using Ntuples inside PAW
Example of creating a cut

\[
\text{NTUPLE/CUT } \text{\$1 (SQRT(A+B).LE.3)}
\]
\[
\text{NTUPLE/CUT } \text{\$2 (A=3).AND.(B=5)}
\]
\[
\text{NTUPLE/CUT } \text{\$3 NOT(\$1).AND.\$2}
\]

Assuming there exists a mask vector MSK and a COMIS function COMIS1.FTN

\[
\text{NTUPLE/CUT } \text{\$4 (COMIS1>30.2).AND.MSK(3)}
\]

Example of using a cut

\[
\text{NTUPLE/PLOT } 30.X \text{\$1.OR.(B=12)}
\]
\[
\text{NTUPLE/SCAN } 30 \text{\$3}
\]
\[
\text{NTUPLE/PLOT } 30.X \text{\$4*5.3 | (5.3 is weight)}
\]

In summary, entries where the result of a cut is non-zero will be histogrammed, while all others will be ignored.

Masks

The Mask facility allows up to 32 selection criteria associated with a Ntuple. One bit of a mask (which has a logical type) is identified by the mask name followed by an index of the bit within parentheses. The index ranges from 1 to 32. Assuming there exists a mask vector MSK, then one can write, e.g. \text{NTUPLE/PLOT 30.X MSK(4) | (bit 4)}. The Mask vector is of type "bit string" and is identified by the name of the mask with no bit index and no parentheses, e.g. \text{NTUPLE/PLOT 30.X MSK.EQ.B'101'}. This is identical to the following, where the mask bits are referenced explicitly:

\[
\text{NTUPLE/PLOT 30.X .NOT.(MSK(2)).AND.(MSK(1))).AND.(MSK(3))}
\]

The bits are treated from right to left, i.e. position 1 is the least significant bit.

Operators

\[
\text{NTUPLE/PLOT 30.X X Y | weight X by Y}
\]
\[
\text{NTUPLE/PLOT 30.X X**2+Y**2 | weight X by X'2+Y'2}
\]

** Relational operators. **
\[
\text{EQ .EQ. = and NE .NE. <> GT .GT. >}
\]
\[
\text{LT .LT. <, GE .GE. >= LE .LE. <=, CT .CT. # (close to)}
\]
\[
\text{NTUPLE/PLOT 30.Y Y EQ 9}
\]
\[
\text{NTUPLE/SCAN 30 Y<Z'BFB'}
\]
\[
\text{NTUPLE/SCAN 30 Z='PAW'}
\]
\[
\text{NTUPLE/PLOT 30.W (W>3).GT.50}
\]
\[
\text{NTUPLE/SCAN 30 X>39.5}
\]
\[
\text{NTUPLE/SCAN 30 Y<10*X AND W>X | This is ambiguous and is NOT ALLOWED!}
\]
\[
\text{NTUPLE/SCAN 30 (Y<10*X)AND(W<X) | This will work.}
\]

Logical operators. \text{AND .AND., OR .OR. and NOT .NOT.}

| Only those entries satisfying the selection criteria are histogrammed
\[
\text{NTUPLE/PLOT 30.X (Z NE.'CERN').OR.(NOT(Y))}
\]
\[
\text{NTUPLE/PLOT 30.W .NOT.((Z EQ.'CERN').OR.Y)}
\]

| Only entries where Y < 0 are used, for the others X is not histogrammed
\[
\text{NTUPLE/PLOT 30.X X>(Y.LT.0)}
\]

| X is histogrammed only for entries where the value of COMISF is > 4
\[
\text{NTUPLE/PLOT 30.X X>(COMISF.F>4)}
\]

| Only entries where the value of cut $1'=0$ will be histogrammed
\[
\text{NTUPLE/PLOT 30.X $1=(COMISF.F+COMIS2.F=X)'}
\]

Functions.
\[
\text{SIN(n) COS(n) SQRT(n) EXP(n) LOG(n) ATAN(n)}
\]
\[
\text{ABS(n) INT(n) LOG10(n) TANH(n) ACOS(n) ASIN(n)}
\]
\[
\text{TAN(n) SINH(n) COSH(n) MOD(n,n) ATAN2(n,n) SIGN(n,n)}
\]

Bit handling routines. Routines from BITPAK (CERN Program Library M441).

| Logical operations \text{I0R(m,n) I0AND(m,n) NOT(m) I0E0R(m,n)}
\]
\[
| Shift operations \text{IS1HTF(m,k) IS1HTF0(m,k,ic)}
\]
\[
| Bit testing \text{IB1SET(n,i) IB1CLR(n,i) B1TEST(n,i)}
\]
\[
| Bit subfields \text{IB1TS(m,i,1en) MB1TS(m,i,1en,n,j)}
\]

Other features

Wild characters are allowed in character strings. The wild character is represented by '?' to replace any single character and '*' to replace any number of characters (including zero characters).
Examples of the uses of wild characters are: '\?A*' \*B' or '\????'

Substring selection notation, e.g., STRING(j:k) or NAME(1:2)="BO"

Mixed mode expressions are not allowed, e.g., \*X being Integer, (X*'TRUE') is invalid. The result of one operation can be the operand for the next operator, but the data types must be compatible.

Fig. 21 below and Fig. 22 on the next page show examples of manipulating Ntuple data.

Figure 21: Read and plot Ntuple elements
More complex Ntuple presentations

PAW > run 2.2
PAW > opt STAT
PAW > set HTYP -3
PAW > 1d 401 'NT/PL - X' 100. -2.5 2.5
PAW > nt/pl 30.1 ! -401
PAW > 1d 402 'NT/PL E option - Y' 100. -2.5 2.5
PAW > gset ntyp 21
PAW > nt/pl 30.y ! -402 !! E
PAW > 1d 403 'NT/PL B option - X' 40. -2.5 2.5
PAW > set bary 0.4
PAW > set baro 0.3
PAW > gset nb 0.33
PAW > set hcol 1001
PAW > nt/pl 30.x y>0 -403 !! b
PAW > 1d 404 'NT/PL PL option - Y' 100. -2.5 2.5
PAW > max 404 160
PAW > nt/pl 30.y sqrt(z)>1 -404 !! pl

| Divide plot in 4 zones
| Select option to write statistics on plot
| Define histogram hatch type
| Book 1 dim histogram
| Plot variable 1 (x) using histogram 401
| 1 dim histogram (different title)
| Select market type for points on plot
| Plot y variable with Error bar option
| 1 dim histogram (different title + binning)
| Define bar width for bar chart
| Define bar origin for bar chart
| Print selection criterion on plot
| Histogram colour black
| Plot x variable as bar chart
| 1 dim histogram (different title)
| Fix maximum for plotting hist 404
| Plot y variable with PL option

Figure 22: Selection functions and different data presentations
5 PIAF—Parallel Interactive Analysis Facility

The Parallel Interactive Analysis Facility’s main design goals were the creation of a system that would be able to process multi-gigabyte databases in real time, that would scale with increasing amounts of data and that would have an affordable price tag. To reach these ambitious goals we decided to run the efficient Ntuple database engine and PAW’s query processor in parallel on a dedicated cluster of powerful workstations, each equipped with large memories and fast disks and interconnected via a high bandwidth, low latency network. Other, secondary, design goals were to make the system as transparent as possible for the existing PAW users and to make the system easily clonable and portable.

In the next sections a description of the hardware configuration of the PIAF cluster currently in production at CERN and the organization of the PIAF client/server software is given. At the end the latest performance figures are given.

5.1 The PIAF hardware

The PIAF cluster currently installed at CERN consists of five HP755 workstations each with 128 Mbyte RAM, 2.6 Gbyte internal disk and a 6.5 Gbyte RAID disk. The machines are interconnected by an FDDI network and a conventional Ethernet.

Of the 2.6 Gbyte internal disk 1.6 Gbytes are configured as device swap, while the remaining 1 Gbyte is used for the operating system and the users’ work directories. The RAID disks are used to store the Ntuple database files. Since the Ntuples are staged from tapes or large file servers on to PIAF, the Ntuples are considered to be “volatile” data. The 6.5 Gbyte RAID systems (HP-C2425) are, therefore, configured to provide high read performance while no data security is required. By running the RAID’s in mode 0/independent with software-disk-stripping (sds) a maximum read performance of 4.5 Mbytes/s is obtained without losing space due to data redundancy. By cross-mounting the RAID systems, using NFS over FDDI, each machine in the cluster has access to a disk pool of 32.5 Gbytes. An overview of the current hardware configuration is given in Fig. 23.

In the near future the cluster will be expanded with three additional HP755 (same configuration as the existing nodes) and five additional 10 Gbyte RAID systems (HP-C2430). Around the same time the eight nodes will be upgraded to 256 Mbyte RAM each and in spring 1994 the CPU’s will be upgraded to the new 130 MHz model (50% faster than the current CPU’s).

5.2 The PIAF software

The core of the PIAF software is based on the PAW query processor and the Ntuple database engine. These core components are packaged together with inter-process communication routines in the piafserv program. Depending on its startup arguments this program can either become a master server or a slave server. Each user connecting to PIAF will get one master server and N slave servers, where N is the number of available nodes, see Fig. 24.

The user’s local PAW session acts as the client which communicates over the CERN-wide ethernet with the PIAF master server. The master server coordinates the database queries: it farms out the query commands to the slave servers and receives and merges the results coming from each slave. Each slave knows exactly which part of the total database it has to process thanks to its group view. The groupview contains two numbers: the slave server’s unique ID (from 1 to N) and N (the number of active slave servers). If a slave does not respond within a certain time limit it is considered dead and the master server broadcasts a new groupview to the remaining slaves.

The interprocess communication is done via BSD sockets (for efficiency reasons we did not use a message passing library like PVM, which relies on a number of extra daemons hanging around the cluster).

5.2.1 Transparency

To keep the system as transparent as possible for the existing PAW users we limited the set of new commands to access PIAF to the bare minimum. The opening and closing of a PIAF connection is done using the commands CONNECT and DISCONNECT. When connecting to PIAF the user is asked for his PIAF login and password. Once connected to PIAF the user can stage Ntuple database files using the PUT and/or STAGE commands.
Figure 23: PIAF hardware

Figure 24: PIAF processes
Figure 25: Command processing on PIAF
To access data files on PIAF the user simply precedes the file name with the keyword "//piaf". For example, compare the two following commands, which both tell PAW to connect the file "staff.hbook".

```
HISTO/FILE 1 //piaf/staff.hbook | connect file on PIAF to logical unit1
HISTO/FILE 2 staff.hbook        | connect file locally to logical unit2
```

From this moment on Nuples in both files can be queried in the same way. The only difference is that when an Nuple from logical unit 1 is queried the query command is transferred to the master server and via the master server to the slave servers, as seen below:

```
NT/PLOT //LUN1/10.age cost>10000 | executed in parallel on PIAF
NT/PLOT //LUN2/10.age cost>4000 | executed by local PAW session
```

Figure 25 shows what happens behind the scenes when a command is executed on PIAF.

5.3 Performance figures

PIAF performance scales nearly linearly with the number of processors participating in the query. This is due to the small amount of time spent in the interprocess communication compared to the time it takes to traverse the database. The larger the Ntuple the better the speedup since the amount of communication is independent of the number of records processed (see Fig. 26).

![PIAF Execution Time](image1)

![PIAF Speedup](image2)

Figure 26: Real time and speedup with PIAF
6 Sharing data between various machines

6.1 Using PAW as a presenter on VMS systems (global section)

The standard version of PAW may be used as an online presenter on VMS systems using the mechanism of global sections. It is possible for two processes to reference the same histograms using *global sections*. For example, the first process may be a *histogram producer* (e.g., a monitoring task) and the second process PAW. As the histograms are being gradually filled by the first task, PAW can view them, and even reset them. To use the global sections, it is also necessary to “page align” the common which is in the global section. This is achieved in the “link step” when making the process (see Fig. 27). The relevant statements are SYS$INPUT/OPTIONS to tell the linker that some options follow the link statement, and PSECT=PAWC,PAGE which is the option to page align the /PAWC/ common.

```fortran
PROGRAM PRODUCE
PARAMETER MAXPAGES=100
COMMON/PAWC/IPAWC(128*MAXPAGES)
CHARACTER*8 GNAMES
INTEGER*4 HCREATEG

* GNAME='GTEST'
WAIT_TIME=1.
NUMEVT=1000

*................. Create Global section
NPAAGES=HCREATEG(GNAME,IPAWC,128*MAXPAGES)
IF(NPAAGES.GT.0) THEN
PRINT 1000,GNAME
1000 FORMAT(' Global Section: ',A,' created')
ELSE
IERROR=-NPAAGES
PRINT 2000,IERROR
2000 FORMAT(' Global Section Error', I6)
ENDIF
CALL HLIMIT(128*NPAAGES)

*................. Book histos.
CALL HBODK1(10,'Test1$',50,-4.,4.,0.)
CALL HBODK1(20,'Test2$',50,-4.,4.,0.)

*................. Fill histos.
DO 10 J=1,NUMEVT
   DO 10 I=1,100
      CALL RANURD(A,B)
      CALL HFILL(10,A,0.,1.)
      CALL HFILL(20,B,0.,1.)
   CONTINUE
10 CONTINUE
CALL LIB$WAIT(WAIT_TIME)
CONTINUE
20 CONTINUE

STOP
END
```

PAW > edit produce
macro produce ntimes=100
zone 1 2
histo/plot 10 K
histo/plot 20 K
do nt=1,[ntimes]
histo/plot 10 U
histo/plot 20 U
wait ' ' 1
enddo
return

PAW > global GTEST
PAW > exec produce ntimes=20

Figure 27: Visualising histograms in VMS global section
6.2 Using PAW and Unix shared memory

On Unix, PAW can be used as an online presenter using the shared memory facility. The example below shows on the left hand side the program, which fills the histograms. It is compiled and linked with the $f77$ command, and then started. It writes the lines shown while going through the event loop. Then PAW is started, communication is established via the command global TEST, which declares that the area called 'TEST' is to be shared for data communication, and the execution of the KUMAC macro shared_kumac, shown at the top right of the figure, is initiated.

The output shown on the screen then allows one to follow interactively (using the Update option 'U' of the plot command) how the event generator (the hserver program) fills histogram number one. The first fifteen iterations have been captured and are shown at the bottom right of Fig. 28.

Program hserver

* * HBOOK program creating a "shared memory"
 * area called 'TEST'
 * Routine HLIMIT replaces HLIMIT.
 * NWORDS is the amount of space requested
 * in the shared area.
 * parameter(nwords=50000)
 call hlimap(nwords,'TEST')
 *
 call bhook1(1,'test1',100,-3.,3.,0.)
call hcopy(1,2,'test2')
call hcopy(1,3,'test3')
 *
 do 10 i=1,100000000
  call rannor(a,b)
call hf1(1,a,1.)
call hf1(2,b,1.)
call hf1(3,a**2+b**2,1.)
  if(mod(i,1000000).eq.0)
    print *, ' hserver in loop index ',i
  end
 10 continue
 * end

$ f77 -l... -l... -o hserver hserver.f
$hserver
GLOBAL MEMORY CREATED,
  offset from LQ = 1037452510
hserver in loop index 1000000
hserver in loop index 2000000
hserver in loop index 3000000
hserver in loop index 4000000
hserver in loop index 5000000
hserver in loop index 6000000
hserver in loop index 7000000

Figure 28: Visualizing histograms in Unix shared memory
6.3 Access to remote files from a PAW session

When running PAW, it is often necessary to access files (e.g. HBOOK files) which reside on a different computer. Therefore a PAW server is provided (the program `pawerv`, which works using a conventional Client/Server model. The client (PAW) typically runs on a workstation. When the PAW command `rlogin` is invoked, a PAW server is automatically started on the remote machine, normally a mainframe or data server.

Once the `LOGIN_REMOTE` command has been executed, the PAW Current Directory is set to `//REMOTE`. The PAW client can now instruct the PAW server to attach a file using the `rshell` command (e.g. `rshell file pawtest.dat`). If an histogram with HBOOK ID=10 is on the remote file, than the PAW command `Histo/Plot 10` will plot this histogram on the local workstation. The histogram resides on `//PAWC` like other histograms coming from local files.

The `rshell` command may be used to communicate with the PAW server. The expression typed following `rshell` is passed to the server. The current implementation of the PAW server recognizes the commands:

- `rshell file filename` Server connects filename
- `rshell cdir //lun11` Server changes current directory
- `rshell ld` Server lists current directory
- `rshell ld //` Server lists all connected files
- `rshell message` Server pass message to operating system

An example of a session accessing data remotely is shown below.

```
```

### Access to remote files from a workstation

```

```

PAW > rlogin CERNVM | connect to CERNVM
PAW > rshell file HRZTEST.HBOOK | PAW server connects HRZTEST HBOOK A to //LUN11
PAW > histo/plot 10 | plot histogram 10 from CERNVM
PAW > histo/fit 20 G | fit histo 20 with a Gaussian and plot it
PAW > rlogin AXCRNA | connect to AXCRNA
PAW > rshell file DISK$DL:[PAW] HEXAM.HBOOK;3 | PAW server on AXCRNA connects file to //LUN11
PAW > histo/plot 110 | plot histogram 110 from AXCRNA
PAW > rshell file HRZTEST.HBOOK | PAW server on AXCRNA connects file to //LUN12
PAW > histo/plot 110 s | plot histogram 110 from HRZTEST.HBOOK
PAW > rshell ld // | on AXCRNA on the existing picture
PAW > cdir //CERNVM | list all files connected on AXCRNA
PAW > histo/plot 110 | Change current PAW directory to CERNVM
PAW > histo/plot //AXCRNA/110 | plot histogram 110 from CERNVM
PAW > cdir //PAWC | plot histogram 110 from AXCRA
PAW > histo/list | current directory to local memory
PAW > Histo/delete 0 | list all histograms in //PAWC
PAW > bria //AXCRNA/0 | delete all histograms in memory
PAW > cdir //CERNVM | read all histograms from AXCRA
PAW > rshell file NEW.HBOOK.D 1024 N | file HRZTEST.HBOOK to //PAWC
PAW > broot 0 | change directory to CERNVM
PAW > rshell file NEW.HBOOK.D | creates a new file on the D disk
PAW > broot 0 | write all histograms from //PAWC
to CERNVM file NEW HBOOK D

175
7 Implementation

The package has been implemented in a **totally portable** way. This was achieved by making full use of Fortran 77 and ANSI C, of a general interface (HIGZ) to standard graphics packages and of a standard data structure manager, ZEBRA. The package is also fully transparent to the operating system, permitting a direct interface to the systems tools. Figure 29 is an overview of the computer systems where PAW is implemented, and which graphics packages are available.

In order to quantify in some way the complexity of the system we could just quote that the total number of lines of code of the seven modules composing PAW is about 300,000. This figure does not include the underlying graphics package.

<table>
<thead>
<tr>
<th>Computers</th>
<th>Graphics Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apollo Domain/Unix</td>
<td>GKS:</td>
</tr>
<tr>
<td>Convex</td>
<td>GKSGRAL</td>
</tr>
<tr>
<td>CRAY (Unicos)</td>
<td>SUNGKS</td>
</tr>
<tr>
<td>DEC Ulitrix</td>
<td>DECGKS</td>
</tr>
<tr>
<td>HP/UX</td>
<td>CERN Minimal GKS</td>
</tr>
<tr>
<td>IBM (VM and MVS)</td>
<td>PLOT10GKS</td>
</tr>
<tr>
<td>IBM RISC 6000</td>
<td>UNIGKS</td>
</tr>
<tr>
<td>Next</td>
<td>NOVAGKS</td>
</tr>
<tr>
<td>NORD</td>
<td>GKS2000</td>
</tr>
<tr>
<td>Sun (Sun/OS, Solaris)</td>
<td>GL (SGI, IBM/RS 6000)</td>
</tr>
<tr>
<td>Silicon Graphics (SGi)</td>
<td>GPR (APOLLO)</td>
</tr>
<tr>
<td>Windows-NT</td>
<td>X-Window</td>
</tr>
<tr>
<td>PC (MS-DOS, Windows, Linux)</td>
<td>PHIGS</td>
</tr>
<tr>
<td>VAX/VMS, VAX Station</td>
<td>CORE/DI3000</td>
</tr>
<tr>
<td>Alpha/VMS, OSF, NT</td>
<td>Tektronix/Falco Terminals</td>
</tr>
<tr>
<td>Macintosh A/X</td>
<td>PostScript</td>
</tr>
</tbody>
</table>

Figure 29: Systems where PAW is installed

8 Usage of PAW

PAW has been installed today at several hundred laboratories in the world. It is almost impossible to say how many users there are today, but our estimation is that their number is well above 20,000. Figure 30 shows statistics related to PAW as gathered during the year 1993. Nowadays most PAW sessions are on workstations, with PIAF's share steadily increasing. It is also seen that the number of distinct users for one year is above 2600.
Figure 30: Statistics on the usage of PAW during the period January 1993–January 1994 inclusive
Acknowledgements

The PAW authors would like to thank the CERN management for continuous support of the project, as well as, the very large number of users who are contributing to the development of the system with their suggestions and comments.

References


Computer Animation for Scientific Visualization

Michael E. Bauer & Ken C. Hu
Sterling Software, Palo Alto, CA, U.S.A.

Abstract

During the past few years computer animation has proven itself as a powerful tool to assist researchers in understanding complex scientific data sets. One major scientific application of computer animation is the field of computational fluid dynamics. In this field, animating the time dependent numerical solutions of such phenomena as shock waves, vortices, shear layers, and wakes is of major importance.

This paper concentrates on the mathematical representation of transformations for hierarchical objects such as translation, rotation, and scaling which is essential for animation. A description of the application of these transformations between world and local coordinate systems is presented. Several interpolation techniques to define the motion in key frame based animation systems are discussed. Comprehensive C-code covering these topics is provided in appendix B.

A brief description of other animation techniques such as cycled objects, camera choreography, parametric based, and model driven systems is presented. In appendix A a short discussion of the various hardware components required for an animation recording environment is given.

Introduction

What is computer animation? The most general description of computer animation may be stated as follows: Computer animations may be created by using a rendering device that will produce consecutive frames consisting of relative changes in visual effects. These changes in effects may be due to either motion dynamics, such as time-varying position of rigid or non-rigid bodies, or update dynamics. Update dynamics is associated with a change in shape, color, transparency, structure, or texture. Other changes in effects can be accomplished by modifying the lighting characteristics or changing the camera position.

Animation Environments

1) Homogeneous Coordinate System

Before one can discuss the various animation techniques, it is essential to understand the basic principles of the environments in which these animations are performed. If one analyzes the transformations of a vertex point P(x,y,z) to P'(x',y',z') where P is represented as a row vector and

\[ P' = P + T \] (for the translations)
\[ P' = P \times S \] (for scaling)
\[ P' = P \times R \] (for rotations),

then one can see that translations are a linear transformation, and the scaling as well as the rotations are associated with scalar transformations. By using the homogeneous coordinate system, all three transformations can be treated as multiplications. This is accomplished by introducing the w coordinate where

\[ P(x,y,z) \rightarrow P(X,Y,Z,w) \text{ and } w \neq 0. \]
The 3D Cartesian coordinates can then be represented as

\[
x = X/w, \quad y = Y/w, \quad z = Z/w.
\]

In computer graphics \( w \) is set equal to 1 to simplify computations. Hence the vertex point \( P \) in the homogeneous coordinate system can be represented as \( P(x,y,z,1) \).

2) Transformation Matrices
To transform an object in the homogeneous coordinate system, vertex points of an object are multiplied by a 4x4 matrix. The representative matrices for translations (T), scaling (S), and rotations (R) are shown below and will be referred to as such throughout this paper.

**Translations**

\[
T(T_x, T_y, T_z) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
T_x & T_y & T_z & 1
\end{bmatrix}
\]

**Scaling**

\[
S(S_x, S_y, S_z) = \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Rotations**

\[
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & \sin \theta & 0 \\
0 & -\sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_y(\theta) = \begin{bmatrix}
\cos \theta & 0 & -\sin \theta & 0 \\
0 & 1 & 0 & 0 \\
\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_z(\theta) = \begin{bmatrix}
\cos \theta & \sin \theta & 0 & 0 \\
-sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Note: The scaling and rotation matrices are derived to transform vectors about the origin of a Cartesian coordinate system.

3) World and Local Coordinate Systems for Hierarchically Defined Scenes
To discuss these two coordinate systems, an example of a robot will be used. The robot consists of a base, two arms, and a claw. A graphical representation of the robot is shown in Figure 1 below. Figure 2 shows the hierarchical relationship of the various objects. One can see in Figure 2 that the top node is referred to as the "picture node". The "picture node" represents the entire scene to be displayed. The world coordinate system (WCS) defines the space of the entire scene, and many objects can be placed within it. The initial placement as well as any transformations of these objects will be performed with respect to the origin of the WCS. An artificial pivot point may be introduced into the scene to provide extended motion capabilities. This will be discussed in more detail below. The data structure associated with the picture node is called "FOREST_DEF". The data structure associated with the root node and the children nodes is referred to as the "NODE". Both of these structures as well as several other
functions referred to later in this paper are available in the C-code reference listed at the end of this paper.

Referring to Figure 1, only one hierarchical object, the robot, which is defined in the scene, is shown. If additional objects were added, they would be attached to the picture node. The node which represents the entire object is called the "root node". Transformations of the "root node" may be performed either in the WCS or the local coordinate system (LCS). It is essential to allow the root node to be transformed in the WCS so that it can be moved throughout world space. In addition it is necessary to transform the object with respect to its own LCS moving about its own axes. Objects are modeled with respect to their own LCS, i.e., all points of the object are defined with respect to this coordinate system. In general the origin of the LCS coincides with the pivot point of the object. The pivot point may either be the center of rotation, center of mass, or any pre-defined point on the object.

In Figure 2, one can see that all other nodes, or objects, defining the characteristics of the robot, are linked in a hierarchical manner. These nodes are called children nodes. The node above a child node is called the parent node. Children nodes may have their own children attached to them. Children nodes may be transformed only in their own LCS. Without this restriction, the hierarchical linkage of the objects could be easily separated. Even with this restriction, problems may arise if care is not taken. It is generally necessary to introduce motion constraints on the individual nodes. For instance, an object that may be moved around a joint should not be allowed to translate; otherwise it will be separated from its parent node. The beauty of defining objects in a hierarchical manner is that children nodes may inherit the transformations as well as attributes from their parent.

Whenever transformations are performed in either the world coordinate or local coordinate system, respectively, world and local transformations are required. These will be discussed next.

![Figure 1: Display of a Hierarchically Linked Robot](image-url)
4) Transformations in World and Local Space

Whenever transformations are performed in either the LCS or the WCS, a set of rules are required to perform the matrix multiplications. The following techniques were implemented in an animation package developed at NASA Ames Research Center called Ames Research Center Animation Development Environment (ARCADE). This paper does not claim that the following method is the only way to perform transformations on hierarchically linked objects. This method, however, has produced the best results for the animation package.

All transformations in either world or local space require incremental update of the node transformations. An incremental change in motion of a selected node is added to the accumulated motion of this node.

4.1) Local Coordinate System

To describe transformations with respect to the LCS the following components need to be defined:

- $R_A$ = Incremental rotational matrix
- $S_A$ = Incremental scaling matrix
- $T_A$ = Incremental translational matrix
- $T_{LP}$ = Translations with respect to the local pivot point
- $M_L$ = Previously computed resultant transformation matrix in local coordinates
- $M'_L$ = New resultant transformation matrix in local coordinates

These components are used in the following matrix calculations:

\[
M'_L = R_A(\phi) * S_A(s_x, s_y, s_z) * M_L \\
M'_L = T_A(x_2, y_2, z_2) * T_{LP}(-x, -y, -z) * M_L * T_{LP}(x, y, z)
\]

For example, these matrix multiplications can transform an object (see Figure 3) from position $P_1$ to position $P_2$ via a set of translations, rotations, and scalings in the local coordinate system. To do this, a translation matrix must be established to translate the object from (A) to (B). Note that the scaling and rotation matrices defined previously are
only valid for transformations about axis' origin. To transform the object with respect to its pivot point, the pivot point is translated to the origin (C), the rotations and scales can then be applied (D&E), and the pivot point is then translated back to its previous location. Now the translational matrix is applied to position the object to the point P₂. Appendix B provides sample C code performing these operations (see the function called "local_transform()").

![Figure 3: 2D Transformation with Respect to the Local Coordinate System](image)

4.2) **World Coordinate System**

To describe transformations with respect to the WCS the following components need to be defined:

- $R_\Delta =$ Incremental rotational matrix
- $S_\Delta =$ Incremental scaling matrix
- $T_\Delta =$ Incremental translational matrix
- $T_{WP} =$ Translations associated with the world pivot point
- $M_W =$ Previously computed resultant transformation matrix in world coordinates
- $M'_W = $ New resultant transformation matrix in world coordinates

\[ M'_W = M_W \cdot T_\Delta(x_2, y_2, z_2) \cdot T_{WP}(-x, -y, -z) \cdot R_\Delta(\theta) \cdot S_\Delta(s_x, s_y, s_z) \cdot T_{WP}(x, y, z) \]

For example, these matrix multiplications transform an object from position P₁ to position P₂ via a set of translations, rotations, and scales in the world coordinate system. Again a translation matrix may be established to translate the object from (A) to (B). Next the rotations and scales of the object may be computed about an arbitrarily defined pivot point in world space. Generally, this pivot point is set to equal the world axes' origin. However, to allow more motion flexibility it may be located anywhere in world space. The rotations and scales are then performed about this arbitrary pivot point. As mentioned earlier, all rotations and scales must be computed about the axes' origin. To
assure this, the arbitrary pivot point is translated to the world axis origin (C), the rotations and scales can then be applied (D&E), and the arbitrary pivot point is then translated back to its previous location. At this point the translational matrix is applied to position the object to the point P2. The C-code performing these operations can be found in the function called "world_transform()". Note in Figure 4, section E, that, whenever a scaling operation is performed on an object about an arbitrary world pivot point, a translation toward this pivot point is introduced. This can be very useful in animating imploding (bursting inward) objects.

![Figure 4: Transformation with Respect to the World Coordinate System](image)

4.3) Switching Between World and Local Transformations

As mentioned above the root node of any object may be transformed with respect to either the WCS or the LCS. To assure continuity between switching from one space to the other, the object matrix must be converted to the respective coordinate space. Without this conversion, the object would not maintain its position (i.e., it would jump when switching between WCS -> LCS ->WCS). Simple matrix multiplications on the node matrix M must be performed to maintain continuity. The following general equation is used to switch from the WCS to the LCS or from the LCS to the WCS depending on the value of "direction."

\[ M' = \text{direction} \cdot (-T_{\text{LP}}) \cdot M \cdot \text{direction} \cdot (+T_{\text{LP}}) \]

where LCS -> WCS: direction = 1
WCS -> LCS: direction = -1
Animation Techniques

5) Key Frame Systems

Now that the fundamentals (e.g., coordinate systems, hierarchy, etc.) have been covered, it is time to move on to animation techniques. The most commonly used animation technique is called "key frame" animation. Key frame animation was originally developed by Walt Disney to create cartoons (reference #1). In these systems, skilled animators designed or choreographed images at a specific instance in time representing the animation. These images, the so called key frames, would then be interpolated by less skilled animators to complete a sequence of "in-between" frames. Animating the key frames along with the in-between frames resulted in the desired cartoon.

Within computer graphics, the key frames represent a 4x4 transformation matrix associated with an object (node). The in-between frames are then computed by the computer using a variety of interpolation techniques. The most frequently used interpolation techniques are linear, Hermite, splines, and quaternions, all of which are described in this section.

5.1) Key Frames in Computer Graphics

To create the desired key frames it helps to have an intuitive user interface to control the manipulation of scene objects. Selecting desired nodes from some sort of tree representation (see Figure 5) of the hierarchical object works quite well. Once a node has been selected, transformations can be used to establish a series of key frame matrices. For example, the manipulation of a hierarchically based robot object (see Figure 5) to create key frames (see Figure 6) could adhere to the following scenario. First the user would select the robot arm1 object (node at (A)) and perform a rotation about the z-axis. This matrix is then saved as the first key frame. Note that a translational motion constraint, to prohibit translations in either x, y, or z direction, should be imposed on any of these "arm" joint nodes. Translating these nodes would separate them from their parent node. It can be clearly seen in Figure 6 that the children nodes inherit the transformation of the applied matrix. Next a transformation matrix of the robot claw object (node at (B)) and the transformation matrix of the robot arm2 object (node at (C)) are saved, each creating their own key frame matrix. By interpolating the motion of the key frame matrices and displaying the object with each interpolated matrix, a nice motion of the robot can be visualized.

The C-code to display the current picture as well as display the nodes of a tree is shown in the functions "display_scene()" and "display_tree()". The function "display_tree()" takes care of the inheritance issues associated with hierarchical objects. This is accomplished by using the function calls pushmatrix, popmatrix and pushattributes and popattributes which preserve or delete the matrix and attribute information on a stack accordingly.
5.2) Controlling the Kinematics of the Animation

The placement of the key frames with respect to time determines the kinematics (motion) of an object. Key frames, which are placed closer together with respect to time, produce a faster motion than key frames that are placed farther apart. It is the opinion of the authors that an interactive user interface should be used to easily facilitate control of
the kinematics in an animation. As an example the user interface developed for the animation package ARCADE is shown in Figure 7.

![Figure 7: Animation Strand within ARCADE](image)

The user interface shown in Figure 7 displays a graphical representation of the time line for each node that will be animated. Each displayed time line is called a "strand." Besides having control features, the strand also contains a key frame "bucket" where graphical representations, i.e., icons, of the key frames are displayed. A user can place these matrix icons onto the strand of the respective node to control the kinematics of the animation.

5.3) Interpolating the Key Frames

Key frame interpolation is an essential part of animation because it allows the animator to define a small number of strategic frames that the software then uses to create numerous in-between frames, thereby producing a smooth animation. There are several interpolation techniques possible (reference #1, #2, #3), however only a few of them are suited to produce an appropriate animation. Figure 8 shows possible candidates for an interpolation method. Each curve in this figure will be discussed in more detail.

![Figure 8: Spline Comparison Diagram](image)
5.3.1) Linear Interpolation

The easiest interpolation technique is linear interpolation. Linear interpolation generates continuous motion, however it also generates first- and second-order derivative discontinuities at the key frames. As a result the animation experiences abrupt changes in direction at the key frames which give the animation a mechanical look. Sometimes this is the desired effect especially with very rigid bodies; however, a much smoother motion is usually desired. One advantage of the linear interpolation is that it guarantees that the animated path will go through the set of selected key frames.

5.3.2) Hermite Interpolation

A much more desirable interpolation technique is the Hermite interpolation. The Hermite interpolation guarantees derivative continuities, hence producing the desired smooth motion. As with linear interpolation, the animated path of the Hermite interpolation passes through the set key frames. One potential disadvantage of the Hermite interpolation is the exaggerated path it produces, which can be seen in Figure 8. The top portion of the loop swings out beyond the region of the key frames. This region may, however, no longer be in the viewing volume (visible on the screen), and the object could temporarily disappear from the displayed scene.

5.3.3) Cubic B-Spline

Splines in general prevent objects from disappearing from the viewing volume because splines are confined by the convex hull of the set of control points. As long as each control point (key frame) is in the viewing volume, at least a portion of the object will always be visible in the scene. The cubic B-spline produces derivative continuities at the control point and enjoys the properties of being confined within the convex hull. The animation path of the cubic B-spline generally does not pass through the control points, which may create a problem. This can be overcome by doubling or tripling the particular vertex point as was done at the beginning and end control point in Figure 8. This action considerably reduces the motion's smoothness if performed on other than the beginning and end control points and it increases the number of required computations.

5.3.4) Bezier Curve

From the curves shown in Figure 8, the Bezier curve is the least desirable since it completely damps the animated path. The loop specified by the key frames shown in Figure 8 is "washed out" when using the Bezier curve to compute the in-betweens; consequently, it is rarely used for key frame animations.

5.3.5) Beta Spline

The Beta spline (see Figures 9 and 10) has many of the same advantages as the cubic B-spline, such as continuous derivatives at the control points and the spline being confined within the convex hull. Furthermore, the Beta spline has unique controlling parameters which provide total control of the curve's shape, making this spline a nice candidate for computing the in-betweens. The parameters that control the shape of this curve are the bias ($\beta_1$) and the tension ($\beta_2$). In Figure 9 the tension is set to zero and the bias is changed. One can clearly see that when $\beta_1 = 0$, a linear interpolation is obtained. When $\beta_1 = 1$, the Beta spline is equal to the cubic B-spline. As the bias is increased the actual shape of the curve can be controlled by shifting toward the left and approaching the control points. In Figure 10 the bias was set to one and the tension was modified. As the tension is increased the curve is stretched vertically to approach the control points.
5.3.6) Quaternions

The above mentioned techniques work very well for computing the in-betweens for the translation and scaling parameters, but not for the rotation parameters. In the past
the Euler angles, the angles defining the 4x4 rotational matrix R above, have been used extensively for animating orientation; however, interpolating Euler angles leads to unnatural interpolation of the angles. For this reason quaternions should be used as the interpolation technique for rotations (reference #1, #2). In previous discussions the 4x4 matrix M was computed for either the WCS or the LCS. This matrix was computed by a series of matrix multiplications of translations, rotations, and scales. In order to perform the quaternion interpolation it is necessary to extract the rotational components from the matrix M. Since the scaling components \( S_x, S_y, \) and \( S_z \) are multiplied by the rotational components, this may not be a trivial task. If \( S_x = S_y = S_z \) then the task of extraction is simplified. However if they are not equal, shearing matrices must be introduced to extract the scaling components. Reference 4 provides a detailed explanation of this procedure. In general the matrix M must be separated into five different matrices, three of them are shearing matrices (see below). The in-betweens associated with shearing matrices, the scaling matrix as well as the translation parameters may then be computed either using splines, or the hermite or linear interpolation. The normalized 3x3 rotational matrix can now be converted to its respective quaternion representation. The quaternion in-betweens for the set of quaternion key frames may be computed using conventional interpolation techniques. This produces a spherical linear interpolation of the key frames. The in-betweens as well as key frames need to be converted back to their respective rotational matrix representations. At this point the related in-betweens for the translations, rotations, scales, and shears must be multiplied to produce the respective in-between matrix M used in the animation.

\[
M = \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
R[3X3] & 0 \\
0 & 0 \\
0 & 0 \\
T_x & T_y & T_z & 1
\end{bmatrix}
\]

C-code for the above mentioned routines is available via several functions (\texttt{normalize()}, \texttt{QuatFromMatrix()}, \texttt{QuatToMatrix()}, \texttt{Interpolate()}, \texttt{slerp()} ) in Appendix B.

6) \textbf{Cycled Object Animation Technique}

Another animation technique used in conjunction with the key frame systems is the cycled object animation technique (e.g., display of consecutive frames of time dependent objects or data), which is used extensively in scientific visualization. A large set of objects, representing either time dependent data or a display of varying physical object characteristics, such as position, shape or color, may be presented by the scientist by this method. Displaying each object of the entire set over time may produce an animation containing vast amounts of data. Generally key frames are established to define the position of the cycled objects within world space. Initially a skip range is established which determines the objects from the set that will be displayed. Next the set of objects are cycled with respect to time using several simple techniques. A bounce cycle may be used where the cycle begins the display with a pre-defined start object and incrementally displays the next object based on the skip range. Once the last object is reached, the cycle begins with the last object and cycles forward to the first object where the cycle starts again.

Bounce cycle: \( \text{obj}_s \rightarrow \text{obj}_i \rightarrow \text{obj}_j \rightarrow \text{obj}_m \rightarrow \text{obj}_n \rightarrow \text{obj}_i \rightarrow \text{obj}_s; \) ............
The looping cycle is similar to the bounce cycle, however rather than continuing with the last object when the last object has been displayed, the cycle continues with the first object.

Looping cycle: obj_1 -> obj_2 -> obj_i; obj_j -> obj_k; ...........

The growth cycle allows the objects to be accumulatively displayed.

Growth cycle: obj_1 -> obj_s + obj_{s+1} -> obj_s + obj_{s+1} + obj_{s+2} -> ...........

7) Camera Choreography

Besides manipulating data, animations can be created by keeping the data static and manipulating the camera (reference #2). This type of animation allows an animator to "fly" about the data and view it from various locations in world space. Zooming and panning techniques may be used to produce other views. Several different parameters may be modified for camera choreography (refer to Figure 11). The camera viewpoint or twist, which establishes the actual physical location and orientation of the camera in world space, may be modified. In addition, the reference point of the camera, the location where the camera is looking at, may be changed. When creating animations with the camera, start and end any motion slowly (slow-in & slow-out). In addition, jerky motions throughout the animation should be avoided.

An advantage of camera choreography is that the data does not have to be manipulated, it only needs to be viewed. For this reason the time required to produce an animation may be reduced considerably.

![Camera Choreography Diagram]

Figure 11: Camera Choreography

8) Other Animation Techniques

Besides using key frames and camera choreography, there are many other techniques that are available to enhance the visualization experience. In general these other techniques are used in conjunction with the key frame and camera choreography animations. They include parametric systems, P-curves, animating object attributes such as color and transparency, as well as introducing special objects used to visualize a range of phenomena. These special objects may be contour lines and sweeps; particles, streamlines and vectors; ribbons and streamers; as well as isosurfaces and cutting planes.

8.1) Parametric Systems

Parametric systems are an extension of the key framing technique. In some applications, while it is tedious to generate key frames, the 4x4 transformation matrix can be easily created by procedures (functions). In this case, an object would be linked to one or several motion functions, where the motion is determined by a set of time dependent
parameters. These parameters may be modified through an interactive user interface. A simple example of a parametrically based system would be the rotation of propellers of an aircraft. By specifying the revolutions per minute as a function of time, the rotational matrices required for the animation can automatically be calculated by the computer. It would be a very tedious job to create such an animation via key frames alone. Another significant aspect of this system, which cannot be achieved by key frames, is to incorporate some constraint into the system. For example, two colliding objects should not penetrate each other.

8.2) P-Curve
The P-curve is a parametric representation of the motion or any attribute associated with an object. Rather than using functions as mentioned in the parametric systems above, the P-curve establishes the animation via a graphical user interface. The positional coordinates of the motion x(t), y(t), and z(t) for instance are described by spline curves (see the 2-D analysis, Figure 12). By specifying the position (or attribute) on the curve at each frame, the animation is created. Obviously, the graphical user interface plays an important role in controlling animations. By editing the parametric curve, animations with different paths or attributes can be created. (For details see references #2, #5)

![Figure 12: P-Curve](image)

8.3) Animation Object Attributes (Color and Transparency)
Color is used extensively in scientific visualization to analyze the dynamic range and distinguish particular values of a data set. Colors are mapped to a specific value or a range of values which allow researchers to have a clearer understanding of the topology of their data. Transparency is used extensively to provide a view of data hidden behind data of another plane.

8.4) Animating Special Objects

8.4.1) Contour Lines and Sweeps
Contour lines are used to visualize the threshold of a range of data sets in a particular plane. Data values equal to user defined threshold settings are graphically linked via lines which may or may not be colored by the magnitude of the data. The contour lines may be shown either by themselves or with the remaining data of the plane that has already been colored. Contour sweeps can be used to demonstrate the progressive change in data by visualizing the contour lines of various data planes. If time varying
data is available, then these contour lines are set into motion depending on the dynamics of the data.

8.4.2) Particles, Streamlines, and Vectors
Particles and streamlines are used extensively in the visualization of computational fluid dynamics. To understand the path and the dynamics of the flow, particles are "injected" in the computed flow. Streamlines are obtained by connecting the paths along which the particles travel. Vectors may also be used to demonstrate the magnitude and direction of the flow. Vortices and other physical flow phenomena can clearly be shown using these techniques. Particles, streamlines, and vectors may be colored by the magnitude of the particular physical parameter to be displayed.

8.4.3) Ribbons and Streamers
Visualization using ribbons and streamers is very similar to visualizing particles and streamlines. The primary difference is ribbons and streamers produce a segmented surface representation of the flow.

8.4.4) Isosurfaces and Cutting Planes
These two techniques are used extensively in scientific visualization. Isosurfaces represent the three dimensional surface associated with a particular value in the data set. As an example, isosurfaces are used to show the volumetric surface of a particular pressure or temperature. Data hidden by an isosurface may be displayed by either making the isosurface transparent or by using cutting planes. Cutting planes allow a scientist to slice discrete planes within the isosurface exposing any hidden visualized data.

Advanced Animation Techniques

9) Soft Object Animation
Shape distortions in animations are generally used to highlight dynamic actions of a scene, such as objects experiencing an impact (reference #6, #7, #8). Generally the shape distortions are completely arbitrary and under the control of the animator. The exception to this is when shape distortions are created from physically based models, i.e., forces acting on an object which produce the distortions. Simply speaking, shape distortions can be achieved by either moving the vertices of the polygonal model or by modifying the control points of a representative parametric surface of the model. When changing vertices, the polygonal resolution of the model may constrain the nature of the deformation. Models with low polygonal resolution may experience shape degradation. A better method for generating shape deformation is to modify the control points of the parametric surface.

By changing the coefficients of the basis functions of the parametric surface, the functional description of the surface is altered. Bezier or B-spline patches are well suited for parametric surface representation. This method can produce deformations of any complexity while maintaining a smooth surface of the model.

10) Animation of Particle Systems
Many complex models may be animated by animating representative particles of the particular model (reference #9). Animating rain drops, the flow of water, a flock of birds or even a table cloth are among the typical applications. There are two different methods for animating particle systems, namely, the scripted and non-scripted systems. The latter provides for much more realistic physically based animations.

10.1) Scripted Particle Systems
In scripted systems the objects, which are represented by particles, may be animated by user defined scripts controlling the particles’ dynamics and appearance. A frame by frame generation of the animation would be accomplished by the following method:
First, new particles are generated and injected into the system. The number of particles released, \( N(t) \), may be computed by

\[
N(t) = M(t) + \text{rand}(r)V(t)
\]

where \( M(t) \) is a mean number of particles as a function of time. The product \( \text{rand}(r)V(t) \) would represent a random variable of variance, as a function of time. Each new particle would be assigned its own initial conditions such as position, velocity, direction, size, shape, color, transparency, and its lifetime. Any particle which exceeds its lifetime will be extinguished from the scene. The particles would move according to their user defined scripts based on dynamic constraints. At this point the particles can be rendered.

10.2) Non-scripted Particle Systems
During the last three years increased computational power as well as sophisticated computer graphics hardware has made it possible to produce very advanced animations using physically based modeling of particle systems. The motion of the particles is completely based on Newtonian mechanics. Various types of forces may act on these particles, which may either be unary forces (gravity, drag, etc.) that act independently on each particle, n-array forces (springs) that are applied to a fixed set of particles, or forces of spatial interaction (attraction, repulsion) that may act on all or pairs of particles. Mass-spring systems may be created that approximate the motion of rigid or elastic bodies as well as fluids. In this system the mass of a body may be lumped into a collection of mass points. The motion between these mass points may be approximated by a set of springs. By changing the spring constant one can modify the viscosity of a fluid or the elasticity of an object. Several papers have been written (reference #4) over the last three years on how to incorporate these techniques into computer graphics animation.

10.3) Behavioral Animation
Behavioral animations may be used to simulate biological systems (reference #10). Once again the concept is based on particle systems. With this method particles are no longer independent, they interact with each other to simulate various flocking mechanisms. The flocking of the entire system is controlled by global positions or global direction vectors. The flocking individuals are controlled by the following criteria. A flock mate will always avoid a collision with a nearby flock mate, while maintaining the closest proximity to the other flock mate. In addition a flock mate will try to maintain the velocity of the nearby flock mates. Very nice animations of a flying flock of birds have been created using this method.

Summary

11) Although its most common application is in the commercial entertainment industry, computer animation has become an important tool for visualizing complex and time dependent data in recent years. This paper details the animation fundamentals and provides an overview of most commonly used animation techniques in Scientific Visualization. Some techniques such as the key frame systems were discussed in more detail because of their underlying importance associated with animations. Due to increased computational power and sophisticated rendering devices, advances in animation techniques, such as physically based modeling, animation will play a more important role in simulation and visualization of the real physical world.
Acknowledgments

The authors owe the warmest thanks to Eric Hibbard, who read the text with meticulous care. His extensive and penetrating comments have been most important in improving the paper in all its aspects. The authors are very grateful to Randy Kaemmerer for commenting on the draft.

References

9) M. Kass, A. Witkin, D. Baraff and A. Barr, "An Introduction to Physically Based Modeling," 1993 Siggraph Course Notes 60, Anaheim, CA;

Appendix A
(Typical Hardware Components)

In general, to produce an animation, naturally some type of display device is required to produce the image on a monitor. In addition a device is required to convert the RGB signal(s) from the display device into a recordable signal like RS-170A. A frame controller is required to accept commands from the computer to cause one or more frames of the animation to be recorded. Finally, a recorder is used to record the image data on some media.

The basic components used at NASA Ames Research Center are listed as follows: The display device used is either a workstation or an X-terminal. A scan converter, such as a Lyon Lamb real time scan converter (RTC), converts the RGB signal to a recordable signal. A Lyon Lamb miniVAS controls the sequence of frames. A Betacam SP deck is used as the mastering deck, however, any recording device that support frame accurate recording may be used (e.g., U-matic SP, VHS, S-VHS, etc.).

Appendix B
(C Code)

/******************** Hierarchical Structures *******************/
/****
**Tree structure**

```c
struct node {
    /*
    .... node data
    */
    PIVOT_PTR node_pivot; /* Local pivot point of the node */
    MATRIX_PTR node_matrix; /* Actual drawing matrix */
    MATRIX_PTR original_mat; /* Will never be changed */
    MATRIX_PTR animate_mat; /* List of saved Keyframes */
    struct node *next;
    struct node *parent;
    struct h_node {
        int length;
        struct node *first, *last;
    } *leaves;
};
```

```c
typedef struct node NODE;
typedef struct h_node H_NODE;
typedef H_NODE *LEAVES;
typedef NODE *TREE;
```

**Linked list of picture structures** (forest of trees)

```c
struct forest_def {
    /*
    ... picture data
    */
    TREE tree_address; /* Pointer to a tree structure */
    Gen_matrix pict_mat; /* Camera transformation matrix */
    CAMERA_LIST camera; /* List of camera keyframes */
    struct forest_def *previous;
    struct forest_def *next;
};
```

```c
typedef struct forest_def FOREST_DEF;
typedef FOREST_DEF *FOREST;
```

............... **Initializing the Node Matrix** ***************

```c
/*
Create the Node Matrix. If an ARCgraph matrix exists, use it as a node matrix.
*/
void init_node_matrix ( TREE tree_node )
{
    if ( tree_node->original_mat != NULL )
        copy_matrix(*(tree_node->original_mat),*(tree_node->node_matrix));
    else
        copy_matrix( Identity, *(tree_node->node_matrix) );
} 
```
 Computing the World Transformation Matrices

Compute the world transformations

void world_transform()
{
    /*
    COMPUTE THE NODE MATRIX
    */
    pushmatrix();
    loadmatrix( Identity );
    /*
    translate the world pivot point to the world origin (0, 0, 0)
    */
    translate( active_node->node_pivot->wrld_piv_pnt[0],
               active_node->node_pivot->wrld_piv_pnt[1],
               active_node->node_pivot->wrld_piv_pnt[2] );
    /*
    incremental rotation about the desired axis of the world origin
    */
    if ( xrot_flag )
        rot( xrotval, 'x' );
    else if ( yrot_flag )
        rot( yrotval, 'y' );
    else if ( zrot_flag )
        rot( zrotval, 'z' );
    /*
    scale about the desired axis of the world origin
    */
    if ( xscale_flag )
        scale( 1.0 + xscaleval, 1.0, 1.0 );
    else if ( yscale_flag )
        scale( 1.0, 1.0 + yscaleval, 1.0 );
    else if ( zscale_flag )
        scale( 1.0, 1.0, 1.0 + zscaleval );
    else if ( xyzscale_flag )
        scale( 1.0 + xscaleval, 1.0 + yscaleval, 1.0 + zscaleval );
    /*
    translate back to the local origin
    */
    translate( -active_node->node_pivot->wrld_piv_pnt[0],
               -active_node->node_pivot->wrld_piv_pnt[1],
               -active_node->node_pivot->wrld_piv_pnt[2] );
    /*
    perform a local transformation
    */
    if ( translate_flag )
        translate( xval, yval, zoomval );
    /*
    multiply the old accumulated matrix to get the new resultant matrix
    */
    multmatrix( *(active_node->node_matrix) );
}
SET THE RESULTANT NODE MATRIX
*/
getmatrix( *(active_node->node_matrix) );
popmatrix();
}

/**************** Computing the Local Transformation Matrices **************/

/****
Compute the local transformations
****/
void local_transform()
{
  /*
   COMPUTE THE NODE MATRIX
   */
   /*
   multiply the local node matrix
   */
   pushmatrix();
   loadmatrix( *(active_node->node_matrix) );
   /*
   incremental rotation about the axis of the local pivot point
   */
   if ( xrot_flag )
     rot( xrotval, 'x' );
   else if ( yrot_flag )
     rot( yrotval, 'y' );
   else if ( zrot_flag )
     rot( zrotval, 'z' );
   /*
   incremental scale about the axis of the local pivot point
   */
   if ( xscale_flag )
     scale( 1.0 + xscaleval, 1.0, 1.0 );
   else if (yscale_flag )
     scale( 1.0, 1.0 + yscaleval, 1.0 );
   else if ( zscale_flag )
     scale( 1.0, 1.0, 1.0 + zscaleval );
   else if ( xyzscale_flag )
     scale( 1.0 + xscaleval, 1.0 + yscaleval, 1.0 + zscaleval );
   /*
   get the local node matrix
   */
   getmatrix( *(active_node->node_matrix) );
   popmatrix();
   /*
   incorporate the total linear translation
   */
   pushmatrix();
   loadmatrix( Identity );
   /*
   translate the local pivot point to the world origin (0, 0, 0)
   */
translate( active_node->node_pivot->pivot_pnt[0],
          active_node->node_pivot->pivot_pnt[1],
          active_node->node_pivot->pivot_pnt[2] );
/*
   multiply the local node matrix
*/
multmatrix( *(active_node->node_matrix) );
/*
   translate back to the local origin
*/
translate( -active_node->node_pivot->pivot_pnt[0],
          -active_node->node_pivot->pivot_pnt[1],
          -active_node->node_pivot->pivot_pnt[2] );
/*
   perform a local transformation
*/
if ( translate_flag )
  translate( xval, yval, zoomval );
/*
   get the resultant node matrix
*/
getmatrix( *(active_node->node_matrix) );
popmatrix();
}

/*************** Switching Between World and Local Space **************/

/****
   Toggle between transforming the objects about the world coordinate axes or about the
   objects' local coordinate axes.
*****/
void switch Coord System( int direction )
{
  /*
   From world to local: direction = -1,
   From local to world: direction = 1.
   */
pushmatrix();
loadmatrix( Identity );
translate( direction * active_node->node_pivot->pivot_pnt[0],
          direction * active_node->node_pivot->pivot_pnt[1],
          direction * active_node->node_pivot->pivot_pnt[2] );
multmatrix( *(active_node->node_matrix) );
translate( -1.0 * direction * active_node->node_pivot->pivot_pnt[0],
          -1.0 * direction * active_node->node_pivot->pivot_pnt[1],
          -1.0 * direction * active_node->node_pivot->pivot_pnt[2] );
getmatrix( *(active_node->node_matrix) );
popmatrix();
}

/******************** Displaying the Current Picture Node **************/

/****
   Display the entire current picture (scene)
void display_scene()
{
  /*
   Clean the matrix stack
   */
  pushmatrix();
  /*
   Make the viewing volume
   */
  make_viewing_volume();
  /*
   Set the lighting environments
   */
  set_lighting_environment();
  /*
   Apply global translation and scaling
   */
  translate( cur_picture->pict_mat.xpos, cur_picture->pict_mat.ypos, -32.0 );
  scale( cur_picture->pict_mat.global_scale[0],
         cur_picture->pict_mat.global_scale[1],
         cur_picture->pict_mat.global_scale[2] );
  /*
   Display each node of the current picture with its appropriate transformations.
   */
  display_tree( cur_picture->tree_address );
  /*
   Restore the original matrix
   */
  popmatrix();
}

******/ Displaying all of the Nodes Associated with the Current Picture Node ******/

/**
   Display the tree nodes using a preorder tree traversal.
***/
void display_tree( TREE tree_node )
{
  TREE      temp;

  /*
   Push the node matrix and attributes
   */
  pushmatrix();
  pushattributes();
  /*
   Apply current node transformations if present
   */
  if ( tree_node->node_matrix )
    multmatrix( *(tree_node->node_matrix) );
  /*
   Draw the node object of the tree with its associated attributes
   */
draw_the_node( tree_node );
/*
   Pretraverse the tree
*/
if ( tree_node->leaves != NULL )
{
    temp = tree_node->leaves->first;
    while( temp != NULL )
    {
      display_tree( temp );
      temp = temp->next;
    }
}
/*
   Restore the original node matrix and attributes
*/
popmatrix();
popattributes();

/******* Computing the Animation Inbetweens for the Saved Keyframes ******/

/* Author: Kenneth Hu
 * Date: 11/1/93
 */
#include <stdio.h>
define X 0
define Y 1
define Z 2
define W 3
define EPSILON 1.0e-8
#define HALFPI 1.570796326794895
typedef float Tmat[4][4];
typedef float Vec[3];
typedef float Quat[4];

/***
   Extract the scaling parameters from the general 4x4 transformation matrix.
   Input is the 4x4 matrix m which contains translational, rotational, and scaling
   parameters.
   Output is a 4x4 matrix only containing the rotations and translations. In addition
   the scaling vector and the shear parameters are obtained.

   void normalize( Tmat m, Tmat mat, Vec scale, Vec shear)
{
  float sxy, sxz, syz;
  int i;

  scale[0] = fsqrt ( m[0][0]*m[0][0]+m[0][1]*m[0][1]+m[0][2]*m[0][2]);
  if ( scale[0] < EPSILON )
  {
    printf("WARNING: singular matrix, can not be processed\n");
    return;
  }
  for ( i=0; i<3; i++) mat[0][i] = m[0][i]/ scale[0];

  */
sxy = 0.0;
/*
   Perform a dot product on the first row and the second row */
for (i=0; i<3; i++)
   sxy += (mat[0][i] * mat[1][i]);
/*
   Subtract first row component from second row */
for (i=0; i<3; i++)
   mat[1][i] = mat[1][i] - sxy * mat[0][i];
   scale[1] = fsqrt(mat[1][0]*mat[1][0]+mat[1][1]*mat[1][1]+mat[1][2]*mat[1][2]);
if (scale[1] < EPSILON) {
   printf("WARNING: singular matrix, can not be processed\n");
   return;
}
for (i=0; i<3; i++)
   mat[1][i] /= scale[1];
   shear[0] = sxy/scale[1];
sxz = 0.0;
syz = 0.0;
/*
   Perform a dot product on the first row and the third row */
for (i=0; i<3; i++)
   sxz += (mat[0][i] * mat[2][i]);
/*
   Perform a dot product on the second row and the third row */
for (i=0; i<3; i++)
   syz += (mat[1][i] * mat[2][i]);
/*
   Make the third row perpendicular to the first two rows */
for (i=0; i<3; i++)
   mat[2][i] = m2[i] - sxz * mat[0][i] - syz * mat[1][i];
if (scale[2] < EPSILON) {
   printf("WARNING: singular matrix, can not be processed\n");
   return;
}
/*
   Normalize the third row */
for (i=0; i<3; i++)
   mat[2][i] /= scale[2];
/*
   Scale the shear factor */
shear[1] = sxz/scale[2];
shear[2] = syz/scale[2];

for (i=0; i<4; i++){
   mat[i][3] = m[i][3];
   mat[3][i] = m[3][i];
}
} 

/**
 * Compute the quaternions from the normalized rotational matrix
 */

void QuatFromMatrix( Tmat mat, Quat q )
{
    int i, j, k;
    float tr, s;
    tr = mat[0][0] + mat[1][1] + mat[2][2];
    if ( tr > 0.0 ) {
        s = fsqrt( tr + 1.0);
        q[W] = s * 0.5;
        s = 0.5/s;
        q[X] = ( mat[1][2] - mat[2][1] ) * s;
        q[Y] = ( mat[2][0] - mat[0][2] ) * s;
        q[Z] = ( mat[0][1] - mat[1][0] ) * s;
    }
    else {
        i = X;
        if ( mat[Y][Y] > mat[X][X] ) i = Y;
        if ( mat[Z][Z] > mat[i][i] ) i = Z;
        j = nxt[i]; k = nxt[j];
        s = fsqrt((mat[i][i] - (mat[j][j]+mat[k][k])) + 1.0);
        if ( i == X ) {
            q[X] = s * 0.5;
            s = 0.5/s;
            q[Y] = ( mat[0][1] + mat[1][0] ) * s;
            q[Z] = ( mat[0][2] + mat[2][0] ) * s;
        }
        else if ( i == Y ) {
            q[Y] = s * 0.5;
            s = 0.5/s;
            q[W] = ( mat[2][0] - mat[0][2] ) * s;
            q[Z] = ( mat[1][2] + mat[2][1] ) * s;
            q[X] = ( mat[0][1] + mat[1][0] ) * s;
        }
        else {
            q[Z] = s * 0.5;
            s = 0.5/s;
            q[W] = ( mat[0][1] - mat[1][0] ) * s;
            q[X] = ( mat[2][0] + mat[0][2] ) * s;
            q[Y] = ( mat[2][1] + mat[1][2] ) * s;
        }
    }
}

/**
 * Compute the inbetweens for two keyframes ( input1 & input2 ) for "steps"
 * inbetweens. Linear interpolation will be performed on the scaling and
 * translational vectors as well as the shear parameters. Spherical linear
 * interpolation will be performed on the quaternions. After all interpolations
 * are completed the components
 */

203
of each inbetween is combined to a respective 4x4 matrix used in the animation.

```c
void Interpolate(Tmat m1, Vec scale1, Vec shear1, Quat q1, /* input 1 */
    Tmat m2, Vec scale2, Vec shear2, Quat q2, /* input 2 */
    Tmat *matrix, int steps) /* output */
{
    int i, j;
    float delta, trans[3], scl[3], shear[3], qt[4];
    delta = 1.0/(float)steps;
    t = delta;
    for (i=0; i<steps; i++) {
        for (j=0; j<3; j++)
            trans[j] = linear(m1[3][j], m2[3][j], t);
        for (j=0; j<3; j++) {
            scl[j] = linear(scale1[j], scale2[j], t);
            shear[j] = linear(shear1[j], shear2[j], t);
        }
        slerp(q1,q2,t, qt);
        QuatToMatrix(qt, matrix[i]);
    /*
      Combine translation, scaling, shear, and rotation to one matrix
    */
    for (j=0; j<3; j++) {
        matrix[i][3][j] = trans[j];
        matrix[i][2][j] = shear[1]*scl[2]*matrix[i][0][j] +
                         shear[2]*scl[2]*matrix[i][1][j] +
                         scl[2]*matrix[i][2][j];
        matrix[i][1][j] = shear[0]*scl[1]*matrix[i][0][j] +
                         scl[1]*matrix[i][1][j];
        matrix[i][0][j] *= scl[0];
    }
    t += delta;
}

/***
 Perform the spherical linear interpolation for the quaternion keyframes to produce
 the quaternion inbetween. P and q are input quaternions, t is the distance from
 p (between 0 and 1), qt is the interpolated quaternion.

 void slerp( Quat p, Quat q, float t, Quat qt)
{
    float omega, cosom, sinom, sclp, sclq;
    int i;
    float temp1, temp2, tempq1[4], tempq2[4];
    /*
      Determine the shortest distance (arclength) on the sphere, use either +q or -q
    */
    temp1 = temp2 = 0.0;
    for (i=0; i<4; i++) {
        tempq1[i] = p[i] - q[i];
        tempq2[i] = p[i] + q[i];
        temp1 = temp1+(tempq1[i] * tempq1[i]);
        temp2 = temp2+(tempq2[i] * tempq2[i]);
    }
    omega = acos(temp1);
if (temp2 < temp1) {
    for (i=0; i<4; i++) q[i] = -q[i];
}

if ((1.0+cosom) > EPSILON) {
    if ((1.0 - cosom) > EPSILON) {
        omega = facos(cosom);
        sinom = fsin(omega);
        sclp = fsin((1.0-t)*omega)/sinom;
        sclq = fsin(t*omega)/sinom;
    } else {
        sclp = 1.0 - t;
        sclq = t;
    }
    for (i=0; i<4; i++) qt[i] = sclp*p[i] + sclq*q[i];
} else {
    qt[X] = -p[Y]; qt[Y] = p[X];
    sclp = fsin((1.0-t)*HALFPI);
    sclq = fsin(t*HALFPI);
    for (i=0; i<4; i++) qt[i] = sclp*p[i] + sclq * qt[i];
}

intval intval

Linear interpolation for the translation, scale, and shear parameters.

float linear (float p, float q, float t)
{
    return (p+(q-p)*t);
}

Convert the quaternion inbetweenes back to the required rotational matrix

void QuatToMatrix (Quat q, Tmat matrix)
{
    float s, wx, xx, yy, ys, zs, wy, wz, xy, xz, yz, zz;
    s = 2.0/(q[X]*q[X]+q[Y]*q[Y]+q[Z]*q[Z]+q[W]*q[W]);
    xs = q[X]*s; ys = q[Y]*s; zs = q[Z]*s;
    wx = q[W]*xs; wy = q[W]*ys; wz = q[W]*zs;
    xx = q[X]*xs; xy = q[X]*ys; xz = q[X]*zs;
    yy = q[Y]*ys; yz = q[Y]*zs; zz = q[Z]*zs;

    matrix[0][0] = 1.0 - (yy + zz);
    matrix[0][1] = xy + wz; matrix[0][2] = xz - wy;

    matrix[1][0] = xy - wz; matrix[1][1] = 1.0 - (xx+zz);
    matrix[1][2] = yz + wx;

    matrix[2][0] = xz + wy; matrix[2][1] = yz - wx;

205
matrix[2][2] = 1.0 - (xx + yy);

matrix[3][3] = 1.0;
Virtual Reality - A General Overview

Michael E. Bauer
Sterling Software, Palo Alto, CA, U.S.A.

Abstract

During the past few years, Virtual Reality (VR) has transitioned from a subject for basic research to a concept that is being employed in many scientific and engineering disciplines (e.g., architecture, computational fluid dynamics, planetary exploration, medical research, etc.). A major reason for this transition is that VR technology has been teamed with advanced computer graphics technology to create new scientific visualization tools, which give scientists and engineers the opportunity to immerse themselves in their data and afford them a new and effective perspective of this data.

This paper gives an overview of VR systems by focusing on VR hardware technologies as well as design approaches of successful virtual worlds. The concepts in this paper were obtained from the 1993 SIGGRAPH course on Applied Virtual Reality, "Virtual Reality Overview", held in Anaheim, California, and taught by Carolina Cruz-Neira of the Electronic Visualization Laboratory, University of Illinois at Chicago.

Introduction

What is virtual reality? There are many different factors that characterize the definition of virtual reality. The following characteristics are generally accepted throughout the industry:

• Virtual reality has to perform its function in real time; the computations associated with a changing viewpoint as well as the changes in the model have to take place in real time. Any significant lag time will have a detrimental effect on the VR sensation.

• Virtual reality space exists in its own 3D world which may be either abstract or concrete. The scenes themselves are limited only by the bounds of one's imagination.

• Virtual reality should provide the illusion of immersion. A participant should feel the sensation of being completely surrounded by the virtual scene.

• Virtual reality should provide and encourage interaction. A participant should be able to interact with the virtual world via some input devices (e.g., gesture recognition, voice recognition, applying forces, etc.). Objects that are manipulated by these input devices should respond with some sort of reaction.

• Virtual reality should provide feedback to the participant. This may be accomplished by either providing visual, sound, or force feedbacks.

To produce an effective virtual reality system, many of the above mentioned characteristics should be realized. The following discussions on VR hardware technologies and VR design considerations should assist a user to accomplish this task.

VR Hardware Technologies

1) A virtual reality system requires many different hardware components, which control the participant's sensory inputs (e.g., visual, audio, tactile, etc.), track the participant's movements, and receives and interprets the participant's input into the system. The purpose as well as advantages and disadvantages of each class of device is addressed in this section.
1.1) Visual Displays

The visual display of the VR system is the most important component from the participant's perspective. It provides the visual stimuli, which determines the degree of immersion that can be achieved. Currently, there are many devices that can function in this capacity; however, they can be grouped into four categories: monitors, head mounted display systems (HMDs), booms, and projection systems.

1.1.1) Most monitors employed as the visual display include some form of stereographics support. Typically, they are the least expensive option for a visual display because they come standard on many graphics workstations or they are an inexpensive option. There are two different stereo viewing systems available for monitors (reference #1). One system consists of stereo eye glasses which contain liquid crystal electronic shutter lenses which, in turn, select the appropriate image for the appropriate eye. The light weight glasses are linked to an infrared emitter which controls the shutter frequency. The other system uses a screen-sized liquid crystal electronic shutter that presents different polarized states for left and right eye images from the CRT. The shutter is controlled directly through the monitor. The user is required to wear only light weight polarized glasses. Due to the larger liquid crystals, this system is usually more expensive.

The advantage of using monitors is that the stereoscopic scene is displayed with a full workstation resolution requiring only the space of a desktop. It is a minimal-invasive system, i.e., light weight glasses need to be worn. In addition, small groups of people may experience the virtual scene simultaneously from the same monitor. Simple interaction with external physical interfaces is maintained. The primary weakness of a monitor system is that it provides the least degree of immersion. This is due to the small angle of view of the scene, and a non-surrounding view as well. There is no superimposition of the real world and the virtual space; essentially the virtual scene is inside of a box.

1.1.2) Head Mounted Display Systems (HMDs)

HMDs use a pair of liquid crystal displays (LCDs) as their screens which are placed directly in front of the users' eyes. In addition the HMDs use special optics to attain a wide field of view. The combination of the wide field optics and the screen's close proximity to the eyes, produces a large angle of view as well as a panoramic view of the virtual scene. Since they are very light weight and flat they can be easily mounted on a users head. This provides for many degrees of freedom in moving about in the real world allowing for good navigation through large databases. The degree of immersion is currently higher than monitors. The potential for increased immersion, once some of the disadvantages listed below are solved, is one of the main advantages of HMDs.

The disadvantages of HMDs can be summarized as follows. The LCDs have a very low resolution compared to a monitor, especially when they are in full color. The LCDs display quality is weak due to problems with contrast and brightness. Since they are worn on a user's head they can be highly invasive. The HMDs cannot be shared at the same time unless the same video signal is sent to multiple HMDs. Without half-silvered mirrors as part of the screen, HMDs provide no superimposition of the real and virtual world, i.e., a user is visually separated from the real world. For this reason physical interfaces as well as hand and body parts need to be reconstructed in the virtual environment. Nevertheless, HMDs are one of the most widely used visual displays in virtual reality systems.

1.1.3) Booms

Booms primarily use monitors (CRTs) as the display technology. The booms' display technology differs from the monitors display technology, described above, in that booms use two small monitors, one for each eye, rather than one larger stand alone monitor. CRTs provide a higher resolution than LCDs and have a better display quality. The presence of high-power, high-frequency electromagnetic fields and a heavy weight make CRTs unsuited for mounting on a user's head. For this reason they are mounted on a
boom comprised of several arms connected via linkages. As with LCDs, the CRTs incorporate wide field optics to increase the field of view. Since booms are not physically mounted to the head, they have the advantage of being only moderately invasive. A user can easily place or push away the CRT from the eyes. This allows for tolerable interaction with physical interfaces. Due to the simplistic design of the boom and the many degrees of freedom it provides, navigation through large databases is good.

The boom's disadvantages are similar to those of the HMDs. Only one person can realize the virtual experience at a time, unless the video signal is shared with other devices. Once the CRT is in front of the eyes, physical interfaces and body parts need to be reproduced in the virtual world, as with HMDs, there is no superimposition of the real and virtual worlds.

1.1.4) Projection Systems

With projection systems the virtual scene is projected on large screens which completely surround the view. An example of such a projection system is the CAVE (Audio Visual Experience Automatic Virtual Environment) developed at the Electronic Visualization Laboratory at the University of Illinois (reference #2). The projection systems have several advantages. A user may have a 360-degree top and bottom view of the virtual scene. Since the stereoscopic images are projected on surrounding screens occupying a large space, many people can share the experience simply by wearing polarized glasses. The projection systems are the least invasive of all the display systems since only a tracker is connected to the body. The high video resolution, panoramic view provides a strong sense of immersion. This makes good navigation of databases possible. The use of physical interfaces is very easy since there is full superimposition of the virtual and real spaces. In addition there is no need to regenerate the hand and body parts.

The projection systems do have some disadvantages. Projection systems are required to be placed in a large room due to the large surrounding screens. Supporting the synchronization and generation of the stereoscopic images for all the screens is very difficult and expensive. Since there is full superimposition of the virtual and real spaces, stereo violations may occur. As an example, the user's hands may have the same spatial location of a computer generated solid object, hence appearing to be inside of the object.

1.2) Tracking Systems

Current tracking systems are used to measure the user's head position and orientation as well as the position of other body parts such as the hands and fingers. Tracking the head position and orientation allows for correct computation of the view of the world from the users point of view. This is critical to obtaining a good illusion of immersion. Tracking the other body parts allows for interaction and control of the virtual world. Several methods employed in tracking systems (electromagnetic, mechanical, acoustical, optical, and inertial systems) that will be discussed next.

1.2.1) Electromagnetic Tracking Systems

This type of tracking system uses a transmitter or source which emits electromagnetic fields along three orthogonal axes. The position and orientation of one or more sensors with respect to the source are reported. Examples of such tracking systems are the Polhemus Isotrack as well as the Ascension Flock of Birds. Electromagnetic tracking systems have some problems. One problem is the interference from other electromagnetic fields. Trying to track the head position using a HMD, which may produce its own electromagnetic field, could result in erroneous information if care is not taken.

The primary problem with electromagnetic tracking systems is the latency and accuracy issues. Latency is associated with the elapsed time between change in position of a sensor and the time required for the change to be reported. To have effective tracking, the latency should be less than 0.1 seconds. Accuracy is extremely sensitive to the presence of metal and other electromagnetic fields as mentioned above. The advantage of the
electromagnetic tracking systems is that the sensors may be moved around freely. In addition, object obstructions (objects placed between the source and sensors) do not interfere with the electromagnetic signal. Electromagnetic Tracking Systems are fairly inexpensive compared to other tracking systems.

1.2.2) Mechanical Tracking Systems
These tracking systems are usually associated with mechanical linkages. A good example of such a device is the boom discussed above. The position and orientation of the user's view point are computed from joint angles. This technique produces very small latency and very high accuracy. The problems associated with the boom are the range restriction due to the mechanical linkages. The price associated with accurate mechanical tracking systems can be very high.

1.2.3) Optical Tracking Systems
This tracking system, which is still in an experimental stage, uses light emitting diodes (LEDs), video cameras, and image processing techniques to determine the position and orientation of a user. Fixed cameras may either track the LEDs mounted on the user or head mounted cameras track fixed LEDs in the surrounding space. The positions of the LEDs on the video screens can be established using signal processing techniques. These positions are then used to determine the positions of the LEDs in the actual 3D space. This technique requires a clear line of sight between the LEDs and the cameras, hence object obstructions are critical. Optical tracking systems which are still in the experimental stages, are rather expensive due to the electronic equipment required.

1.2.4) Acoustic Tracking Systems
Acoustic tracking systems use ultrasonic sound which is picked up by several microphones arranged in a triangular fashion. The position and orientation are calculated by the difference in elapsed time that is required for the sound pulses to reach the microphones. This type of tracking system has the same latency problem as magnetic trackers; the acoustic tracking systems, however, are not affected by metal interference. Since sound waves are affected by the media through which they travel, object obstructions will affect the sound pulses. Examples of acoustic tracking systems are Logitech Mouse and Mattel Powerglove, which are less expensive tracking systems.

1.2.5) Inertial Tracking Systems
There are two types of inertial tracking systems: gyroscopes and accelerometers. Both of these systems are in experimental stages. Gyroscopes measure the angles of pitch, yaw, and roll. This system is based on the principle of conservation of momentum. Accelerometers detect the acceleration of a tracked object, and, through integration, the position can be determined. The advantage of inertial tracking systems is they do not require a separate source, so their range is limited only by the length of their cord.

1.3) Computation Systems
The computer hardware is the "heart" of a virtual reality system. The computer controls all input and output devices, determines the state of the environment as well as the stereoscopic viewpoints associated with the left and right views of the virtual scene. Last, but not least, the computer also renders all of the objects of the virtual world. All of these functions have to be synchronized on a frame by frame basis at a video rate no less than ten frames per second (i.e., the graphics update rate, which includes the computation time, time to read the input devices, and time to produce the stereo rendering, is greater than ten frames per second). To create an effective virtual reality system, a multi-processor computer is often used, where each processor is assigned one of the following tasks: handling the operating system, processing the computations, performing the rendering, and controlling the input and output devices. Some VR systems employ a dual headed graphics
system (two independent graphics pipelines) or a pair of graphics workstations working in tandem to produce the stereographic image pairs. If the virtual scene requires complex computations, these can be performed on a separate supercomputer. The computed data as well as the state of the virtual environment can be passed to and from the workstation and supercomputer via a fast network. Suitable networks may be either the UltraNet or HIPPI networks.

1.4) Haptic Devices
Haptic devices allow a user to manually explore the virtual environment. By measuring the position, orientation, and forces of the user's body parts, virtual objects may be easily manipulated. Force and torque feedback as well as tactile devices provide a user with the sensation that the objects are real. In addition, devices that can produce the stimulus of temperature highlight the illusion of reality.

1.5) Audio Systems
Audio systems can provide excellent interaction with the virtual environment. Voice recognition for input commands and voice as well as sound output synthesizers create feedback about the virtual world and are powerful interaction tools. The output synthesizers may provide two different sound cues: monaural, and spatially localized sounds. Monaural sounds can be used to indicate events or user feedback on actions such as glove gesture recognition, when objects have been grabbed, or display states of the environment. Spatially localized sounds are used to give individual sounds a spatial location. Assigning these sounds to objects in the virtual environment can simplify the task of localizing objects.

Virtual Environment Design Considerations

2.0) There are many issues to consider when designing an effective virtual reality system. They include immersion, presence, combining real world objects with virtual objects, degree of intrusiveness of the interface, physical and audio feedback, navigation and control, as well as successive refinement or level of detail, which will be discussed below.

2.1) Immersion
An important aspect of a VR environment is that the participant feels completely surrounded by or immersed in the virtual world. This virtual world may be generated dynamically by a computer graphics system or scripted from pre-computed images on a video disc player. The key to immersion is in the quality and quantity of visual cues provided to the participant. Currently many of the human factors issues involved with creating a good sense of immersion are still unknown. This is a subject of ongoing research at the NASA Ames Research Center. From experience, researchers have learned that obtaining a good sense of immersion depends on the performance of the virtual reality system. As a rule of thumb, and as mentioned earlier, the graphics update rate should be greater than ten frames per second. Furthermore, the latency between the participant's actions and the instances at which these actions are presented in the scene should be less than one tenth of a second.

2.2) Presence
Presence is the sensation that objects in the VR scene are "really out there in space". Two properties dominate in the field of VR to provide computer graphics scenes with a sense of presence. These are depth cues in the virtual scene as well as behavior and interactivity of the virtual objects. There are several parameters that produce stimulating depth cues. Perspective rendering simulates the view as we see it in the real world. Objects that are farther away appear to be smaller, two parallel lines converge to a point at the horizon. Stereoscopy creates a strong sense of depth perception. Two stereoscopic
images of the virtual scene are created by rendering the images from two different perspective points of view. The disparity between the two images is interpreted by the brain to create the sensation of depth. It should be noted that stereo is most effective when objects are within a ten meter proximity. Objects that are farther away cease to provide stereo depth cues since the disparity between the two objects is significantly diminished.

Head motion parallax produces strong depth cues as well. As the viewpoint of the scene changes, the brain receives images from different viewpoints at infinitesimal time intervals, thus inducing perceived disparity. In addition, objects that are closer hide the view of objects that are farther away. Changing the location of light sources to highlight features of objects provides additional depth cues. Depth cues can also be obtained by introducing shadows into the scene. Real shadows are very expensive to render; their approximations can, however, be rendered at lesser cost while still providing good depth information. Atmospheric blurring is used to create the impression that objects are far off in the distance.

Providing interaction with an object, such as touching an object and having it respond to the person's actions, certainly creates a true sense of presence. Assigning a behavior to the objects, particularly in the response to a participant's interactions, as well as objects possessing their own behavior and the capability of self-generated motion also increases the sense of presence.

2.3) Combining Real World Objects with Virtual Objects

When designing a virtual reality system, it should be determined in advance whether a user needs to operate specialized devices in the real world. It may be necessary to see the devices as well. In this case there are several options. The display device could provide complete superimposition of the real and virtual worlds, such as the projection system. As an alternative, half-silvered mirrors could be used in a head mounted display to see the real and virtual worlds simultaneously. Using a desktop monitor allows a user to have full interaction of the real world while sacrificing the sense of full immersion. To maintain the full sense of immersion, the operating devices could be reproduced in the virtual world to facilitate interaction with virtual operating devices. In this case, the user's body parts would need to be reproduced as well.

2.4) Degree of Intrusiveness of the Interface

The user's interaction that is required with the virtual world affects the selection of the user interface. Some user interfaces require a participant to wear encumbering devices and attachments which create unacceptable motion restrictions. It may be necessary to minimize intrusion by sacrificing other features such as full immersion.

2.5) Physical Feedback

Currently physical feedback is created by using force/torque controllers based on sensor technology used to measure the forces and torques to control robotics. This technique significantly increases the level of immersion. Unfortunately the current understanding of physical feedback is still very limited. There are two major areas of research in physical feedback. These can be described as force and tactile feedback.

Force feedback relates to the forces acting on the muscles and joints of the participant. Caution must be exercised in providing reasonable force feedback to guarantee the participant's safety.

Tactile feedback can provide the sensation of touch, i.e., forces acting on the participant's skin as well as the sensation of temperature.

2.6) Audio Feedback

There are several methodologies to incorporate audio feedback into a virtual environment. As discussed above in the section on audio systems, there are monaural sounds and spatially localized sounds. Audio feedback of spatially localized sounds may
be used as a navigational tool by using audio as a clue to orient users in the virtual world. It may also be used as a localization tool where sounds are associated with objects or a particular location in the virtual space. The monaural sounds can be used as a sonification tool, thus transforming numerical information into sound. In addition monaural sounds are useful as an interaction tool, using audio for input and output such as voice recognition and speech synthesis, or providing sound feedback for actions. Voice commands used for environment control can free the user's hands for more essential tasks such as object manipulation.

2.7) Navigation and Control

Navigation within the virtual world is an important aspect of the virtual experience and it has an affect on the participant's immersion. Of the many different navigation techniques implemented for virtual environments, the most common is to "point and fly" through the virtual space. With this method a participant sets the desired direction of travel via an input device such as a mouse or the DataGlove (produced by VPL Research). Additional interaction with the input device activates the "flying" motion. By scaling the size of the virtual world a user can easily travel through a macroscopic landscape or the microscopic world of atoms. Tele-transport is also a very useful navigation method. By stating the location (e.g., next room) via voice control for instance, a user can automatically be tele-transported to the specified virtual location. A graphical interface may be used by displaying two or three dimensional maps in the virtual world to provide the capability of selecting a location on the map and instantly placing the participant at the respective location in the virtual world. Other exotic navigation devices such as bicycling on a stationary bicycle, or walking on a steerable treadmill, where both actions occur in the real world, have been successfully employed. In any case, if a participant is allowed to walk around freely in the real world without a clue of object obstructions (e.g., bumping into walls, tripping over cables), collision control must be provided.

Manipulating the parameters which control the state of the environment, such as changing the light settings, or manipulating an object's position and orientation in the environment, require other user interfaces. Pull down menus or 3D widgets in the virtual space have proven useful.

2.8) Successive Refinement or Level of Image Detail

Scenes in a virtual reality environment may be very complex and require much time to render. Such complex images may prevent the virtual environment from running within the minimal graphics update rate of ten frames per second. In this case, the use of successive refinement (images are rendered in greater detail only when there is no motion), is recommended. As soon as the participant moves his viewpoint, a much lower resolution image is displayed to guarantee real-time update rates.

Minimizing the complexity of the image to assure high update rates can be accomplished in several ways as discussed below.

2.8.1) Visibility Culling

A virtual reality scene may be comprised of a large virtual space containing many virtual objects. At any point in time a user may only see a fraction of this entire space in the viewing frustum (3D scene displayed to the user). There is no need to process the data (pass any of this data through the graphics pipeline), that is outside of the viewing frustum. This selective processing is called visibility culling. Visibility culling may also be extended to objects that are in the viewing frustum and are hidden behind other objects, e.g., objects hidden behind a hill or building. There are several papers published that cover visibility culling extensively (reference #3-#7).
2.8.2) Texture Mapping

Texture mapping can provide rich scenery with complex textures on simple geometry. With texture mapping any complex 2D digitized image, e.g., a tree, may be mapped onto a simple geometrical object such as a polygon. To render a tree with polygons would require much computational time. However, rendering the 2D image of the tree mapped on simple polygons can be accomplished in a fraction of the time using hardware texture mapping available in most medium- and high-end graphics workstations.

Texture mapping can also be used to animate a sequence in a virtual world, e.g., displaying an explosion or flames. Presenting this type of animation in a virtual environment using standard rendering techniques (e.g., particle animation and polygonal rendering) would be very difficult to accomplish within the desired frame rate. A good solution to this problem is to display a series of preprocessed animated textures of the animation on a simple polygon.

2.8.3) Level of Detail (LOD) Modeling

The concept of level of detail modeling suggests that objects only be modeled at their necessary resolution. For instance, objects that are closer to the viewer should be rendered with more detail than objects that are further away. There are many different schemes to provide LOD modeling, some of which are very specific to the virtual scene. A more general scheme is described as follows. LOD models of an object may be switched based on the displayed object’s size. The closer a particular object is to the viewer, the larger it appears to be. This size appearance can be used to switch to different LOD models. This technique has serious problems in that it introduces visual "popping", i.e., when the LOD changes a visual artifact is produced. One method to reduce popping is to blend both models together for a period of time. This however, does not altogether eliminate visual popping. A more appropriate method to produce the best visual effect, is to incorporate a morphing technique (reference #8, #9, #10). This technique takes the vertices of one LOD model and displaces them over time to coincide in position with corresponding vertices on the next higher LOD model, with color and attributes similarly interpolated. A specific LOD management tool (reference #11) was used at NASA Ames Research Center to accommodate the large databases associated with planetary terrain data.

2.8.4) Load Management

The objective with load management is to maintain an even load balance on the VR rendering system. In designing a virtual scene, one should reduce heterogeneously distributed scenery. Frequently this is very difficult to do, due to the nature of the scene. To maintain the desired frame rate, one could set high and low threshold values as a percentage of this frame rate. If the frame rate exceeds the high threshold value, the system is stressed and a lower LOD model of the scene would be displayed. Once the frame rate drops below the low threshold value, a higher LOD model could replace the lower LOD model. Such a technique is discussed in reference #11.

2.9) Shared Virtual Environments

Shared virtual environments can play a significant role in virtual environments since multiple users may share and control a VR environment; scientist may exchange knowledge and ideas using shared data. Generally a shared virtual environment has a host machine which is a single computational platform. It maintains the state of the environment as well as providing a large storage space. Graphics workstations which handle all of the graphics and communicate with the host machine via a high speed network (UltraNet or HIPPI). Additional information on shared virtual environments is available in the upcoming paper written by Steve Bryson.
Summary

Virtual Reality is evolving to provide a new paradigm of how we view and interact with data. It is a serious tool that can now be applied to many disciplines such as scientific and information visualization, architecture, engineering, design and medical applications, remote presence, learning and training, leisure, sports, health, telecommunications, entertainment, and art and advertising. Several books (reference #12-#14) are available on VR to obtain detailed information on VR systems. Currently the state of the art of VR is: "it is almost here"; it suffers from slow frame rates and lag time, poor display technology, poor registration with real world, tedious model-building, and various problems with the interface technology. With many of the technological developments in the hardware being fueled by the entertainment industry in VR, many of the limitations associated with current VR systems will be surmounted within the next few years and VR will become an overwhelming experience as well as a powerful tool to visualize and interact with information.

Acknowledgments

The author thanks Carolina Cruz-Neira for organizing and teaching (amongst other speakers) the course on Applied Virtual Reality taught at SIGGRAPH '93, Anaheim, California. The material presented at the course provided the foundation for this overview paper.

The author is very thankful to Randy Kaemmerer, Glenn Deardorff, Margaret Bradshaw and Eric Hibbard for commenting on the drafts, which resulted in an improvement of the paper in many aspects.

References


Virtual Reality in Scientific Visualization

Steve Bryson†


Abstract

The use of virtual environments in scientific visualization is discussed. Successful examples are surveyed in depth. Lessons about the usefulness and applicability of virtual environments to scientific visualization are drawn. Lessons learned from these applications for the development of virtual environments are also drawn. Difficulties in the application of virtual environments to scientific visualization are discussed. Problems encountered in virtual reality implementations of scientific visualization systems are briefly surveyed. Possible futures are briefly examined.

Introduction

1) Scientific visualization is the use of computer graphics in the investigation of scientific phenomena [1][2]. Some investigations involve large amounts of data arising from observation or from numerical simulation. Other problems, such as the structure of an exact solution in general relativity, are understood in principle but not in detail. Computer graphics assists the researcher's understanding of the qualitative structure of phenomena by drawing pictures which can be obtained in no other way. Interactive computer graphics, which allow the real-time control over how the graphics is generated, further enhances the researcher's ability to explore a phenomenon through computers. In conventional real-time computer graphics systems, a mouse and keyboard can be used to control, for example, the initial position of a streamline in a fluid flow. When the phenomenon under study is three-dimensional, the display is projected onto the two-dimensional display screen and the two-dimensional mouse movements are mapped into three-dimensional control. The mouse typically controls both the view point of the projection and the position of the objects in the view. Virtual environments, or virtual reality, provide a fully three-dimensional interface for both the display and control of interactive computer graphics [3]. A wide-field-of-view stereoscopic head-tracked display system presents a compelling illusion of a three-dimensional world generated by the computer graphics. The researcher feels immersed in this world, which is populated by computer generated objects which appear and behave as if they were real (figure 1). This three-dimensional display provides many of the depth cues which we use in the real world, such as binocular disparity and head-motion parallax, providing a display of three-dimensional structures that overcome many of the ambiguities that occur on two-dimensional screens. The display device tracks the user's head and controls the point of view of the computer generated scene. Using an instrumented glove, the researcher can reach out and directly manipulate virtual objects' position and orientation in three-dimensions. The glove also senses the user's fingers, allowing the computer to interpret hand gestures. A virtual object can be "picked up" by simply closing the fist over the object, just as in the real world. Another object can be indicated for some action by

† Employee of Computer Sciences Corporation. Work supported under government contract NAS 2-12961
literally pointing at it in the virtual environment. Using these techniques, virtual environments attempt to create the illusion of a computer generated reality so compelling that one naturally interacts with it as one interacts with the real world. Virtual environments do not attempt to mimic the real world, rather they provide a natural, intuitive interface to computer environments. The goal is to present a researcher with a virtual object for study, as opposed to a picture of an object.

Virtual environments have found fruitful application in the field of scientific visualization. When scientific phenomena under study are three-dimensional and contain complex structures, virtual environments provide a natural means of display. Virtual environment control via the glove and other input technologies allow simple, intuitive control of the three-dimensional position and orientation of the displays involved in the visualization of phenomena. The researcher need not remember things like “when the control key is pressed the mouse motion is mapped into roll and yaw”. This control capability is particularly useful when the richness of the phenomena allows only partial display at any one time. Using virtual environment control techniques, the researcher can rapidly change what and where data is displayed, allowing the exploration of complex data environments. We feel that it is this exploration capability which brings out the real power of virtual environments. As Brad Comes of the Army Corps of Engineers Waterways Experimental Station exclaimed when he first tried a virtual environment, “with this kind of interface, you can get inquisitive!”

Virtual environment interfaces are very striking, but the advantages of this interface are apparent only for those using the virtual environment system. For this reason, virtual environment interfaces are not particularly useful for the presentation of scientific results. They are useful, however, for the exploration and hopefully discovery of phenomena which can then be presented in more conventional ways.

Research in the use of virtual environments in scientific visualization is underway at several locations. NASA Ames Research Center is pioneering these applications in several areas of scientific investigation, particularly fluid flow visualization. The Army Corps of Engineers Waterways Experimental Station has duplicated the system at NASA Ames for research in water flow visualization. The National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign is developing several virtual environment setups, including a duplicate of the system at NASA Ames. They are investigating applications in cosmoology, magnetohydrodynamics, and medical imaging. The Electronic Visualization Laboratory at the University of Illinois at Chicago has developed a virtual reality display environment known as the CAVE [4] dedicated to a variety of scientific visualization applications. The University of North Carolina at Chapel Hill is investigating several applications of virtual environments in such fields as molecular modeling and medical visualization. SRI International is also evaluating the use virtual environments for the study of molecular models. Other institutions are acquiring or thinking seriously of acquiring virtual environment systems.

While virtual environments are rather new, there have been interesting applications developed for scientific visualization. After describing a major example, we will discuss what makes a virtual environment work and how to evaluate whether a particular visualization problem is suited for a virtual environment with current technology. This technology certainly has a long way to go before the full potential of virtual environments is realized, but it has matured to the point where significant scientific visualization problems can be addressed.
The Virtual Windtunnel

2) The virtual windtunnel is an application of virtual environments to the problem of fluid flow visualization developed at the Applied Research Branch of the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center [3][5]. It is designed to visualize pre-computed simulated unsteady three-dimensional fluid flows which are the product of computational fluid dynamics (CFD) calculations. These calculations are typically performed on supercomputers and provide velocity, energy, and pressure data of fluids on curvilinear multiple-zone numerical grids. Visualization of these unsteady flows is difficult due to their often extremely complex time-varying three-dimensional structures. There are many methods for visualizing these flows using computer graphics, such as isosurfaces of scalar quantities, cutting planes rendered with color maps that indicate scalar quantities, and streamlines of the flow. Due to the inherently three-dimensional structure of flow phenomena, virtual environments were expected to be useful.

The virtual windtunnel uses a variety of techniques to visualize a fluid's vector (i.e. fluid velocity) and scalar fields (i.e. pressure or density). Streamlines (figure 2) are an example of a technique used to visualize a vector field. Streamlines are integral curves of a vector field given an initial position or seedpoint, and provide insight into the vector field's geometrical structure. These visualizations are rendered either as lines connecting the points of the paths or as disconnected points. Streamlines are computed by iteratively integrating the vector field given the initial seedpoint. Another technique analogous to streamlines is to inject simulated particles into the vector field from an initial seedpoint. A collection of such particles is known as a streakline. Streamlines and streaklines are interactively controlled by controlling the positions of their seedpoints. These seedpoints are controlled by rakes, which are linear collections of seedpoints that are moved by the user's hand via an instrumented glove. A rake is simply picked up and moved to a new, desired location. Several rakes with different visualization techniques can be operated at the same time (figure 3).

By creating the illusion that the researcher is immersed in the flow under study with real rakes "out there" within reach, the researcher can concentrate on the science of the problem, and not worry about the details of the interface. By waving a rake of streamlines around, the interesting areas of the geometry of the flow can be quickly identified. By watching a streakline develop, vortical structures can be identified which can then be explored with streamlines. The passage of time can be sped up, slowed down, stopped or reversed. The scale of the display and interaction can be controlled at will, as can the display of objects inside the flow.

The design and implementation of the virtual windtunnel involves several constraints. The most severe constraint is the requirement that as the user's head moves the scene must update sufficiently quickly that the illusion of viewing a real environment is not destroyed. Slow update causes the display to look like a series of still pictures rather than a dynamic view of a three-dimensional world. Experience has shown that to maintain the illusion an update rate of about ten frames/second is required. The rendering in stereo of the environment must take place in less than one tenth of a second for the illusion of reality to be compelling. In order to reduce this problem to one of graphics only, it was necessary to decouple the graphical update process from the computational process. Thus even if a very long computation is taking place the graphics will proceed uninterrupted.

A second constraint is that the various visualization tools respond to the user's actions in something resembling real-time. This requirement derives from the desire to provide continuous feedback to the researcher as to the state of the tool being manipulated as well as the phenomena being displayed by the tool. For example, in the use of streamlines to locate vortical phenomena, the user may sweep a rake of streamlines about looking for vortical structures. A long delay between the time when the user moves a rake to a specific
location and when the system displays the streamlines from that location will have the following detrimental effect: The streamlines being currently displayed will not correspond to the user's actual hand position if the hand is moving. Thus when an interesting feature is observed, this feature corresponds to where the researcher's hand used to be sometime in the past. As is well known from various human factors studies [6], locating the specific location corresponding to the interesting phenomena in the presence of long delays is a frustrating and difficult task. Experience has shown that to maintain good feedback and control these delays should be less than 1/10th of a second. Somewhat longer delays, up to about a third of a second, can be compensated for by using slower control motions, but in this realm control is difficult and frustrating.

In choosing the visualizations, it was important that the rendering involved could be performed within the ten frames/second constraint, and that the computation could be performed within a tenth of a second. Streamlines and their unsteady generalizations based on simulated particles are an example of a technique which can be made to satisfy this constraint. Computation of streamlines involves only simple numerical integration of the vector field and the rendering of streamlines involve drawing simple lines in the three-dimensional environment. The accuracy of the computation was also determined by performance considerations. The second order Runge-Kutta integration technique was chosen as a good compromise between performance and accuracy.

The virtual windtunnel provides a rich environment for the visualization of fluid flows, which contains several visualization techniques and data displays. The properties of these techniques can be controlled by the researcher via various tools and widgets in the virtual environment. These tools exist as part of the environment (as opposed to conventional graphical user interfaces in conventional desktop graphics), and include menus and sliders. It has been found that these tools allow the control of the virtual environment using only two (logical) gestures: grabbing objects and pointing at objects.

The display device was constrained by the demand that it provide as high quality display as possible. Many wide field of view stereoscopic head-tracked displays use a pair of four-inch diagonal liquid crystal displays (LCDs) worn on the head. These LCD systems were, however, judged to provide too low a resolution to be acceptable. The Fake Space Labs (Menlo Park, Ca.) Binocular Omni Orientation Monitor (BOOM), a device using a pair of four-inch diagonal monochromatic NTSC cathode ray tube (CRT) monitors supported on a counterweighted yoke assembly (figure 4), was chosen because of its superior display quality. This was later upgraded to the Fake Space Labs BOOM IIIC, which uses 1000x1000 resolution monitors with two color channels. A standard shadow-masked three-color CRT display was rejected because the shadow mask degraded the image and because of safety uncertainties. Head tracking on the BOOM systems is performed by providing the computer system with the angles of the joints in the counterweighted yoke assembly, which are detected via optical encoders.

The Dataglove Model II developed by VPL Research, Inc. (Redwood City, Ca.) was chosen for the control device (figure 4). The dataglove uses a magnetic tracking system built by Polhemus Inc (Colchester, Vermont), which provides the absolute position and orientation of a sensor relative to a source. While somewhat inaccurate, this tracker is sufficient for our purposes. The glove itself uses specially treated optical fibers at ten finger joints to measure the finger bend angles of the user's hand. These angles are interpreted as gestures by the computer system, which cause different things to happen depending on where the gesture is performed. For example, if the closed fist gesture is performed at the center of a rake, that rake is picked up and moved with the user's hand until the closed fist gesture is released. A closed fist gesture on a slider changes some parameter of the environment such as scale or the number of streamlines on a particular rake. A closed fist gesture in the open air allows the movement of the entire display relative to the researcher. A point gesture accesses a menu of command options.

Finally, the computation and rendering platform must be sufficiently powerful to integrate the computation, control and display of all the elements of the virtual environment.
For small unsteady flows, we use a Silicon Graphics (Mountain View, Ca.) Iris 380 GT/VGX system, which has eight MIPS R3000 processors for a total 37 megaflops performance and a VGX high-speed three-dimensional graphics geometry engine capable of drawing 800,000 small triangles/second. Our system contains 256 megabytes of physical memory. The Iris system reads the control devices to determine the position of the user's head and the state of the user's hand, computes the visualizations for the current display, and renders the display in stereo from the user's point of view. This system is sufficient for unsteady data sets which involve less than a total of 250 megabytes of data and visualizations involving up to about 20,000 integrations of the vector field.

On the Iris, sufficient speed is attained by storing the entire data set in memory so it can be very quickly accessed. Most interesting unsteady flows involve more than 256 megabytes of data, however, and so will not fit into the Iris' memory. Storing the data on disk and reading each timestep's data when it is needed is too slow on the Iris to allow the system to run at ten frame/second. For this reason, a distributed architecture has been implemented where the flow data and computation of the visualizations all take place on a remote supercomputer [5]. We currently use a Convex C3240, which has four vector processors and a gigabyte of physical memory. This allows four times as much flow data to be visualized, but many unsteady data sets are larger still. A disk bandwidth of about 50 Megabytes/second allows the dynamic loading of data stored on disk so long as the size of one timestep's worth of data does not exceed five megabytes. One might expect that the computational performance of the Convex 3240 would also provide advantages, but particle integration is not well suited to vectorized computation due to the constant access of memory, and the performance of the Convex with four processors is comparable to the Iris with 8 processors.

Because the virtual environment display fills the user's field of view, shutting out the real world, virtual environments are a highly personal experience. An advantage of a distributed architecture is the ability to build shared virtual environments, where two or more users using different virtual environment systems can view the same data set that is resident on a remote supercomputer. The UltraNet network system has sufficient bandwidth to transmit the visualization data. By sending the user's commands directly to the supercomputer, each of the users can interact with the same virtual environment and see the effects of the other user's actions.

The most serious problem encountered in the implementation of the virtual windtunnel is that of data management. An typical example of a modern data set is the computation of a hovering Harrier jump jet [7] which contains 106 of timesteps each containing three million points with 53.4 megabytes of data. The total size of the data set is 5.5 gigabytes. This size of data set presents serious data management problems when the available hardware systems have a physical memory of only one gigabyte. Solutions include investigation of small volumes of data, subsets over time, or the investigation of single components of the full dataset (i.e. only the pressure data set).

**Other Application Examples**

3) The techniques developed in the virtual windtunnel have been applied to the visualization of curved spacetime in the general theory of relativity [8]. The flow data in the virtual windtunnel is replaced by geometry data in the form of the metric of spacetime. The streamlines are replaced by geodesics in the geometry data (figure 5). Geodesics in spacetime require an initial speed, direction and position as opposed to the initial position required by streamlines of a flow. The user's hand position and orientation are used to supply this initial position and orientation for a preset speed. This application is being developed jointly by the Applied Research Branch at NASA Ames and the numerical relativity group at NCSA to study the results of numerical spacetime simulations.

At NCSA, the BOOM has been used to provide a display of a static three-dimensional map of galaxy distribution due to Margaret Geller. The data in this application are the
measured three-dimensional positions of galaxies on a very large scale. This application was very successful in bringing out structures in the galaxy distribution that were not previously perceived. Another application under development at NCSCA is the investigation of large scale cosmological structures that are the result of n-body simulations.

The CAVE system at the University of Illinois at Chicago has been used to implement systems for molecular visualization and medical visualization. The cave provides an illusion of presence and immersion by surrounding the user with large stereo projection screens. The environment is rendered on the screens from the user's point of view, as tracked via an electromagnetic position and orientation tracker. This system has the advantage of allowing other observers to see the actions of the user without the need for another virtual reality system.

A somewhat different application that has been explored is the use of interactive virtual environments to allow the manipulation of mathematical surfaces to illustrate geometrical and topological concepts. This application involves some rather difficult questions of how the surface is to react to the user's inputs [9]. The visualization of mathematical concepts leading to the discovery of the remarkable properties of four-dimensional spaces is one possible application [10].

**Considerations in the Development of Virtual Environments**

4) From the virtual windtunnel example, we can draw several lessons about what makes a virtual environment useful. The three-dimensional control and head-tracked display in real time are crucial to the illusion of a real virtual world. While the virtual objects must be meaningful to the researcher, it is not important that elements of the virtual world look like anything in the real world. The sense of presence that the virtual objects have due to the head-tracked display and user interaction compel the illusion of reality. This sense of presence allows the researcher to interact with the objects exactly as if they were objects in the real world. When the researcher says "I want to move the streamline over there", the researcher simply reaches out and grabs the seedpoint of that streamline and moves it there. Asking "I wonder where the vortices are in this time frame", the researcher grabs a rake of streamlines and waves it about in the virtual space watching the resulting streamlines to look for signs of vortical structure. This kind of exploratory interaction requires fast update rates in the virtual environment.

The control and display devices are also critical. The control device must track the user's body in such a way that the user can forget that the device is there. We wish to make the interface to the virtual environment as invisible as possible. The magnetic trackers and gloves used in current virtual environments have serious accuracy problems and are functional within only a limited range. The display devices suffer from a severe resolution restriction, due to the fact that a four-inch display is blown up to a typically 100 degree image to cover the user's field of view. Pixels become large with this kind of magnification, and even in the highest resolution systems with 1000 pixels on a side the pixels are plainly visible. In this case the pixels are as large as one tenth of a degree across, which is one fifth of the full moon. Stereoscopic display and color rendering help considerably. The CRT systems used in the boom have enough resolution to be useful, but higher resolution is highly desirable.

Other control and display devices would also be useful. The virtual windtunnel system contains only the minimum control and display capability for a viable virtual environment. An obvious addition would be a voice recognition system. This would allow the researcher to talk to the system directing various aspects of the environment. While the glove is appropriate for 'manual' tasks such as the movement of a rake, it is less suited for the control of more abstract quantities such as the type and number of seedpoints on a rake. A voice recognition system will be integrated into the virtual windtunnel this year for additional control. Other possible controls include the six-degree-of-freedom spaceball, which senses the force and torque applied by the user about a point. The force
and torque can be interpreted as six numbers that are used to control various aspects of the environment. One use of the spaceball is to control the user's position and orientation within the virtual environment.

The computational platform for the virtual environment must be capable of performing the computations involved in the desired visualizations. Isosurfaces, for example, are computationally intensive and require more computational speed for real-time interaction than is available in most graphics workstations. The computer platform must, on the other hand, read the interface devices and render the graphics associated with virtual environments. Workstations are typically well suited for this task. The further development of high-power graphics workstations will enable a wider variety of applications of virtual environments to scientific applications. Distributed architectures when available are often desirable, where the workstation reads the interface devices and renders the graphics, while a remote supercomputer performs the computations.

The most severe problem is that of large amounts of data. The nature of interactive exploration requires that the data be accessed at high speed in unpredictable ways. This problem was already discussed in the context of large unsteady flows in the virtual wind tunnel. Interesting data sets besides fluid flow have this problem in very severe ways. Numerical spacetime data are on grids of size comparable to those in fluid flow, but each grid point has much more data. Data returned by the Earth Observation Satellite (EOS) will be massive. The ability to quickly get at large amounts of stored data is one of the primary bottlenecks in the use of virtual environments for the visualization of large data sets.

One may be tempted to suggest that many of the problems encountered in the application of virtual environments to scientific visualization will be solved through advances in technology. While these advances will significantly ameliorate these problems, particularly the data management and computational performance, the demands of data sets will stay well ahead of the capabilities of the systems. Greater computational power and storage capacity will lead to significantly larger and more complex data sets, increasing the demands on the visualization system.

**Future Directions**

5) The advantages of virtual environments in the unambiguous display of three-dimensional structures and the intuitive three-dimensional control of objects in the virtual environment can potentially be of great use in scientific visualization. The ability to explore complex data by selectively displaying aspects of that data is the most dramatic of these advantages.

The real power of virtual environments comes when abstract concepts which have no tangible real world counterpart such as streamlines in an unsteady flow are used to create virtual objects. By making these abstract concepts tangible their investigation is greatly facilitated. One is tempted to call this process "scientific reification". The examples above take physical quantities such as airflow and render derived structures such as vortices as real tangible things with tangible properties. The same principles can be applied to more abstract quantities, such as statistical data or mathematical models. This would aid the pedagogical presentation of abstract concepts as well as the investigation of abstract phenomena.

What kinds of applications will find virtual environments useful? At this stage of development, virtual environment technology is "clumsy", full of difficulties and limitations which must be worked around to provide a viable application. There are undoubtedly several interesting applications that can be developed now with the current technology. As the technology advances, qualitatively different applications may appear that are as far beyond the current systems as the current systems are beyond early computer plotter drawings.

Given the current state of the art, any visualization that involves complex three-dimensional data which is moderately computationally intensive will probably benefit from
a virtual environment interface. While the current cost in both hardware and labor of virtual environment development is high, research in the effective use of virtual environments in scientific visualization is needed. The few successful examples that exist only hint at the usefulness and limits of virtual environments.

Conclusions

6) Virtual environments, at least at the current level of technology, is not intended to be a panacea for every computer graphics interface problem. Virtual environment interfaces are a tool and like all tools have their place. Ultimately, it may be that virtual environments will include all other interface paradigms as subsets, but that day is far in the future. In the meantime, however, there seems to be a class of problems, particularly in the scientific visualization of complex three-dimensional phenomena, where virtual environments provide significant advantages.

References


Figure captions

Figure 1: The Virtual Windtunnel in use, with the flow around the space shuttle.

Figure 2: Streamlines of the flow around a harrier jump jet in hover [7].
Figure 3: The user environment in the virtual wind tunnel, showing a rake of streaklines rendered as bubbles used in combination with a rake of streamlines.

Figure 4: Boom and glove hardware interface to the Virtual Wind tunnel.
Figure 5: An illustration of virtual spacetime in use. The user is manipulating a spray of geodesics in a curved spacetime with an instrumented glove while observing the results in a head-tracked, wide field of view stereo display.
Open Software: UNIX, DCE, and Competitors

Thomas W. Doeppner Jr.

Department of Computer Science, Brown University, Providence, RI 02912, USA

Abstract

Open software is software contributing to a general goal of systems that do not depend on any one hardware or software vendor and are easily extensible by adding new software built upon the functionality of existing software. We discuss some of the defining characteristics of open software and frameworks that ease its production. These frameworks include operating systems and software supporting distributed computing. Examples are UNIX, NT, and DCE. We also examine two technological developments that are becoming very important in the development and use of these frameworks. The first is a structuring technique for operating systems—microkernels. The second is a technique for doing concurrent and parallel computation—multithreaded programming.

1 Introduction

What is open software? It is software that contributes to a general goal of systems that do not depend on any one hardware or software vendor and are easily extensible by adding new software built upon the functionality of existing software. A proprietary operating system, i.e., one that runs only on the hardware of one particular vendor, is not open software. Its users are dependent of the vendor of the hardware—they cannot easily take advantage of the hardware of other vendors. An operating system that runs on the hardware of a number of vendors but is supplied by a single software vendor is also not open software, since its users cannot take advantage of alternative implementations supplying new functionality. A document-preparation package that does not allow the importation of text and drawings from other packages or that does not produce output that can be incorporated into documents produced by other packages is not open software. It is not extensible by means of other software—it is a closed system.

Open software is not necessarily free software. People who produce software need to make a living. How can company A and company B both produce implementations of operating system X? They could both license the operating system from company C. Or they could both develop the system from scratch based on specifications published by some organization D. In the first case, if company C is a competitor of A and B, then A and B, by licensing C’s operating system, become subservient to C and the operating system is closed software. But if C is not a competitor of A and B but is a consortium of competing companies including A and B, then operating system X is open software. Similar arguments apply in the second case: if the organization that publishes the specification produces that specification through an open process, i.e., a process involving all parties affected by the specification, then software adhering to the specification is open software.

What is important to software users is that it fit within a heterogeneous environment, so that they are not beholden to any one particular vendor for their computing needs. This means that software should be portable—it should run on different machines and on different operating systems.
Furthermore, software should be *interoperable*—each software package should be able to work in conjunction with other software packages as appropriate. A simple example is the ubiquity of PostScript—programs that produce PostScript can be used in conjunction with programs that consume PostScript. In the same vein, the drawings produced by a drawing program should be easily integrated into a document produced by a desktop publishing package. A publishing package that does not accept input from other sources is closed software.

What is important to software vendors is that they be able to produce open software. This means that they need to know the *specifications* of the software with which they must interoperate. Vendors of computing platforms must be able to *license* systems software, such as operating systems.

For open software to be practical, there must be industry-wide agreement on the *frameworks* for building systems. A company that produces software to run on a number of different machines will have an easier time developing and supporting such software if it can rely on a common operating-system interface on all the target machines. Similarly, developers of software that is to fit within a heterogeneous, distributed environment can greatly benefit from a standard framework for distributed computing.

In the remainder of this paper we discuss these frameworks. First we look at operating systems, briefly comparing UNIX and NT. Then we examine two technological developments that are becoming very important in the development and use of these frameworks. The first is a structuring technique for operating systems—microkernels. The second is a technique for doing concurrent and parallel computation—multithreaded programming. Finally we discuss DCE—the OSF's distributed computing environment.

2 Operating Systems

A basic framework for building systems is the operating system. UNIX is probably the first example of an operating system that comes close to being "open." MS/DOS, in conjunction with Microsoft Windows, certainly runs on more computers than UNIX is. But MS/DOS and Windows are not portable¹ and no one other than Microsoft has any say on what goes into them—they are not examples of open software. The MS/DOS operating system has been described as "functionally impaired"—in that a great number of desirable features found in "modern" operating systems that are not found in MS/DOS.

Recently Microsoft has introduced a new operating system, NT, and a new version of Windows to run in conjunction with it. Is NT a better operating system than UNIX? Is NT an open operating system? Before discussing these questions, let's first look briefly at UNIX.

There are other operating systems on the market as well, for example, OS/2 and the Macintosh operating system. We do not wish to slight either of them, but, to keep our discussion simple, we look at only UNIX and NT.

2.1 UNIX

The UNIX operating system has changed considerably since its birth in the late '60s and early '70s. At first it was strictly a time-sharing operating system for small computers. In the late '70s and early '80s it evolved into a time-sharing operating system for much larger computers. In the latter stages of this period it became the standard operating system on the ARPAnet and was the base operating system for much research on distributed computing. In the mid-'80s it began its most recent evolutionary stage and became a workstation operating system (and perhaps a personal-computer operating system).

1. However, a number of companies have recently introduced the capability to emulate MS/DOS and Windows on non-Intel hardware to the extent that one can run Windows applications at speeds comparable to its performance on Intel hardware.
A great deal of new technology has been integrated into UNIX in the past decade, much of it in the past few years. Within the kernel of the operating system, this new technology includes advanced file systems, support for multiple threads of control within a process, shared libraries and dynamic linking, support for large address spaces, enhanced security, and support for multiple processors. New technology provided on top of the kernel includes multiple GUIs (graphical user interfaces) and internationalized libraries.

2.2 NT

Is NT better than UNIX? In terms of the functionality provided by the operating systems themselves, there appears to be little difference between the two. Many current implementations of UNIX are more "mature" than the current implementation of NT but the immaturity of NT is no doubt a transient phenomenon. The feature set of NT is certainly larger (and better) than that of "traditional" UNIX, but modern UNIX implementations support the same sorts of desirable features as does NT.

Is NT an open operating system? Unlike MS/DOS, NT is portable, both in theory and in fact. As of this writing, NT has working implementations on at least three different processors: Intel, MIPS, and Alpha. NT is certainly licensable—this is Microsoft's means for making money. There are no detailed published specifications for NT, but this also appears to be temporary. The only criterion for openness not met by NT is the open process—the design of NT is dictated strictly by Microsoft. However, just as a benevolent dictatorship may perhaps be the best form of government, this is not necessarily a bad thing.

2.3 Does the Operating System Matter?

If we agree that both UNIX and NT are acceptable operating systems, and if software vendors sell versions of their software to run on both operating systems, do software users care which operating system is used? Clearly the answer is no, it doesn't matter. One can use the operating system of one's choice and have no restrictions on the software packages that can run on the system.

There are two parts to the premise of the statement that begins the previous paragraph. That the operating systems are "acceptable" is a technical issue, that software vendors sell versions of their software to run on both operating systems is a marketing issue. The technical issue is relatively easy to decide. Some argue that the marketing issue is easy as well (i.e., that no one can compete with Microsoft). The point is that the answer to the question "Does the operating system matter?" appears now to be strictly a marketing issue.

3 Microkernels

Microkernels are a structuring technique for building operating systems. Rather than structure an OS as a single, monolithic piece of software, it is split into microkernel and one or more personalities. The microkernel provides basic functionality, such as virtual memory, low-level I/O, and scheduling, while the personality uses the functionality of the microkernel to build the higher-level abstractions, such as processes and files, that are used by user applications. The code implementing the microkernel must run in privileged mode, i.e., it is in the "kernel." However, the code implementing the personality can run in user mode—it does nothing that would require it to run in privileged mode. The basic functionality of a microkernel can be used to build a variety of different personalities and thus provide a variety of different operating-system interfaces. Figure 1 shows a microkernel running in privileged mode and a personality and a number of user applications running in user mode. The personality and each of the user applications reside in different address spaces—thus the personality is isolated from user applications, just as it would be if it were in the kernel. The personality is a client of the microkernel—it obtains services from the microkernel. The user applications are clients of the
personality—they obtain operating-system services from the personality (though the services of the microkernel might be used to provide communication between the applications and the personality).

Two microkernels that have achieved some degree of success are Chorus [7], from Chorus Systems, and Mach [4], from Carnegie Mellon University. Both have been used as the basis of UNIX implementations: Chorus with UNIX System V Release 4 [1] and Mach with Berkeley UNIX and OSF/1 [5]. There have been a few experiments in building personalities for other operating systems on top of Mach [6, 9]. IBM has demonstrated an OS/2 personality for Mach. Furthermore, the architecture of NT is based on microkernel concepts [3].

3.1 Why Use Microkernels?

Why is there interest in microkernels? We argue below that execution speed is not the reason—a microkernel-based implementation of an operating system may well be inherently slower than a traditional monolithic implementation. The interest is there because of the promise of convenience—using microkernels should make a number of things easier to do, including debugging, supporting multiple operating systems, and distributing functionality.

In the monolithic approach to operating system design, all components of the operating system reside in the kernel. This often makes debugging the operating system difficult, because most debugging tools are designed to debug user-level software. Kernel-level debugging tools are typically more primitive and cumbersome than user-level tools. In the microkernel approach, since the personality runs in user level, it can be debugged using the standard user level tools. This is accomplished as shown in Figure 2: The personality being debugged, the "test personality," is run in an environment set up by the "master personality." The debugger, which gets its operating-system services from the master personality, has control over the test personality and all of its clients (i.e., user applications). Since the master personality has control over the test personality, it can pass this control on to the debugger, allowing the debugger to control (and monitor) the actions of the test personality and its clients.

Personality modules for different operating systems can be built on top of a single microkernel. Thus one can run two different personalities simultaneously on the same computer, for example run a DOS program while running UNIX, or debug one operating-system personality using the debugging tools of another.

![Figure 1: The Microkernel Approach](image-url)
What is more important is that a vendor can take advantage of the microkernel approach to ease its task of supporting multiple operating systems. Functionality common to a number of operating systems can be either put in the microkernel or made available as separate, user-level servers that can be used by the different personality modules. For example, device drivers and other machine-specific code can be shared by the various personalities—this reduces the amount of code that must be written and the number of modules that must be maintained. Any one computer might run just one personality, but some of the modules used by this personality can be used by other personalities as well.

Why is this sort of sharing not done among monolithic implementations of the operating systems, using their device drivers and other machine-specific modules? The problem is that such modules fit only into the systems for which they were designed. The internal interfaces within an operating system may be specified and well known, but they are peculiar to a particular operating system. The modules use data structures and global variables that are defined only within their intended system. The advantage of the microkernel approach is that it forces a modular design approach that can be used to eliminate such dependencies. It is not that such a design approach cannot be used with monolithic kernels, it is that with microkernels this approach must be used. Furthermore, as discussed below, a lot of machinery is available in the microkernel itself to make the communication between modules, even if they reside in different address spaces, relatively efficient.

A final reason for the convenience of the microkernel approach is that it aids the development of multicomputer-based systems, i.e., collections of computers that do not share memory, but communicate via some sort of high-bandwidth communications interconnect. We would like such multicomputers to behave as much as possible as if they were a single computer system, running a single operating-system image. Even though we want to reap all the advantages of parallelism, we would like to avoid the difficulties of parallel programming.

The microkernel approach does not eliminate the difficulties of parallel programming, but it does give us a good start toward an implementation of an operating system for a multicomputer. How this is done is illustrated in Figure 3, which shows a multicomputer consisting of six interconnected computers. What was originally a single operating-system personality has been further subdivided along functional lines—here it is split into process management, file management, network management, database management, X/Window management, video man-

![Diagram](image)

Figure 2: Debugging Personality Code
agement, and audio management components. The components reside in separate address spaces and communicate with one another using the facilities of the microkernel. If all components are on the same computer, then communication is straightforward. However, the microkernel's communication facilities allow individual components to be moved to other computers transparently—two components can communicate with each other without regard to location. Thus, in Figure 3, each component has been assigned to one or more computers, with no change to the logical structure of the composite operating system. User applications can run on any computer (or collection of computers) and have full, transparent access to all components of the operating-system personality.

More problems must clearly be dealt with before we have a good multicomputer operating system, but many of the major pieces are in place. This approach has been used for the design of the OSF/1 AD operating system, which is used on the Intel Paragon and the Thinking Machines CM5.

3.2 How Microkernels Work
The basic concepts used by both Chorus and Mach are quite similar. We discuss Mach, but most of what we say also holds for Chorus.

Mach provides a small number of basic abstractions, of which the most important are listed below:

- **task**: an address space and a holder of capabilities.
- **thread**: the abstraction of a processor. Threads execute independently of one another; their execution is multiplexed on the available processors. Threads are restricted to execute within a task; there can be any number of threads within a task.
- **port**: a one-way communication channel. One task is the holder of receive rights for the port; it, and only it, can receive messages from the port. Any number of tasks have send rights for the port; they can send messages through the port.
- **message**: the unit of communication through ports. A message contains an arbitrary amount of data. Messages may include receive or send rights for ports. Thus the

![Diagram of microkernel approach on a multicomputer](image)

**Figure 3**: The Microkernel Approach on a Multicomputer
holder of the receive right for a port may pass it on to another task via a message. Similarly, a holder of a send right may give a copy of the send right to another task by including the right in a message sent through some other port.

- memory object: some "thing," e.g., a file, that is mapped into the address space of a task.

Ports are used not only as communication channels, but also as a means for identifying objects: An object is maintained inside of some task. A port is created in the task and the receive right for that port is associated with the object; send rights for the port are given to other tasks that as references to the object. A task sends a message through such an object-reference port, and this message is interpreted by the receiving task as pertaining to the associated object.

Tasks and memory objects form the pieces of a system. Threads are the active agents; ports and messages are the glue that holds things together. To make this a bit more precise, Figure 4 shows how UNIX might be implemented via a microkernel. The UNIX personality is implemented as a task with a number of threads executing inside of it. It maintains process objects, one per UNIX process (we think of these processes as being clients of the personality), and associates with each process object a port. These objects contain process-specific information such as the table of open files, signal-handling information, process ID, etc. The personality owns the receive rights for these ports and gives the send rights to the respective processes.

Each UNIX process is implemented as a task containing one or more threads. It is given the send rights to the port associated with the process's process object in the personality. Calls by the process to the operating system (i.e., system calls) are converted into messages sent through the port to the personality. (How are the responses to these calls handled? Ports are one-way communication channels, so the port used to send a request cannot be used to send the response. Instead, the requesting task provides send rights to a separate response port as part of

![Figure 4: UNIX via the Microkernel Approach](image)

234
the message. The receiving task uses these send rights to send its response back to the requestor.) Thus the port/message facility is the means for transferring a process's requests to the UNIX personality and for transferring the responses back to the process.

What remains to be seen is how each process's system calls are converted into message exchanges. The most straightforward way to accomplish this involves the system-call library a library of procedures linked into each process that has one entry per system call. The body of each routine in the library performs the appropriate trap instruction to transfer control to the kernel where, in a monolithic implementation of UNIX, one would find the UNIX operating-system functionality. For the microkernel version, this library can be replaced with one in which the traps are replaced with code that invokes Mach's message-passing calls.

The only problem with this approach is that it does not work if it is not possible to replace the system-call library. This is the case if the library has been statically linked into the program executed by the process; this static linking was, until recently, the only form of linking in UNIX. Thus another technique is required if we are to support such statically linked programs.

Mach provides a system-call redirection facility in which the microkernel automatically redirects system calls back to user space. As used in OSF/1 Release 1.3, whenever a process issues a UNIX system-call trap, the microkernel converts the trap into a message and sends the message to the UNIX personality, via the process's port. Thus a program that was statically linked for a monolithic UNIX implementation can execute correctly without modification on the microkernel implementation.

3.2.1 Performance

There is nothing about the microkernel approach that makes microkernel-based implementations of operating systems any faster than monolithic implementations. In fact, all published studies comparing the two show that the microkernel-based implementations are always slower. Why this is so is not hard to see. With the personality code running in user mode, requests of user applications for operating-system services must first go into the kernel, then back to user space to the personality module. Furthermore, large amounts of data must be transferred among the kernel, the user application, and the personality.

A great deal of work has been done to optimize the performance of microkernels. The time to switch from user mode to kernel mode and back to user mode has been minimized and a variety of techniques are employed to reduce the costs of the data transfers. But monolithic operating systems are still faster. An approach pioneered by Chorus, and now used in Mach, is to develop and test the system with the personality running in user mode, but move the personality into the kernel (while still maintaining its modularity) for production purposes. This does speed things up, but even still the monolithic approach is faster.

This problem with performance is not, however, a serious blow to the microkernel approach. Few new operating systems are faster than their predecessors. Advances in operating-system technology makes programming easier, systems easier to maintain and easier to extend. In some cases speed is the most important criterion, but in many cases, if not most, convenience is what sells.

4 Multithreaded Programming

Multithreading is a technique for writing concurrent programs, i.e., programs in which many things are taking place at once. It is supported by most modern operating systems, including UNIX, NT, and OS/2.

A thread of control (or, simply, thread) is the abstraction of a processor; it is an execution sequence. A normal, sequential program has a single thread of control. Thus if we have two programs running independently but at the same time, we have two threads, each executing a
separate program. *Concurrency* arises when a number (at least two) of threads are *in progress* at the same time; in contrast, *parallelism* arises when a number of threads are executing simultaneously.

A good example of an application that is easier to write as a multithreaded program than as a single-threaded one is a server handling queries of a simple database from a number of different clients (see Figure 5). In a single-threaded program, shown in the upper portion of the figure, the single thread either would have to serve one client at a time, resulting in excessive delays for clients with quick queries, or would explicitly need to multiplex the requests of the various clients. This latter approach would yield a complex program that would be difficult to debug and test.

On the other hand, the multithreaded program, shown in the lower portion of Figure 5, would be very simple. The programmer writes the code to deal with one client. Each client request is handled by a separate thread, but all such threads are executing the same code. There is no need to worry about multiplexing—the threads-support system takes care of that. The only thing different from a single-threaded program is that the programmer must deal with synchronizing the access to the database, but this is not difficult. Thus the resulting program does not delay clients with quick queries and is much simpler and easier to debug and test than the single-threaded version. A further advantage of the use of threads in this example is that, while one thread does I/O in response to its client’s request, other threads can do other processing. Thus we get concurrency between I/O and computation.

Another advantage of multithreading is that it can be used to exploit a parallel computer. Figure 6 shows the multiplication of two matrices on a *shared-memory multiprocessor*. Rather than use some fancy parallel matrix-multiplication algorithm, we simply parallelize the simple approach: element \( i,j \) of the product matrix is computed directly as the inner product of row \( i \) of the multiplier matrix and column \( j \) of the multiplicand matrix. If the product matrix has \( m \) rows and \( p \) columns and if \( m*p \) processors are available, then each of the \( m*p \) threads can simply compute a single inner product. The threads-support system, in conjunction with the operating

---

**Figure 5:** Two Approaches to the Design of a Database Server
system, insures that each thread runs on a separate processor. If, for example, there are only \( m \) processors, then only \( m \) threads are created and each thread computes a row of the result. With only two processors, two threads are created and each thread computes half of the product.

One might argue that traditional UNIX already supports the concept of threads—each process contains a (single) thread, so that by programming with multiple processes, one is programming with multiple threads. This is true, but misses the point of programming with threads. Creating a process is expensive; creating a thread within an existing process is cheap.

Consider Figure 7. The operating system represents a process with the help of some sort of control block, the process control block (PCB). But a process is also an address space. Creating a process involves creating a new address space. To do this, the operating system must create some sort of memory map (the form of which depends upon the hardware). This new address space must be filled from somewhere. In UNIX, the address space is created as a result of the fork system call and is filled with a copy of the parent process. Of course, the parent’s address space is not really copied into the child’s; it is a highly optimized “logical copy.” But, even so, making it is still a lot more expensive than doing nothing. If the fork is followed by an exec system call, then there is the additional expense of loading the new image from the file system (this too is optimized, but, again, is more expensive than doing nothing).

![Figure 6: Multiplying Two Matrices](image)

![Figure 7: Architectural Support for Processes](image)
Associated with each processor is some sort of address-translation cache, commonly called a translation lookaside buffer (TLB). This TLB contains address translations for the current address space. References to pages whose translations are not in the cache cause a cache miss and force a lookup of the translation in the memory map, which involves a number of extra memory cycles. If the processor switches from one address space to another, the previous contents of the TLB are now useless, and it must suffer a high proportion of cache misses until enough translations for the new address space are in the cache.

Communicating between processes involves transferring data from one address space to another, perhaps via a pipe. This typically requires that the data being transferred be copied, often twice. Some systems provide optimizations for even this, involving tricks with virtual memory, but, once again, doing them is still more expensive than doing nothing.

How is a thread represented? As shown in Figure 8, a thread consists of a thread control block (TCB), a stack, and perhaps some operating system control state, a kernel thread control block (KTCB). Creating a thread is easy—the process already exists, so all we have to do is create a new TCB, allocate a new stack, and possibly create the KTCB. The time required to create a thread depends on the operating system and the hardware, but it is on the order of a thousand times less than the time to create a process.

Switching between threads does not involve switching between address spaces; thus we do not have the problem with refilling the TLB that arises in switching between processes.

Communicating between threads of one process is simple because the threads share everything—in particular, they share their address space. Thus data produced by one thread is immediately available to all the other threads.

One can get the benefits of multithreading in a number of ways. One approach is to build it into the programming language. Not very many languages were designed with multithreading in mind, but one that was is Ada, in which threads are known as tasks. Another approach is to augment an existing language by adding thread support as a new, grafted-on language feature. There are a number of parallel Fortran, all based on this approach. The canonical parallel Fortran feature is the "parallel do-loop," in which multiple threads execute a do-loop in parallel.

![Figure 8: Architectural Support for Threads](image-url)
A final approach is to supply multithreading capabilities via a subroutine library (The subroutine library does not implement all the thread support, but the interface to the thread support is via the subroutine library.) We can use this library with C and, perhaps soon, with C++. (There is no conceptual or technical problem with using threads with the C++ language; the only issue is standardization—what interface to threads should the C++ programmer see?)

Not all applications can benefit from multithreading. A good example of one that cannot is a compute-bound application on a uniprocessor system: the use of threads would slow down such an application. But a number of types of applications do benefit.

As we have already seen, servers that handle multiple clients concurrently can benefit immensely from the use of threads. Such applications do not necessarily run faster with threads, but they are easier to write, debug, and test when written with threads.

Applications that are slowed down because of synchronous waiting can be speeded up with threads—one thread can wait while another thread continues with normal execution. This class of applications includes both client applications that use the services of some server and applications making synchronous I/O requests.

Finally, as we saw in the matrix multiplication example, applications involving parallelizable algorithms running on parallel processors can benefit from threads.

Consider an application that is making a request of a server. If there is something else to do in the application and the request will take a while to be handled, then it makes no sense to have the entire application wait for the response. With threads, one can simply create one thread to wait for the response while the original thread goes on to do other work.

Note that the server need not be on a different machine, or even in a different process. For example, consider a document-formatting system (such as the one used to produce this paper). At times typed (or moused) input is not responded to, because the program is busy doing something else, such as checkpointing the document. If this program had been written with the use of threads, then such waits would be unnecessary, since input processing and checkpointing could take place concurrently.

We have discussed how to use threads to exploit a shared-memory multiprocessor but what about nonshared-memory multiprocessors? A common example of such a system is a collection of workstations. By using threads in conjunction with a remote procedure call (RPC) package, we can distribute our application relatively easily and treat the collection of workstations as a multiprocessor.

One thread might create a number of other child threads. Each of these children could then place a remote procedure call, invoking a procedure on another workstation. The original thread has merely created a number of threads that are now running in parallel; however this parallelism involves other computers.

The history of multithreaded programming goes back to at least the '60s. Its development on UNIX systems began in the mid-'80s. Perhaps surprisingly, there is fair agreement about the features necessary to support multithreaded programming. Even so, a number of different thread packages are available today, each with a different interface. However, for several years a group, known as POSIX 1003.4a, has been working on a standard for multithreaded programming. At the time of this writing, this standard is still in draft form, but it is widely believed that the final form will be agreed upon early in 1994. Once this happens, most vendors of systems supporting multithreaded programming will support the POSIX interface. This will have the extremely important benefit of allowing multithreaded programs to be portable, and thus bring them into the world of open software.
DCE

The **distributed computing environment (DCE)** is an enormous package of software from OSF (built from software provided by a number of its member companies). It provides support for a model of distributed computing known as **client/server computing**: certain computers, servers, provide services to other computers, clients. An important aspect of DCE is that it is operating-system independent (though its early implementations were all UNIX implementations). Thus it provides a computing model that may differ from that provided by the operating system.

The most important part of this model is the interaction between client and server. A server provides some sort of service, something as simple as printing or a bit more complicated such as database management. DCE provides the following features to support this sort of interaction:

- **remote procedure call (RPC)**: this facility extends the notion of procedure calling across machine boundaries. One can place a call to a procedure, even though the code of the procedure and all of its data reside on a different computer. Thus the model of computing is that servers provide procedures that are called by clients.

- **naming and directory services**: servers can put information about themselves into a general directory system so that clients can obtain information about how to access the services provided. This includes support for a number of **name spaces** so that services can be referred to via convenient names (much like file names).

- **security**: one establishes an identity which is valid throughout the distributed environment. Clients and servers can **authenticate** each other; i.e., each can prove to the other that it really is who it claims to be. Furthermore, the client and server can protect the data be passed between them to prevent other parties from modifying it and, if necessary, from reading it. A standard means is provided for **authorization**: objects managed by servers can be tagged with **access control lists (ACLs)** indicating who is allowed to access the object and what they are allowed to do to it.

In addition, DCE provides the following three facilities:

- **distributed time services**: for a variety of reasons, it is important that the notion of time on all computers within a distributed environment be fairly consistent with one another.

- **distributed file system**: though the NFS distributed file system is already used widely, DCE includes another one, **DFS**, that is integrated with the rest of the DCE facilities. Also included is a local file system, known simply as **LFS**, which is also integrated with the other DCE facilities (including ACLs) and which allows quick recovery from crashes.

- **threads**: the use of threads is essential for many aspects of DCE. Since not all operating systems support multithreaded programming, included with DCE is a threads package that supports multithreaded programming using strictly user-mode facilities. It is not a replacement for a good, operating-system-supported threads package, but a stand-in so that DCE can be used while such a good threads package is being developed.

In the following paragraphs we first discuss RPC and how it makes use of some of the other pieces of DCE. We then say a bit more about these other pieces.
5.1 RPC

To understand remote procedure calls, we must first understand local procedure calls. Let's assume that the context of each procedure is stored on the stack in an area known as a *stack frame*. When a procedure is called, the first step is to prepare the current stack frame: the arguments of the call and the address of the next instruction (the return address) are pushed onto the stack. Next, a jump is made to the target procedure, which sets up its own stack frame: it pushes onto the stack the values of any important registers it will be modifying (i.e., it sets up the *register save area* and allocates space on the stack for any local variables it uses. The target procedure can now find the arguments being passed to it, since they are at a known location at the end of the previous (i.e., the caller's) stack frame. When this procedure returns, it restores the caller's context (the saved registers), passes any return value back to it, and transfers control to the return address.

Now consider what happens if we try the same thing when the caller and the callee are on two different machines (or at least in two different address spaces). A naive way of handling this would be to make no changes at all to the code of the caller and the callee, so that the caller's stack frame is on the caller's machine and the callee's stack frame is on the callee's machine. Assume that somehow we arrange for the transfer of control between the caller and callee machines.

We have an immediate problem once we get to the callee procedure: it is running on one machine, but its arguments, inside the caller's stack frame, are on another. Furthermore, when the callee is about to return, it puts its return value in a register, but this register cannot be seen by the caller. We could provide special code inside the callee so that, whenever it attempts to access its arguments in the caller's stack frame, a request is sent to the caller for these arguments. However, among the problems with this approach is that the code produced for the remote version of the procedure would have to be different from that produced for the local version. A better approach would be to transfer the arguments to the remote (i.e. callee's) machine as part of the call, so that they are there when the callee needs them. Furthermore, we can do this so that the code for the remote version of the procedure is identical to the code for the local version.

The technique for doing this is to make use of additional code known as *stub procedures*. These come in pairs—the client-side (or caller-side) stub and the server-side (or callee-side) stub. The client-side stub procedure appears to the caller as if it were the actual callee procedure. The caller calls it, and eventually it returns with the desired result. However what this stub actually does is to put together a message containing the arguments. It transmits this message through the communication medium to the callee's machine, where the message is received by the server-side stub. This stub pulls the arguments from the message and calls the callee procedure with these received arguments. Thus, from the point of view of the callee procedure, the server-side stub is playing the role of the caller procedure. When the callee returns, it returns to the server-side stub, which puts the value being returned into a message and transmits this message back to the client-side stub. This stub pulls the value out of the message and returns it to its caller, the original caller.

A number of details are omitted from the above description:

- *Binding*—where is the callee? I.e., to which machine should the RPC request be sent?
- *Stub generation*—how are the stubs created? How can this be automated?
- *Data transfer*—how are input parameters, output parameters, and return values transferred between machines that may have different ways of representing the various data types?
- *Identification*—how are procedures identified? E.g., how do the client and the server know that they are referring to the same thing?
• *Transport*—what is the transport? Is it reliable? If not, how do we cope with it?
• *Exceptions*—how are runtime problems dealt with?
• *Failures*—what happens if the caller, the server, or both crash?
• *Security*—who is the caller? Who is the server? Who is listening?

### 5.1.1 Binding

In the case of a local procedure call, one must somehow bind the name supplied by the caller to an instance of a procedure with that name, the callee. This is normally a fairly straightforward operation (at least as far as the application programmer is concerned)—one uses a linker that finds an appropriate routine in a library and arranges so that when a call is made to the callee procedure, control is transferred to the procedure obtained from the library.

This must be done in the remote case as well, but here there is the additional problem of identifying the computer to which the call is to be placed. A simple approach is that the programmer explicitly specifies the computer, perhaps as part of the call. More generally, it would be nice to leave this open, to be determined at runtime. Furthermore, this determination of the server machine should be done as transparently as possible to the programmer—ideally the programmer should not need to be aware of the process at all.

For example, the purpose of the remote procedure call might be to retrieve information from a database (perhaps to read one’s mail). Though normally this database might be expected to reside on a single, designated machine, the machine might be down and a backup machine used instead. Or the database might be replicated and any one of a number of different sites could be used.

While the selection of the server could be explicitly coded in the client program, we would like to treat this aspect of writing a distributed program as a separate issue, so as not to interfere with the logic of the program, which should be the same regardless of the machine to which the call is bound. A nondistributed program can be linked to different instances of the procedures it calls merely by supplying it with different libraries. In the same vein, we should be able to bind calls from a distributed program to different servers without having to change the code of the program itself.

In general, we have the name of a desired procedure and pass this name to a name server which returns to us the identity of the server or servers we may use for the call. Such name servers are conceptually very simple: potential servers store information about themselves in the name server’s database; prospective clients query the database, getting names of suitable servers. If there are a number of possible servers, then in most cases some simple technique, such as random selection, is used to determine which server is chosen, though more sophisticated techniques, such as attempting to choose the most lightly loaded server, might be used (though there is no support for this in the current implementation of DCE).

### 5.1.2 Stub Generation

Stubs make remote procedures look like local procedures: the client-side stub looks to the caller as if it were the callee; the server-side stub looks to the callee as if it were the caller. The stubs are responsible for moving the arguments and return value from one side to the other. Creation of the stubs thus involves determining what is being moved.

The big problem is making this determination automatically—we want to generate the stubs from some sort of description of the actual remote procedure. Aside from the complete source code, the only description of a remote procedure commonly available is its declaration (or *signature*). But, certainly in C programs, this declaration is not all that helpful. To produce the stubs, we need to know how much data is being sent as arguments to the procedure. If an argument is an integer (and is declared unambiguously in C as a *long*), then we would know
that four bytes are being transferred\(^1\). But what if the argument is a \textit{char} *? Clearly we do not want to pass just the pointer to the remote procedure; we would want to pass what it points to as well. But how big is the item pointed to? It is probably not intended to refer to a single character. It might refer to a null-terminated character string, or perhaps to an array of bytes of some particular, but specified, size. Sometimes \textit{char} * is used by C programmers to refer to "generic storage," i.e., storage of unspecified size for which, for some reason, it is inconvenient to provide a proper type. In any event, it is, in general, not possible to determine from the procedure's declaration the size of the data that should be passed to the remote procedure.

Clearly, in some languages enough information is provided in procedure declarations to determine the size of data to be transferred. Unfortunately this is not true of C and C++. Rather than redesign these languages completely, what is done is to provide additional information, outside the language, that completely specifies the data being passed to and returned from a remote procedure. Various specification languages have been designed for this purpose, such as the MIG language (used in Mach), the RPC \textit{Language} (used in Sun RPC), and the Network Interface Definition Language (NIDL, used in Apollo's Network Computing System; DCE uses an enhancement of NIDL known as \textit{IDL} (interface definition language).

5.1.3 Data Transfer

Besides knowing what is to be transmitted between caller and callee, we also need to know the form in which data should be transmitted. There are two issues here: the order in which the various pieces of complex data items (such as arrays and structures) are transmitted, and the representation of the primitive data items, such as integers and floating-point numbers.

Two approaches have been used for dealing with these issues. One is to include with the data being transmitted an indication of what is being transmitted. The other is for the caller and callee to agree upon what is to be done ahead of time.

In the Xerox Courier protocol, the first issue (order of transmission) is handled by the first approach: a description of each argument is supplied as part of the call and the return. Most other protocols use the second approach: the sender transmits the data in an order expected by the receiver. This order is determined from the datatypes of the arguments and return value, using some standard scheme. For Sun RPC, this scheme is given by the \textit{external data representation} protocol (XDR); in DCE, it is given by the \textit{network data representation} protocol (NDR).

The representation issue has also been dealt with via both approaches. In Courier and Sun RPC, a single data representation is used by all parties. So, for example, if an integer is being transmitted, then it is always transmitted in \textit{big-endian}\(^2\) form. This avoids all confusion about the representation. However, if a machine's native representation for integers is \textit{little-endian}, it has to convert. Thus if two little-endian machines are communicating with each other they each must convert all outgoing data to big-endian and all incoming data to little-endian.

The alternative approach is used in DCE RPC: the sender transmits the data in its own format and supplies a tag indicating what that format is. If the receiver's format differs from the sender's, then the receiver must convert. But if the formats are the same, no conversion is necessary. This approach is practical because the number of data formats in use is not large.

\footnotesize
\begin{itemize}
\item[\(^1\)] Note, however, that even this is not always true. The DEC C compiler for the Alpha architecture uses \textit{long} to mean an eight-byte integer.
\item[\(^2\)] The "big-endian" vs. "little-endian" distinction, an allusion to Swift's \textit{Gulliver's Travels}, refers to whether the byte addressed by the address of an integer contains the most significant bits (big-endian) or the least significant bits (little-endian). This allusion was first made by D. Cohen in [2].
\end{itemize}
5.1.4 Identification

We ordinarily think of a remote procedure as being a service or part of a service that is being made available to potential clients. If you have developed such a service, how do you arrange so that your service can be unambiguously identified? Putting this another way how does a caller determine that it has contacted the appropriate callee, and vice versa? Note that this is a different problem from that of locating a service. There may be many instances of a remote procedure (or, more generally, of a remote service, supplying multiple entry points), all at different sites. Each of these instances should have the same identity, however such an identity is specified.

What all approaches for specifying the identity boil down to is associating with each service a unique integer. In DCE RPC such integers are known as UUIDs and are generated automatically. In other RPC approaches, such as Sun RPC, there is no such generator; identifiers are either assigned by a central authority (e.g., Sun) or supplied by the programmer (with the programmer guaranteeing that the integer is really unique).

5.1.5 Transport

The transport protocol is responsible for the actual transmission of data between the caller and the callee. Some protocols, such as the transmission control protocol (TCP), provide reliable transmission of data. This means that, as long as communication is not hindered by line outages, etc., the protocol makes certain that what is transmitted by the sending application is received by the receiving application unmodified (i.e., no bytes are changed, added, or deleted, and all bytes are received in the order in which they were sent). Other protocols, such as the user datagram protocol (UDP), do not guarantee reliable transmission. Since such reliability is essential for the desired RPC semantics, the RPC implementation itself must provide it.

5.1.6 Exceptions

As with local procedure calls, something may go wrong while a remote procedure is being executed. The problem might be reported to the caller via a result parameter or the return value. However, a number of potential problems are typically handled via some sort of exception mechanism. For example, on UNIX systems, an arithmetic problem is reported to the program via a signal. If the signal is generated on a remote machine, something must be done to propagate it (or its equivalent) back to the calling machine. The intent is to make the behavior in response to exceptions for the remote case identical to what it would be in the local case.

Signals may or may not be the desired exception-handling mechanism. Even on UNIX systems, we might want an exception mechanism whose semantics differ from those of the signal mechanism.

5.1.7 Failures

Situations arise with remote procedure calls that do not arise with local procedure calls. In particular, with local procedure calls, if the host computer crashes, then we have lost everything and there is no thought of recovering—nothing remains that can perform any recovery actions. (While some state information may be left on nonvolatile storage, say on disk, that can used once the computer comes back to life, this is beyond the scope of an RPC mechanism.) In the remote case there are three independent types of failures: the remote computer might crash, the local computer might crash, or the communication path between the two machines might be lost. In addition, of course, the remote procedure may itself make remote procedure calls, complicating the possibilities for failures even further.

If the local computer crashes, the effect should be analogous to the case in which the only computer crashes during a local procedure call—all activity related to the call ceases. But for this to happen, some sort of notification must be sent to the callee telling it to abort. We could
simply terminate the callee, but we might want enable the callee to quit what it is doing gracefully, perhaps continuing its execution until it gets to a convenient stopping point or calling a special cleanup routine.

If the remote computer crashes, we need to inform the caller, perhaps through an exception mechanism. What is relevant to the caller is how far the call progressed before the crash. Perhaps the crash occurred before the call even started. In this case, the caller may simply retry the call, either on the original remote computer when it restarts or on some other computer. On the other hand, the call may have started before the crash took place. The call may have even completed before the crash—the results were just about to be transmitted back to the caller. Unfortunately, it is not possible for the caller to determine when the crash took place. This information is known only to the callee, and the assumption is that everything was lost when the computer crashed. (Information may of course be stored in non-volatile storage that can be used to determine what happened, but such use of application-specific information is beyond the scope of a general RPC mechanism; it could be used, however, by the application itself.)

What to do about this uncertainty becomes an issue of semantics. What we would like is what is called exactly-once semantics—if the caller places a call, that call executes exactly once on the remote machine; if there is a crash, then the RPC implementation figures out whether the call should be repeated. As we have just argued, however, this is generally not possible. So we must weaken the semantics so that we have something that is possible to implement. One such weakened semantics is known as at-least-once semantics—the caller continually repeats the call until it gets a sure indication it was completed. Another semantics is known as at-most-once semantics—if the callee machine crashes, the call is aborted and the callee is notified. It is then up to the callee to determine what to do next.

At-least-once semantics has the advantage that the calling program is not complicated by considerations of failure. However, there is of course the possible disadvantage that the call may actually take place on the remote machine more than one time. In many situations this presents no problem, but in some situations it is most definitely a problem (consider a banking application in which the remote procedure transfers money from your account to mine). With at-most-once semantics, the caller is protected from unintended repeats of procedure calls, but it must provide the code that determines what to do in the event of a crash.

The third potential failure is the loss of the communication path while the caller and callee remain up. Neither the caller nor the callee can communicate with the other so both consider the other to be down. As long as the caller does not repeat the call on another machine, then when it finally regains communication with the callee machine and repeats the call, there is (or may be) information on the callee machine indicating the state of the original call. Thus if the call had completed the first time, the callee should be able to (finally) transmit the results to the caller, while if the call had aborted, the callee can supply the appropriate notification to the caller.

5.1.8 Security

A final general concern is security. For the local case, security, at least in the interaction between the caller and the callee, is not an issue—the security identity of the executing thread of control is of course the same in both caller and callee, and we assume that no third party can interfere in any way. But the situation is very different in the remote case. If the remote procedure is performing an action that requires some sort of privilege, the remote machine must have a reliable means of determining the identity of the caller. Similarly, if the remote procedure is supposed to perform some important action, the callee must be assured that it has actually contacted the correct server and not some impostor. Furthermore, a third party (the "enemy") may be trying to learn sensitive information by eavesdropping, or may be attempting to interfere actively by modifying the data being transmitted.
Providing security is certainly possible but can be expensive, in terms of both the compute time required and the extra data that may have to be transmitted. The application writer should be allowed to make a tradeoff between expense and security through the provision of levels of security. Authentication, i.e., proving one's identity to another party, is relatively cheap to accomplish, but the other forms of security are not. Thus, if authentication is considered a sufficient level of security, it can be provided without the cost of the other levels.

5.2 Naming and Directory Services

As discussed above, we need some sort of support for a name service—a database that maps service names to collections of servers providing the service. In DCE, such a name service is provided as part of a more general directory service which establishes a hierarchical name space and associates a variety of information with each name. DCE actually provides two such services: CDS (cell directory service), handling the special needs of DCE itself, and GDS (global directory service), an implementation of the international X.500 standard.

For administrative and naming purposes, DCE is organized into large units known as cells. A cell is a collection of clients and servers and is intended to mirror organizational structure: a cell might be an entire company or university, or perhaps a division, site, or campus. Each cell is managed by a single administration that is responsible for cell-wide policies and security.

CDS provides a hierarchical naming scheme within each cell. A name represents a path through a tree, in much the same way as files are named in most systems. Any sort of entity can be named in this way, for example, services, servers, hosts, etc. CDS is implemented as a replicated, distributed database, providing complete service even if some of the sites implementing it are down.

The collection of all cells is a forest of naming trees. Individual cells can be named in a number of ways:

- via the Internet name service—known as the domain name service (DNS)
- via X.500
- via CDS (this feature will not be implemented until Release 1.1 of DCE, due out in mid- to late 1994)

Within a cell, the paths defined by the CDS naming tree can lead to other name spaces. For example, both DCE Security and DFS have their own private name spaces. A file within DFS is identified by supplying the name of the cell, the name of the DFS namespace within the cell, and the name of the file within the DFS namespace. This is much easier than it sounds, because the syntaxes of the various names are "blended" with one another. Thus such a file might be identified as /../osf.org/fs/u/twd/file. In this name, "/../osf.org" identifies a cell within the Internet name service, "//fs" identifies the root of the DFS namespace within the cell, and "u/twd/file" identifies a file within the DFS namespace.

5.3 DCE Security

DCE security consists of three related services:

- An accounts database that maintains information about user accounts, etc. for use throughout the cell.
- An authentication service that provides the means for a client and a server mutually to authenticate each other and to insure that data is transmitted securely.
• An authorization service that manages the use of ACLs (access control lists), allowing servers to determine if clients should be allowed to perform the operations they have requested.

The authentication service is based upon MIT’s Kerberos system [8]. This is a private key approach in which one or more trusted servers store a copy of each principal’s private encryption key. (A principal is some entity, such as a user or a server, than can enter in security negotiations. The encryption key, at least for human principals (users), is derived from the password). Kerberos provides a means by which two principals can prove to each other that each knows its own encryption key. It goes on to provide to the pair of principals a new encryption key, known only to the two principals, that can be used for further protection, such as data integrity (assurance that transmitted data is not being surreptitiously modified), and data privacy (assurance that transmitted data is not being surreptitiously read).

The accounts database stores account information (including encryption keys) for all the principals in a cell, along with information describing protection groups. One can “log in” to DCE—this process involves both the authentication service and the accounts database: one establishes one’s credentials, i.e., proves his or her identity, using information provided by the user and information obtained from the accounts database. At no time is the unencrypted version of one’s password ever transmitted over the communications network.

When one contacts a server, i.e., places an RPC to it, the RPC package automatically invokes DCE security and includes with the request an copy of one’s credentials, encrypted by the security service in the encryption key of the server. The RPC runtime package on the server then decrypts these credentials and thus learns who the client is. The server can then invoke the DCE authorization service (which is linked into the server application) to determine, via access control lists, whether the client should be allowed to perform the requested operation.

5.4 DFS

DFS, the DCE distributed file system, is based upon AFS (the Andrew file system) from Transarc. Like AFS and Sun’s NFS, DFS supports the transparent access to files within a distributed environment. Like AFS, it takes advantage of local disk space on client computers to form a cache of recently accessed file data. For read-only and private files, this can dramatically reduce the use of the network and the load on the servers. Furthermore, unlike both AFS and NFS, the semantics of operations on files over DFS are nearly identical to the semantics of such operations on local file systems. DFS fully supports DCE ACLs for files, assuming that the underlying local file system also supports ACLs.

In addition, DFS supports a number of features that are not so visible to users. Read-only collections of files may be replicated, i.e., redundantly stored on a number of different servers. If the server currently being accessed by a client crashes, then the client’s file accesses are transparently switched to another server. Techniques for automatic and semiautomatic replication are provided.

The DCE local file system (LFS) is a nondistributed file system that can be used in conjunction with DFS. It provides full ACL support for files and directories. It supports a fast duplication technique, known as cloning, in which a logical (but not physical) copy is made of a collection of files. This collection can then be safely copied, perhaps to a backup medium, while the original copy goes back to normal use.

LFS also is a log-based file system, so that little time is required to recover it after a crash. This is accomplished by maintaining a log of all changes to what is known as meta-data, i.e., data for use by the system only that describes the contents of a file system. If a crash occurs, rather than perform lengthy consistency checking as in standard UNIX file systems, the contents of the log are "played back," a procedure which takes a few seconds rather than many minutes.
5.5 Whither DCE?

DCE contains an enormous amount of code—it is many times the size of the kernel of many modern operating systems, even monolithic implementations. Its first releases have been what are euphemistically called “functional” releases—meaning that they demonstrate the functionality of the product but are neither fast enough nor reliable enough to be used in production. Will it ever become mature enough to be used in production?

A number of companies are betting that it will be. It is a required piece of COSE, the Common Operating System Environment—a coalition of UNIX vendors formed in response to Microsoft’s NT. Even Microsoft appears to be giving at least portions of DCE its de facto support—for example, the RPC package used by NT is a extension of DCE RPC and is interoperable with it (as long as no Microsoft extensions are used).

OSF has released its fourth minor release of DCE (1.0.3) and is scheduled to release the next major release, 1.1, in mid- to late 1994. Whether this release will be commercially successful remains to be seen. If so, it will represent a major step towards making open software in a distributed environment a reality.

6 References


Broadband Telecommunications Networks and Asynchronous Transfer Mode: a Survey

G. Ciccarella, T. Tofoni
Scuola Superiore G. Reiss Romoli, L'Aquila (Italy)

Abstract

Telecommunications networks are in the process of a profound mutation under the influence of two main factors:

- developments in photonic and electronic technology are making it possible to transmit and switch connections of very high bandwidth;

- an expansion in user demand is creating the need for a variety of more sophisticated and powerful telecommunications services such as High Definition TV, supercomputer networking, remote visualization and virtual reality, which require high performance networks capable of supporting information transfer between user interfaces at rates in excess of hundreds of Mbit/s.

The introduction of Fast Packet Switching techniques such as Frame Relay and Switched Multimegabit Data Service (SMDS), currently being offered by many public operators, is one manifestation of this mutation. Frame Relay and SMDS cover a data rate speed range between the classical 64 kbit/s up to 45 Mbit/s which is insufficient for many of the services we have mentioned above.

A further and more advanced manifestation of the ongoing mutation of telecommunications networks is the planned Broadband-Integrated Services Digital Network (B-ISDN). B-ISDN is a standardized public switched telecommunications network infrastructure capable of supporting both broadband and narrowband services on a single flexible network platform and providing customer access over a single family of interfaces.

To meet these requirements, a new multiplexing and switching technique called Asynchronous Transfer Mode (ATM) has recently been defined by ITU-TS (International Telecommunication Union - Telecommunication Standardization).

The purpose of the paper is to present the factors influencing telecommunications networks evolution, to give a status overview of broadband reference technologies and services, and to shed some light on the principles underlying ATM technique along with why and how it offers advantages when compared to existing networking solutions.

1. Factors Influencing the Evolution

The rapid transformations in the telecommunications world are characterized by two major driving forces: technology innovation and increase of demand for services, both affecting, and being affected by, the rapid changes in the market scenario, where demand and supply for telecommunication services meet, [1].

The process of telecommunication network evolution is therefore strongly influenced by complex interactions among all the relevant technological and marketing factors, which in turn modify the demand/supply balance and often result in conflicting and unstable requirements. Within this paradigm, the major growth of the demand is coming from
business users. Their requirements are felt more comprehensive, urging and demanding, and influence the overall development of industry and economy; on the other hand, the needs emerging from residential users can still be adequately met through the modernized telephone network infra-structure.

From the business user stand-point, the development of industrial and trade processes involves a growing exchange of information among different companies and within multi-site organizations. The following critical factors emerge: i) the need to interconnect a growing number of systems (e.g. workstations and super-computers, local area networks LANs, imaging and automation systems) through telecommunication networks, ii) the need to achieve a wide geographical connection capability, ideally with the removal of distance, bandwidth, and quality constraints, and iii) the need of quickly getting the communication links when a company adopts new systems, or expands the power of those in use, or moves to another site. In practice, these users require ubiquitous access to "high-speed data services" that the public networks of today are not able to provide effectively.

From the public network side, the huge complexity of the telecommunication network, the large investments needed, the long operational life of most of the installed equipment and systems, and the subsequent coexistence in the network of systems of different generations are all elements asking for a smooth introduction of technology innovation, to reduce costs related to investments and operations. In the area of broadband systems and services, considerable effort has been devoted to identify and define the target solution (i.e. Broadband Integrated Service Digital Network, B-ISDN). To achieve this objective the intermediate evolutionary steps, [2], are fundamental to cope urgently with high-demanding market segments, since in the past several years, the direct support of information in-house systems (host and LANs) was missing in the public network offerings and the more advanced users created the market-niche for the so-called "informatic networking", which is basically different from "networks".

A plethora of vendor-specific communication solutions, built around mainframe computers and LANs were introduced, e.g. bridges, routers and gateways, mainly suited to point-to-point communications over leased lines. These proprietary solutions, do not meet the real demand of ubiquitous access and generalized connection capability. Due to their limited compatibility, poor performance, and lack of mechanisms for comprehensive management, they cannot be regarded as a platform for the introduction of new services.

The difficulty of attaining compatibility and integration capabilities is further increased by deregulation and competition; in such a scenario, fairly new for telecommunications in Europe, standardization is playing a key role to achieve convergence towards common solutions. For the Public Telecommunications Operator (PTOs) the reference to widely accepted standards becomes a must, while for the customer it represents the only way for achieving adequate and cost effective solutions. Section 2 of the paper is aimed at presenting the technologies which are felt to be fundamental for the evolution of the network. These technologies are seen in the B-ISDN framework; they include the Synchronous Digital Hierarchy (SDH), Metropolitan Area Networks (MAN), Asynchronous Transfer Mode (ATM), and Telecommunications Management Network (TMN). As for the high-speed service offer, attention is given to
permanent and semiPermanent connections, Broadband Virtual Private Networks (B-VPN), Switched Multi-megabit Data Service (SMDS) and Frame Relay.

Section 3 presents networks evolution path and highlights several initiatives undertaken in Italy. In particular, the aspects related to the exploitation of emerging technologies are discussed, including the evolution of the transmission infrastructure, cabling systems, users gathering solutions and high-speed services. Moreover, early provision of high-speed data services at a metropolitan, as well as extended range is analyzed.

Section 4 gives an overview on the fundamentals of ATM technique. In particular, attention will be focused on the cell structure and the protocol reference model. A brief outline of source modelling, multiplexers performance, traffic control issues and ATM switching will also be given.

2. Reference Technologies and Evolution Trends

2.1 Reference Technologies

A common view has emerged [1], about the key elements of the B-ISDN scenario, i.e. SDH, MAN, ATM, TMN. Figure 1 depicts a B-ISDN scenario, which includes the above elements. In the following a brief overview of the mentioned technologies is given.

![Diagram of B-ISDN scenario including ATM and MAN technologies.](image)

**Fig. 1** - A picture of the B-ISDN scenario including ATM and MAN technologies.
SDH

The concept of SDH [3] was pioneered as SONET (Synchronous Optical Network) in the American National Standards Institute (ANSI) and made possible by the availability of new technologies such as optical fiber transmission, software, microelectronics and very accurate clock devices. Qualifying features which make SDH well suited for public networks are: synchronous multiplexing with payload capacity in excess of 150 Mbit/s and enhanced Operations, Administration, Maintenance and Provisioning (OAM&P) capabilities.

These features not only enable SDH to satisfy the basic criteria for the introduction of a new technology, but also allow for a backward compatibility with the widespread Plesiochronous Digital Hierarchy (PDH) and are suitable to serve as a transport vehicle for ATM and MAN structured signals. For the above features and the provision for transverse (multi-vendor) compatibility, SDH represents the basic choice for layer 1 of B-ISDN.

MAN

Metropolitan Area Networks (MANs) [4] have been in the recent past the subject of a growing attention world-wide by both telecommunication manufacturers and operators. This sector is now at the forefront of market policies for early provision of high-speed communication services.

MANs can extend LAN-like performance to a larger area, benefiting from the use of more assessed LAN-derived technology: it is therefore feasible for MANs to be in widespread operation before other broadband techniques are fully available. MANs are being designed to exploit the high-capacity of optical fiber transmission systems, both the currently available plesiochronous (PDH) and synchronous (SDH) ones: this is being reflected into the definition of appropriate protocol mapping procedures. This feature, along with the support of "technology independent" bearer services, make MANs able to cope with the evolving network and service scenario. Several MAN segments can be integrated into a global broadband environment as access infrastructures.

In addition to manufacturers and users, Public Telephone Operators (PTOs) have played an important role and contribute to the positioning of MANs in the public broadband network environment, that resulted in several basic concepts been clearly established, i.e.:

- the concept of publicly-operated MAN offering early provision of high-speed data services to business users;
- the MAN architecture and the bearer services to be supported, with the connectionless data service deserving first priority;
- the use of a medium access technique based on the Distributed Queue Dual Bus (DQDB) protocol, as defined by IEEE 802.6;
- the use of MANs with different transmission systems (both the PDH and SDH families).
The planned technological evolution includes in the next future provision of isochronous service, enhanced management capabilities, protocols for direct MAN interconnection and the compatibility with 622 Mbit/s SDH systems.

ATM

ATM can be considered as an efficient form of packet switching, handling fixed size, short packets called cells. Its most important characteristic is the flexibility in bandwidth allocation through statistical multiplexing of different streams of data, allowing every bit rate, up to the link speed, to be arranged between the user and the network. The capability of supporting a wide range of services, including real-time, high bit-rate and multimedia services on network infrastructures based on optical fibers, the high degree of flexibility, in conjunction with embedded management facilities, makes ATM the reference technology for the target broadband service scenario. The fact that ATM is being developed in conjunction with B-ISDN has brought to several significant refinements of the original concept, which make it more suitable for introduction in the evolving telecommunication network. Among these, the possibility to use the transmission facilities of SDH, the definition of Connection-less service and the use of Virtual Paths (VP) are the most important. The VP concept can be exploited for the implementation of very flexible ATM Digital Cross Connects (DXCs), which are foreseen in the first introduction step of ATM technology in the network. The ATM support to broadband services can be given on permanent / semi-permanent basis via ATM-DXCs and, in the future, via ATM on-demand switched connections, when signalling capabilities will be available.

ATM was developed as an international standard for wide area telecommunications networks, however its attractiveness for LANs has resulted in the development of ATM LANs that can be used for the following purposes:

- client-server computing with high-end workstation and servers (Work group ATM)
- connecting existing hubs, bridges and routers (Backbone ATM)
- connectivity with widearea ATM networks.

The simplicity of the algorithm for forwarding ATM cells allows an efficient hardware and software implementation. Moreover ATM LANs can utilize any combination of speeds and transmission media (including coax, twisted pair and fiber) and then it is extremely "open" in comparison to the other competing technologies. In fact being based on a network of switches and dedicated links to each host the aggregated bandwidth of an ATM LAN increases as hosts are added. The network cannot be saturated like networks based on shared medium (e.g. DQDB).

TMN

Integrated network management capabilities is of major concern for the PTOs, in order to provide an high degree of flexibility in services deployment and a quick answer to
market demand. Moreover it can satisfy requirements for system openness and interoperability among the management systems of heterogeneous networks, to support the network operator in the administration of communication resources. The basic concepts are defined in a set of recommendations for a Telecommunication Management Network (TMN) [5].

TMN architecture has been assumed for the management of SDH, ATM and also for MANs. Specifically it has been taken into account by ETSI, in the definition of harmonized MAN management principles, aimed at ensuring complete integration of MAN technologies in the future global network [11].

In this context, the OSI's approach is widely used since it guarantees the means for exchanging management information independently from the technology used. A particular reference is made to the tools necessary for building a network management system, e.g. the use of Object Oriented (OO) methodologies. This conceptual approach stimulated ITU-TS to produce recommendations capable of making OSI management standards applicable in a real environment and used by manufacturers when developing new equipments.

2.2 High-Speed Data Services: SMDS and Frame Relay

While bearer service specifications define how the network carries the user traffic, satisfying several requirements such as bandwidth, error rate and delivery-time delay for the different type of connections, the following types of broadband services have been identified:

Permanent and Semipermanent Connection Service.
They support high speed data transfer over connections established by management procedures.

Connectionless Service (CL).
It performs transfer of variable-length data units that can tolerate variable end-to-end delivery delay but require error detection functions, as LANs (e.g. the IEEE 802 family) operate. This service is viable for LAN interconnection and for computer to computer communications.

Connection-oriented (non isochronous) Service.
It supports the transfer over a previously established virtual channel, of segmented information flows, where the segments (cells or frames) have no specified inter-arrival time. This service can be viewed as compatible with ATM in B-ISDN, as well as with Frame Relay in packet switched network.

Connection-oriented Isochronous Service.
It supports the transparent transfer of data over a previously established connection with guaranteed bandwidth and constant end-to-end delay, but without error detection capabilities. This service provides stream-mode access to the upper layers of the network. This deterministic access to the communication resources can effectively be exploited by applications with real-time need and high continuity degree, like voice and
video. Among the emerging high-speed data services, the major examples worth to be mentioned are: SMDS, Frame Relay, B-VPN.

The *Switched Multi-megabit Data Service* (SMDS), defined by Bellcore (Technical Reference TSV-772, May 1991), is a public high-speed connectionless packet-switched service suitable for LAN to LAN interconnection; it aims at extending the LAN performances across metropolitan areas and even at longer distances. While the service is specified independently of the network technology, it is well suited to be provided within MAN DQDB-based, environment and extended later in the ATM scenario. SMDS principles are being taken as reference within ETSI for the definition of the Connectionless Broadband Data Service (CBDS).

*Frame relay* is a connection-oriented non-isochronous packet service based on a revisited version of the HDLC data link layer procedures; it is conceived to be run on packet switched networks by using simplified protocols between the intermediate nodes, instead of using the cumbersome X.25 procedures. This is made possible thanks to the increased reliability of today’s transmission systems; protocol simplification, rather than new technology introduction, is the rationale for the "speed" of this service. The service needs the establishment of permanent virtual circuits.

Broadband Virtual Private Network (B-VPN) gives a business user, having a company structure spread over a country or internationally, high speed data communication features for LAN interconnection similar to a private network by using public network capabilities. In particular, the B-VPN service can be based either on semi-permanent or on switched connections. In the former case, within an ATM environment, traffic related virtual channel dimensioning can be provided by the public network operator in order to efficiently use network resources and to minimize user costs. Furthermore, sophisticated customer control features can be provided to cope with specific and variable user needs.

### 3. Network Evolution Paths

A general model for telecommunications networks can be given by considering three interconnected layers (fig. 2). The top level is related to the management functions, that is operation support, customer control and service creation; the middle level is related to the intelligent functions, that is mobility, personalization and security; the lowest level is related to transport functions, that is wireless and broadband communications.

The evolution of the TLC networks can be arranged along three axes, correspondent to the levels of the previous model: transport, control and management (fig. 3).

* Along the transport axis, the evolution is from the ISDN towards the Broadband Networks by the use of MAN and then of the ATM.

* Along the control axis, the evolution is from the Intelligent Network (IN) towards the Advanced IN (AIN).

* Along the management axis, the evolution is from the heterogeneous TMN towards the integrated TMN.
The evolution paths of IN and TMN will likely converge to an architecture that integrates both control and management capabilities. This architecture is called TINA (Telecommunications Information Networking Architecture) and has the following technical objectives: to enable the rapid introduction of information networking services by promoting application modularity, reusability and portability, to provide the ability to handle multimedia, multisession and multipoint communications; to define user-to-network interfaces that are synergistic with the evolving customer equipment capabilities; and to separate the transport technology aspects of the equipments from the generic control and management aspects.

Fig. 2 - Telecommunications Networks Layers.

Fig. 3 - Evolution dimensions for telecommunications networks.
In conclusion the following statements can be expressed:

- business of all shapes and sizes are now making extensive use of telecommunications to improve competitiveness;
- this process is eased by the improvement of the information technology;
- computing, telecommunications, broadcasting are converging into the development of common applications;
- public networks are playing a strategic role since they provide the infrastructure for all the applications.

Sections 3.1 and 3.2 address the evolution steps foreseen by the Italian PTO. With reference to the network evolution in Italy, the "tactical" introduction of specialized infrastructures added to the existing ones is foreseen. First priority is presently on existing demand sectors, mainly high-speed data service (2 Mbit/s and over), and above all for Host and LAN interconnection. The attitude towards a diversified offer is even overlapping in some cases: a typical example is the 2 Mbit/s service envisaged on the Italian public packet network (ITAPAC) in the near future.

Based on a well known "offer map for data services" originated by US Regional Bell Operating Companies (RBOCs) a similar map has been drawn for the Italian situation (fig. 4).

The figure, which shows the bandwidth-versus-usage convenience areas with reference to the present telecommunication options, gives a rationale to a multi-facet offer; such a scenario, driven by cost/benefit criteria and real customer demand, will be characterized by a dynamic coexistence of different (both traditional and new) choices, among the new ones MAN and ATM will have a role (fig. 5).

![Offer Map for Data Services](image_url)

**Fig. 4** - Offer map for data services (SIP Elaboration from data provided by RBOCs).
3.1 Transmission Infrastructure Evolution

The broadband network infrastructure will be built up by gradually incorporating the emerging technology and taking profit of cabling strategies specifically suited to optical fibers. The pervasiveness of optical fibers, at different network levels, will also allow for an easy build up of suitable configurations for cabling the large business users community (as a first-priority goal), as well as to gradually extend the connection capability, while considering opportunities for other services, (e.g. distribution services). The deployment of SDH, in Italy, can be foreseen in three main phases. A first phase (1992) relies upon the deployment of the first SDH rings in metropolitan areas; then SDH will start interconnecting large business users on wider geographic range; transmission network management functions will be developed based on TMN principles.

The following phase (1995 on) will see SDH deployment in all new transmission system; a large spreading of ring networks in urban and large regional contexts will be achieved; TMN-based transmission network management functionalities will be consolidated. Eventually, in a longer time perspective (1997 on) the freezing of PDH systems and a pervasive SDH deployment will be carried out at all network levels.

As far as the cabling aspects are concerned, the diffusion of fiber optics in the distribution area allows for the introduction of new strategies for the customer premises
connection. In the Italian network more than one million kilometers of optical fibers have been deployed by 1992, with a growing portion in the access network.

Exploiting optical cables features, the operator orientation is to substitute the present offer of bare transmission media with the provision of a "portion of bandwidth", taken from the overall bandwidth of the distribution network. A first fundamental improvement with respect to the traditional installation methodologies can be obtained by utilizing Digital Loop Carrier (DLC) systems, which may roughly be considered as concentrators or multiplexers; they permit to effectively extend the offer of optical fiber up to the locations of particularly demanding subscribers. DLCs are situated in proximity of user premises and are connected to the Local Exchange through optical fibers at 34-150 Mbit/s; starting from a DLC the user can be connected either with traditional copper cables (Fiber To The Curb, FTTC approach) allowing for digital flows from e.g. 64 to 2 Mbit/s, or with optical fiber (Fiber To The Office, FTTO approach) to carry higher speed traffic.

3.2 Transport and Services Evolution

Given the priority on high-speed data services, the use of MANs is being promoted by the Italian TLC operator since it is an available technology that allows a suitable response to communities of users already existing as such and calling for an advanced network solution.

MANs have been installed in urban areas with high density of business users and will provide connectionless data services. A limited number of user premises will be interconnected, selected among big companies. In this phase, remote MANs can be interconnected by means of dedicated or semi-permanent links.

In the meantime, ATM has been consolidated: the basic standards have been produced by international bodies, industries are going to release the first products, the most advanced countries have carried out field trials and are planning its introduction in the public network, while user ATM based products are foreseen in the very short term.

From a technical viewpoint, the evolution and extension of high-speed services to wider ranges stands on three major factors impacting the feasibility of above transition:

- services need to be defined independently of the specific network technology (MAN, ATM or other),
- management features should grow in harmony,
- the interconnection among distributed MAN segments and centralized ATM nodes (ATM DXC) has to result extremely efficient, avoiding discontinuities and bottlenecks.

To this purpose it looks encouraging that the above items are taken as guide-lines for the on-going standardization activity, namely a system independent specification of services (e.g. CBDS in ETSI) and the interoperability between MAN and ATM both in ITU-TS and ETSI.
4. The ATM Technique: an Overview

The future broadband network will have to handle services that require a bandwidth varying from a few kbit/s to tens of Mbit/s (e.g. High definition TV). Therefore, the choice of the type of architecture to be implemented will by necessity involve a compromise between the classic techniques of circuit and packet switched networks.

ATM combines the advantages of both classical circuit- and packet-switching techniques allowing all services to be transported and switched in a common digital format which employs fixed-size packets (cells). Besides, to achieve the necessary flexibility, cells from different sources are multiplexed together on a statistical basis.

ATM offers a number of advantages over current network architectures, such as support for complex multimedia applications, high performances (i.e., low transfer delays), flexibility (i.e., support of dynamically varying bandwidth requirements), compatibility with existing networks investments, ability to integrate LAN, MAN and WAN technologies.

4.1 ATM General Features

ATM is based on the following two fundamental principles:

- adoption of information units (cells) of fixed length of 53 bytes, of which 5 used for the header and the remaining 48 for users information;
- assignment of transmission bandwidth on demand.

The basic characteristic of the B-ISDN network, as defined in ITU-TS Recommendation I.121, is the use of ATM technique. ATM is applied to the transport, multiplexing and switching of information.

The fundamental principle of the ATM technique is the segmentation of various types of information flow (discontinuous as in data transmission, or continuous as in telephone and video signals) into a sequence of elementary cells that are introduced in the network to be transmitted and switched.

Cells are assigned on demand according to the activity of the source and to the availability of network resources. As in circuit switching, in the ATM technique there is a separation between the signaling and information transfer phases. In other words, after the call set-up phase, the network is completely transparent with respect to the information field, that is, transit nodes do not act upon the information fields but only on the header. In this way it is possible to reduce the transit time of the cells at the switching centres obtaining as a result a shorter end-to-end delay.

The fundamental difference between ATM and circuit switching lies in the fact that
ATM is based on a statistical assignment of resources, while circuit switching is based on a deterministic assignment\(^1\).

![Fig. 6 - ATM multiplexing.](image)

Just like classic packet switching, ATM is based on celle statistical multiplexing, i.e., cells from different traffic sources share the same physical medium (fig. 6). However, two important factors differentiate ATM from packet switching:

- traffic control, that in ATM occurs only at the user-network interface;
- information content error control, that in ATM pertains to higher levels protocols.

The elimination of error control has been introduced to minimize end-to-end delay and is made possible by the low bit error rate typical of fiber optic transmission.

4.2 ATM Cell Structure

As mentioned in the introduction to this section, the ATM cell is 53 bytes in length, of which 48 are reserved for user information and 5 for the header (fig. 7).

![Fig. 7 - ATM cell structure.](image)

\(^1\) If \(\lambda_p\) and \(\lambda_m\) are the peak and average bandwidth respectively and \(C\) the channel transmission speed, a deterministic assignment allows the connection of up to \(N\) calls, where \(N\) is the maximum integer that satisfies the condition:

\[
\sum_{i=1}^{N} \lambda_{p,i} \leq C
\]

while a statistical assignment allows the connection of up to \(N\) calls, where \(N\) is the maximum integer that satisfies the condition:

\[
\sum_{i=1}^{N} \lambda_{m,i} \leq C
\]
As far as the header is concerned, there are two different ATM cell formats, one at the user-network interface level and the other at the network-node interface level. Header formats of this two type of cells are reported in fig. 8.

![Header format of the ATM cell at the user-network interface, (a), and network-node interface, (b).](image)

The 5 bytes header is divided as follows:

- **VCI (Virtual Channel Identifier)** and **VPI (Virtual Path Identifier)**
  
  These are two fields assigned for routing. Both have the meaning of a logical and unidirectional association between the two ends of a physical connection. That is, if terminal A communicates with terminal B, the VCI assigns different values for each direction of transmission.
  
  The VCI is assigned to a connection in the call set-up phase and is released when the connection is cleared. It assumes different values link after link (and in the same link it has different values in each direction of transmission) and therefore it is not the same for the whole route.
  
  The VPI identifies a set of virtual connections and, therefore, a whole group of virtual connections may have the same VPI (fig 9).

![Relationship between VCI and VPI identifiers of ATM virtual connections.](image)

**VP = Virtual path**

**VC = Virtual channel**

![Fig. 9- Relationship between VCI and VPI identifiers of ATM virtual connections.](image)
Generally VPI are assigned on a contractual basis; this makes possible the realization of a service similar to the classic PVC (Permanent Virtual Circuit) of X.25 packet switched networks.

- **GFC (Generic Flow Control)**
  It is a 4 bits field found only at the user-network interface. It is used to control cells flow when in presence of network architectures in which the transmission medium is shared by several terminals. The purpose of the GFC field is to prevent the rise of dangerous congestion situations.

- **PTI (Payload Type Identifier)**
  It is a 3 bits field used to distinguish user information from data produced by the network for management and supervision purposes.

- **CLP (Cell Loss Priority)**
  It is a 1 bit field used to define a two-level cell priority. In case of congestion, if the cell has low priority, it will be first to be discarded. The CLP field can be managed by the user or some by other element of the network.

- **HEC (Header Error Check)**
  It is an 8 bits field used to correct errors. It makes possible to correct single errors and to detect multiple errors. It uses the 8th degree polynomial $X^8+X^2+X+1$ to detect errors. The operation of the HEC algorithm at the receiver is depicted in fig. 10.

![HEC Operation Diagram](image)

**Fig. 10 - HEC operation at the receiver.**

### 4.3 Protocol Reference Model

As far as protocol models are concerned, the ITU-TS Recommendation I.321 has agreed upon the standard for a reference model that includes a User Plane, a Control Plane and a Management Plane. The Management Plane is divided in two parts defined respectively Layer Management and Plane Management (fig. 11).
Fig. 11 - B-ISDN Protocol Reference Model.

Besides, for each plane (with the exception of Plane Management), there are three levels:

- Physical Layer;
- ATM Layer;
- Adaptation Layer.

The User Plane transfers user information. The Control Plane transfers and manages signal information. Layer Management regulates the operation and maintenance of information flows and in particular it can regulate meta-signalling procedures, that is, the ATM signalling virtual channel assignment procedures. Plane Management has the function of managing resources and coordination of intra-frame operations. Unlike the other planes, it lacks a layer structure.

As far as layers are concerned, ITU-TS at the moment has agreed upon standards only for functions at the lowest layers.

As described in ITU-TS Recommendation I.432, the Physical Layer is divided into two sublayers:

- PM (Physical Medium Sublayer)
- TC (Transmission Convergence Sublayer).

All functions pertaining to the physical medium, such as for example optoelectronic and electro-optic conversion, are grouped in the PM sublayer. All the functions necessary for the adaptation of a cell flow into a bit flow that can be transmitted and received by the
system in use, are performed in the TC sublayer. For example, the generation of HEC during transmission and verification at reception occur at this sublayer.

The ATM level, described in the ITU-TS document I.361, includes functions that are independent from the physical medium. They are:

- ATM cells multiplexing and demultiplexing;
- VPI/VCI values translation;
- cell header generation and extraction.

The ATM level acts only and exclusively on the label (thus it is transparent with respect to the information field) and has the task of translating VPI and VCI values on the basis of tables found at the switching nodes that are defined at the call set-up phase. Besides, ATM level acts also, at the user-network interface, on the GFC field to control cells flow.

The ATM Adaptation Layer is specified in ITU-TS Recommendations I.362 and I.363. Functions expected at this layer are grouped in four types of services (fig. 12), identified on the basis of the following three parameters:

- temporal relation (between source and destination);
- bit rate;
- connection mode (connection oriented, connectionless).

<table>
<thead>
<tr>
<th></th>
<th>CLASS A</th>
<th>CLASS B</th>
<th>CLASS C</th>
<th>CLASS D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timing relation</strong></td>
<td>Required</td>
<td>Not Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>between source and destination</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bit Rate</strong></td>
<td>Constant</td>
<td></td>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td><strong>Connection mode</strong></td>
<td></td>
<td>Connection-oriented</td>
<td>Connectionless</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 12 - ATM Adaptation Layer functions for different services.

During transmission the adaptation layer performs the segmentation of user information in ATM cells, while during reception it performs the reverse function, that is the reassembling (however, the task of adding during transmission and subtracting in reception the 5 bytes long header is performed at ATM layer).

4.4 ATM Traffic Control

The ATM technique has the advantage of great flexibility (that is, capability of managing on an integrated basis a wide variety of services), and efficiency (that is, high
usage of transmission bandwidth). On the other side of the coin is, though, a greater complexity in traffic management. As for every telecommunications network, the major problem facing ATM networks is avoiding the rise of congestion. Indeed, in severe congestion situations, the network is no longer able to guarantee the quality of service (as given by end-to-end-delay, cell loss probability, etc.) of the connections on line and of incoming calls.

The rise of congestion may be caused by:

- statistical fluctuations of traffic flow;
- inefficient procedures in resource allocation and/or in user traffic control;
- faults in the network.

In this light, it is of paramount importance the definition of procedures apt to control traffic so as to avoid dangerous congestion problems. Traffic control procedures must have the following characteristics:

- simplicity, to avoid burdening the network with signalling traffic;
- effectiveness, so that control procedures must truly prevent congestion;
- fairness, so that the network will not privilege some services and/or users to the detriments of the others;
- efficiency, so that the use of the network resources is optimized.

Traffic control in ATM networks cannot be of the reactive type, that is, it activates just as congestion arises as in the case of classic window mechanism typical of current packet switched networks. Indeed, in this case, given the high speed with which traffic sources generate cells, reaction time would be too slow to prevent the network being invaded by more cells than it could handle. Therefore, traffic control in ATM networks has to be preventive by necessity, that is, to be successful it must prevent the rise of congestion.

The control procedure can be structured in three levels:

- admission control;
- routing control;
- bandwidth enforcement.

Admission control procedure is a rule that makes possible to decide whether or not to accept a connection. The acceptance decision is made at call set-up phase and is based on the current state of the network, on the traffic parameters of the source requesting the connection and on the user requirements in terms of maximum delay and cell loss probability. It is clear that whatever the procedure adopted, the user must declare its
own traffic characteristics, or at least the service it intends to use (that must be standardized).

Routing control occurs at call set-up phase. It consists in providing each network switching node of procedures that make possible the choice of the best route for the virtual connection to be established. The aim of these procedures is to distribute traffic uniformly over the network so as to prevent dangerous overloading of the physical links.

Bandwidth enforcement, also called *Usage Parameter Control* (UPC), is designed to prevent the user from generating more traffic than declared, that is, by using more bandwidth than allowed.

Traffic control is not a simple problem. The literature is full of proposals and methods that we do not deal with here in that it is not the purpose of this paper to do so. We just want to mention the most popular mechanism for UPC, the *Leaky Bucket*, [6].

The operation principle behind Leaky Bucket is very simple since it is based on discarding excess cells. The core of the mechanism is a counter that is increased by one as a cell arrives and decreased by one at (predefined) constant rate (called leaky rate). If the counter reaches a predefined threshold, and that means that the source is not respecting the declared bandwidth, all arriving cells are discarded until the counter does not go down below the threshold. From a modelling point of view, the leaky bucket behaves like a single server queue that has constant service rate equal to the leaky rate and finite buffer of capacity equal to the predefined threshold (fig. 13).

A modification of the Leaky Bucket, called Virtual Leaky Bucket, has been proposed in the literature, [7]. The only difference with Leaky Bucket is that cells are not discarded but simply marked as they are forwarded into the network. If the network experiences any congestion, marked cells will be discarded firstly. The idea behind this approach is that not all cells in excess of the requested average bandwidth will cause congestion. An account of all the recent research in the field of traffic control can be found in [8].

![Fig. 13 - Model of the Leaky Bucket mechanism.](image-url)
4.5 Models of ATM Traffic Sources

The introduction of ATM technique is posing new challenge to researchers working on traffic problems. The answer to this challenge is in the definition of new and original techniques to model traffic sources and to analyze queueing systems. Traffic problems arise above all because in ATM technique all various types of traffic generate a cells flow which has characteristics that strongly depend on the type of service. This poses firstly, the problem of modelling single traffic sources and the superposition of many of these sources.

Besides, to measure and evaluate the performance of ATM multiplexers, it is necessary to study single server queueing systems with deterministic service times (that is, cell transmission times, which are constant because of cells fixed length) and limited queue size (the multiplexer buffer). These service systems are fed by an incoming flow resulting from a superposition of several heterogeneous traffic sources.

ATM traffic sources can be classified according to the required transmission bandwidth. In general, we can identify two large groups:

- **Constant Bit Rate** (CBR) sources, characterized by a constant transmission bandwidth;

- **Variable Bit Rate** (VBR) sources, characterized by transmission bandwidth that varies during connection.

In turn, VBR sources can be classified in two ways (fig. 14):

![Diagram](https://via.placeholder.com/150)

**Fig. 14** - Bitrate vs. time for CBR and VBR sources.
- VBR sources of ON/OFF type, characterized by an alternation of activity and inactivity periods;

- VBR sources of continuous type, in which the bandwidth varies continuously during the connection (video codecs are typical examples).

CBR sources are easy to model since the cells flow generated by such sources is synchronous. Therefore the cell interarrival time is constant. It is to be noted however that this synchronism is altered by the network because of variable delays the cells experience in going through the network. Synchronism alteration produces a phenomenon called jitter, that can cause unacceptable degradation of isochronous services such as voice and video.

ON/OFF sources are more difficult to model. In this type of sources we may distinguish there different time scales (or layers) (fig. 15):

- Call layer;

- Burst layer;

- Cell layer

The call layer has the same duration as the connection. For example, in case the connection is a phone call, the duration is on the order of minutes. The burst layer has a much smaller time scale and is characterized by the periods in which the source is active (ON periods) and by the periods in which the source is idle (OFF state). For a phone connection these periods are on the order of hundreds of milliseconds.

![Diagram](image)

**Fig. 15 - Time scales for ON/OFF sources.**

The cell layer has a still smaller temporal scale and is characterized by the cells emission rate when the source in the ON state. Typically, this cells flow is synchronous with
period that varies according to the type of service. In the case of a phone connection the period is of the order of tens of milliseconds.

To model a single ON/OFF source is relatively simple. As a matter of fact, it can be shown that a single source may be modelled by a renewal type process. The difficult problem is to model the superposition of many (even homogeneous) ON/OFF sources. The key point is that the superposition of many renewal type sources is not renewal.

Two are the modelling approaches which has gained success. The first, proposed in [9], attempts to model the aggregate ATM stream by an approximating renewal process characterized by mean and variance of the distribution of interarrival times. In the second approach, pioneered in [10], the superposition process is approximated by a special nonrenewal process called Markov Modulated Poisson Process (MMPP). Both approaches lead to a relative simple ATM multiplexer analysis.

VBR sources of continuous type are more difficult to model. Autoregressive models of ARMA type may be used to characterize bit-rate, [11],[12]. Although rather accurate, unfortunately they lead to difficult analysis of the ATM multiplexers. Models based on continuous time Markov chains may also be employed, leading to an elegant analysis of ATM multiplexer based on fluid-flow techniques, [11], [13].

The interested reader may find a complete and comprehensive account of modelling and performance evaluation aspects of ATM in [14].

4.6 ATM Switching

The operating principle and the main blocks of an ATM switching node, as far as the basic switching capability is concerned, are summarized in figure 16.

![Diagram of an ATM switching node](attachment:atm_switching_node.png)

**Fig. 16** - Operating principle and main blocks of an ATM switching node.
The packet processors perform the label switching acting on the VPI/VCI field, i.e., assign the new VPI/VCI for the outgoing ATM multiplex, and at the same time provide the information necessary for cell routing within the switch fabric. In case of multipoint connections, an intermediate label is assigned and the switch fabric itself replicates the cells and assigns the final VPI/VCIs.

The information for cell routing is contained in a "routing tag" that the packet processor adds to the cell header. The length of the routing tag depends on the structure of the switch fabric. Figure 17 depicts the structure of a two-stages switch fabric and how the routing tag is used to self-route the cell.

Fig. 17 - Structure of a two-stages switch fabric and self-routing technique.

Fig. 18 - Buffering options at switching elements.
Each element of the switch, usually called ATM switching element, is a cell-synchronous block capable of realizing the input-output cell transfer in a single step. At the beginning of any step a cell is either stored in some input or shared buffer location waiting to be switched or is stored in an output (or shared-output) buffer waiting for transmission. ATM switching elements can be classified according to the position of buffers, [15]. There are input, output and shared buffer architectures, as well as modules with mixed buffering strategies (fig. 18). A comparison of input, output and shared buffer option based on an approximated analytical evaluation of cell loss probability has been reported in [16]. From the analysis in [16] it appears that, from a pure performance point of view, the shared buffer option is the best choice.

5. Conclusions

The evolution to high-speed network and services has now got stronger bases; the challenge to cope urgently with high-demanding market segments is creating, and will continue to create, the need for overlay networks and other specific solutions in the short term. While in the past, considerable effort has been devoted to identify an ideal future-proof solution (i.e. the B-ISDN), more recently the interest has shifted to the intermediate evolutionary steps driven by the market demand.

Among the basic technologies discussed in the paper, SDH is of pervasive nature, is in progress and needs an accurate overall network planning; MANs are systems of limited complexity and are therefore very useful for spot intervention due to the rapid crystallization of suitable products; the introduction of ATM technologies is seen firstly feasible in Digital Cross Connect products and access concentrators; eventually, TMN principles will be of pervasive extent to provide homogeneous management criteria.

References


SEARCHING FOR GRAVITATIONAL WAVES

Bernard F. Schutz
Department of Physics and Astronomy
University of Wales College of Cardiff, Cardiff, U.K.
b.schutz@astro.cf.ac.uk

Abstract

The next generation of gravitational wave detectors will include large-scale detectors based on principles of laser interferometry. They will take data at rates of many gigabytes per day for months or years at a time, and the data have to be searched for anticipated waveforms and for unanticipated events. Gravitational wave events of detectable strength will be rare, and the analysis methods need to give confidence in detections at the minimum possible signal-to-noise ratios. The software used needs to be flexible and adaptable.

After a brief review of gravitational waves, their likely astrophysical sources, and their methods of detection, we introduce the basic ideas of signal analysis. We then describe how present-day detectors, both bar and interferometer, deal with their data. We discuss the data challenges that the large interferometric detectors will pose, including the problem of performing an all-sky search for unknown neutron stars, whose sensitivity will be computer-limited for the foreseeable future. We conclude with a description of new data-analysis software systems being designed to cope with the needs of interferometers.

1 Gravitational Waves

1.1 Foundations of general relativity

We give here a short introduction to gravitational waves. Readers who want more information about general relativity are referred to Misner et al (1973), Price (1982), Schutz (1985), and Schutz & Will (1993).

1.1.1 The Equivalence Principle and Tidal Forces.

When Galileo conducted his famous experiment of dropping a large and a small ball from the Leaning Tower of Pisa and verifying that they reached the ground at the same time, he was demonstrating what we now call the equivalence principle. The modern way of formulating this principle is due to Einstein.

Let us focus on an experimenter who falls freely in a uniform gravitational field. Suppose Galileo jumped off the Tower and fell with the balls. They would simply have stayed alongside him, and he would not have been able to tell there was a gravitational field if he had only observed the motion of the balls. More generally, any observer who works in a freely falling reference frame in a uniform gravitational field will not be able to determine that there is a gravitational field at all.

The implication is that all the real, irremovable effects of gravity must have to do with the non-uniformity of the gravitational field. The differences between gravitational forces at nearby locations are called the tidal forces of gravity. They get their name from the tides raised on the Earth by the Moon and Sun. The Earth's oceans bulge out on both the near side and the far side with respect to the Moon. The near side of the Earth wants to fall towards the Moon slightly faster than the average, while the far side falls slightly slower. The size of the bulge is determined by the difference in the acceleration of gravity due to the Moon across the Earth. This gradient is called the tidal force of the Moon. If we could not see the Moon, we could still infer its existence from observations of the tides: tidal forces are the observer-independent gravitational forces.

Notice that tidal forces, being differences, grow with the size of the region over which they act. One does not see tides in a glass of water because the difference in the Moon's gravity from one side to another is small. But one does see tides on the Earth because it is large.

274
1.1.2 Gravitational Waves.

Gravitational waves (GWs) are present in any relativistic theory of gravity. When fields are weak and motions slow, the relativistic equations must reduce to Newton’s gravitational field equation,

\[ \nabla^2 \phi = 4\pi G \rho, \]

(1)

where \( \rho \) is the density of mass. This involves the differential operator \( \nabla^2 \). If fields are weak but motions may be rapid, then one expects the theory to be relativistically invariant. The operator \( \nabla^2 \) will be replaced by its relativistic generalisation

\[ \frac{1}{c^2} \frac{\partial^2}{\partial t^2} + \nabla^2. \]

This is the wave operator, with the characteristic wave speed \( c \). We should therefore expect wave effects in relativistic gravity, propagating at the speed of light. Different theories of gravity will differ in the details of the waves, but GWs will be present in all of them. For an elementary discussion, see Schutz (1984).

In general relativity, the waves bear a striking mathematical resemblance to electromagnetic waves. They are transverse, have two independent polarisations, and fall off in amplitude as \( 1/r \). Electromagnetic waves do not carry monopole radiation: for slow-motion sources they are dominated by dipole emission. Gravitational waves in Einstein’s theory do not carry either monopole or dipole radiation, but are dominated by quadrupole emission.

1.1.3 The Interaction of Gravitational Waves with Matter.

Waves can only be detected through their time-dependent tidal forces. If a wave passes a single particle, one can always choose comoving coordinates, so that the particle remains at a fixed coordinate position. By the equivalence principle, the particle will not feel anything locally from the wave: bowls of soup will not spill, pocket calculators will work normally. The particle will only be able to detect the wave if it looks at other things sufficiently far away for the tidal effects of gravity to be noticeable.

The simplest way to detect a GW is to monitor the proper distance between two nearby free particles, particles that have no external forces on them. The distance can be monitored for example by measuring the round-trip light-travel time between the particles. A GW can affect this distance. Gravitational waves are transverse, so if the wave moves in the \( z \)-direction, then only distances in the \( x-y \) plane will be affected.

The pattern of distance changes produced by a given wave is shown in Figure 1. The dots represent a ring of free particles. The circles and ellipses are not physical connections: they only illustrate the pattern of the placement of the particles. The first ring is the undisturbed position: a circular ring of particles. The second diagram shows the effect on proper distances of a GW. It has lengthened the \( z \)-axis of the circle and shortened the \( y \)-axis by the same fraction, preserving the area of the ring. When the wave reaches its opposite phase (final diagram), the effects on the axes are reversed. At the bottom of the diagram are shown the lines joining certain particles to the center of the ring. The top lines are for the particles at 12 o’clock and 3 o’clock (particles oriented in the “+” configuration relative to the ellipse’s principal axes). Their lines remain perpendicular but change their relative length. The bottom lines are for the particles at 1:30 and 4:30 (particles oriented in the “×” orientation relative to the ellipse’s principal axes). Their lines remain the same length but the angle between them changes.

Nothing is preferred about the \( z \)-axis in this diagram. The entire ellipse can be rotated about the \( z \)-axis to give possible patterns. A rotation of 90° effectively carries the pattern back into itself, which is characteristic of a tensor (spin-2) wave. The second independent polarisation is obtained by a rotation of 45°. Any other polarisation is a linear combination of these two fundamental ones.

Because tidal effects grow linearly with distance, the shape of the ellipse in Figure 1 is a property only of the wave, independent of the size of the ring. The relative change in the axes is, therefore, a measure of the amplitude of the wave itself. It is conventionally called \( h \), and can be defined from Figure 1 as

\[ h = 2 \frac{(\delta t)_{\text{max}}}{\ell}. \]

(2)
Figure 1: Illustration of the action of a GW on a ring of free particles transverse to the direction of propagation.

Polarization of a Gravitational Wave

Phase $\phi = 0$  Phase $\phi = \pi/2$  Phase $\phi = \pi$  Phase $\phi = 3\pi/2$

Response of a three-mass gravitational wave "detector" in two orientations

"+"  

"X"
where \( \ell \) is the length of the semimajor axis of the ellipse. (The factor of 2 is conventional.) The relative distortions shown in the figure are, of course, exaggerated in order to make them easy to see. Modern GW detectors are being built to detect amplitudes as small as \( 10^{-22} \). We shall consider the way this is done in Section (1.3) below.

This tiny amplitude means that the effect of matter on the gravitational wave will also be of this order: the wave will lose or scatter only a fraction of order \( 10^{-22} \) of its energy as it passes through a detector. By extension, this means that the waves that arrive at our detectors have lost little of their original form: they carry information from their sources uncorrupted by scattering or absorption. In this they are a unique carrier of information in astronomy: even the neutrinos detected from SN1987A were from a thermal distribution that had scattered many times before leaving the collapsed core. Traveling at the speed of light, gravitational waves follow locally straight lines through the Universe; they can be gravitationally lensed, but scattering and absorption are negligible.

1.1.4 Wave Emission: the Quadrupole Formula.

Sources of gravitational waves respect the conservation laws for energy and momentum that are built into Einstein's equations of general relativity. For weak GWs, these have a similar effect to that produced by the conservation of charge in electromagnetism, which ensures that there are no monopole electromagnetic waves: the lowest order of radiation is dipole. In general relativity, the law of conservation of energy similarly ensures there is no spherical gravitational radiation. Any oscillating spherical mass leaves its exterior gravitational field undisturbed in general relativity, just as it does in Newtonian gravity, so it does not radiate GWs. Conservation of momentum has the same effect on dipole radiation: there is no dipole gravitational radiation.

From a fundamental point of view, charge conservation (and its consequence, the absence of monopole radiation) in electromagnetism follows from the gauge-invariance of the theory. In general relativity, energy-momentum conservation (and the absence of monopole and dipole radiation) follow from the general coordinate invariance of the theory.

At quadrupole order, the formulas for the radiation emitted by a slow-motion system are remarkably similar to those for electric quadrupole radiation in electromagnetism. (For highly relativistic, fast-motion systems, the formulas below can only be used approximately.) We define the quadrupole moment tensor of the mass distribution:

\[
I_{jk}(t) = \int \rho(t, \mathbf{r}) x_j x_k d^3 x, \tag{3}
\]

where \( \rho \) is the mass density (rest mass dominates in a nonrelativistic system), and where we follow the usual convention of employing Latin indices for purely spatial values (1,2,3). This equation can only be used in a Cartesian spatial coordinate system. The trace-free or reduced quadrupole moment is derived from this by

\[
I_{jk} := I_{jk} - \frac{1}{3} \delta_{jk} I, \tag{4}
\]

where \( I \) is the trace of \( I_{jk} \).

The amplitude of the radiation at a distance \( r \) from the source is roughly

\[
h \sim \frac{4G \dddot{I}}{c^4 r}, \tag{5}
\]

where \( \dddot{I} \) stands for the second time derivative of a typical component of the quadrupole moment. If the motions that are responsible for the time-derivatives are driven by the internal gravitational fields of the source, which is the case for all the realistic sources of GWs that we will consider below, then there is a convenient upper bound on this amplitude, given by

\[
h \leq \frac{GM \phi_{\text{internal}}}{rc^2}, \tag{6}
\]

where \( \phi_{\text{internal}} \) is the typical size of the Newtonian gravitational potential within the source (Schutz 1984).
Although the local conservation laws of energy and momentum eliminate the monopole and dipole radiation, it does not follow that GWs remove no energy or angular momentum from the source. Unlike electromagnetism, which is a linear theory, general relativity is nonlinear, and GWs can act as sources for gravity. So when they leave the system, there is a gradual decrease in its energy. This is compensated by a well-defined energy flux carried by the waves. Energy from this flux can be transferred to other systems, such as GW detectors. The GW luminosity (Landau and Lifshitz 1962) is probably the most useful of the quadrupole formulas:

$$L_{SW} = \frac{G}{5c^5} \left( \sum_{j,k=1}^{3} \left| f_{jk} \right|^2 \right),$$

(7)

where the angle brackets denote an average over one period of the motion of the source. Notice that this depends on the square of a third time-derivative, just as the analogous formula for electric quadrupole radiation does. It is therefore very sensitive to the size of the velocities inside the source. As a source becomes more relativistic, the power radiated goes up very rapidly, as \((v/c)^6\).

1.2 Likely Sources of Detectable Waves

The first aspect of designing a detector is to estimate what amplitudes and frequencies one might expect from astronomical sources. These set targets for the experimental development. Because of the \((v/c)^6\) factor in the quadrupole formula, sources must be relativistic to produce observable radiation. Here is a brief review of the principal candidates and the astronomical information we might expect to get from them. It is by no means an exhaustive list, but it represents the most conservative predictions that have been made about possible sources. Many of the phenomena mentioned here are studied in more detail in Shapiro & Teukolsky (1983). Typical waveforms are illustrated in Figure 2.

1.2.1 Supernovae.

Supernovae are the explosions of giant stars that occur when their cores can no longer support themselves, as they run out of nuclear fuel. They are rare events, occurring once in perhaps 50 years in any galaxy. We would like to be able to detect them, therefore, in a volume of space containing perhaps 2000 galaxies, so that we have a reasonable chance of seeing one. This means that realistic detectors must reach as far as the Virgo cluster, a cluster of galaxies about 15 Mpc away, containing more than 1000 galaxies.\(^1\)

Supernovae will give off GWs if the collapse event is very non-spherical. The mass involved will be about a solar mass, and the size of the emission region is about 10 km. These give, from Eq. (6), and upper limit of a few times \(10^{-21}\) for waves from the Virgo cluster. Given that this is an upper bound, the usual target for detector development is \(h \sim 10^{-21}\). A detector with that sensitivity has some chance of seeing an occasional supernova explosion, unless they are all very symmetrical.

If such bursts are seen, they may allow us to identify the object (neutron star or black hole) formed by the collapse; they may provide crucial information about high-density nuclear physics; they may allow rapid notification of other astronomers that a supernova has occurred at a particular position; and they will enlighten our understanding of the late stages of stellar evolution.

1.2.2 Formation of Giant Black Holes.

Astronomers have good evidence that most galaxies contain giant black holes in their centres. If these form in a single collapse event, then they may emit strong bursts of GWs. The wavelength of the emitted radiation is proportional to the size (mass) of the black hole, and for holes in the centers of galaxies this can be in the milli-Hertz region or lower. Such events are not observable from the ground since ground vibrations and the near-zone Newtonian gravitational disturbances produced by atmospheric mass motions in this frequency range are too strong to screen out. But space-based detectors (below) are ideal for these frequencies. A suitable detector could see a giant black hole formation event anywhere in the Universe; it could verify or rule out that mode of formation of galactic black holes.

---

\(^1\) The astronomers' distance measure is the parsec, abbreviated pc. It equals \(3.086 \times 10^{16}\) m. It is typical of the distance between stars. Galaxies are separated by distances measured in megaparsecs (Mpc).
1.2.3 Coalescing Binaries.

Pulsar observations have shown us that neutron stars (which become pulsars if they rotate fast enough and have strong enough magnetic fields) often occur in binary pairs, close enough for the energy carried away from their orbits by the GWs generated by their motion to cause the orbit to shrink on time-scales short compared to the age of the Universe. (See Will 1993 for a discussion.) Such a system will be a strong source of GWs at relatively high frequency during the last few minutes before the stars coalesce. The frequency of the waves is twice the orbital frequency.

Such events are, of course, rare, occurring perhaps once every $10^5$ to $10^7$ years in any galaxy. We therefore would need to be able to detect them in a volume of space up to 200,000 times larger than that for supernovae, or as much as 60 times further away. Fortunately, this is made possible by the regularity and predictability of the signal from such a system.

In contrast to supernovae, which are likely to be messy, binaries of compact objects emit a steady, almost monochromatic "chirp" signal, whose frequency increases with time in a predictable way. If one has $N$ cycles of such a signal in one's data stream, then we shall see below that one can use pattern matching techniques (called matched filtering) to find it at an amplitude that is smaller by a factor of $\sqrt{N}$ than one could find if one had only a single cycle. Supernovae are basically single-cycle signals, so we can see binaries 60 times further away if we can detect 3600 cycles. The binary wavetrain will be followed by a much less predictable burst associated with the coalescence of the two stars.
The detector has to have a broad bandwidth at a few tens or hundreds of Hz to permit such filtering and following of the signal. In fact, the broadband interferometers described below are less noisy at the lower frequencies of binary signals, but on the other hand the intrinsic signal from a binary is a bit weaker than the strongest possible supernova (because $\phi_{\text{internal}}$ is smaller when the stars are still several radii apart), so the calculation must be done carefully. The result is that detectors will have about 40 times the range for binaries as they have for moderate supernova explosions. Detectors that can just barely see supernovae in Virgo (such as the first-stage interferometers described below) are unlikely to see coalescing binaries. But the second stage of development, with 10 times better amplitude sensitivity, ought to see hundreds per year.

The binary signal contains much information, including the individual masses of the component stars. By observing them, we will directly identify neutron stars and black holes and get much better statistics on their mass distributions. Since they occur at great distances, these events sample the cosmological mass distribution on very large scales. The signal contains enough information to allow the absolute distance to the binary to be estimated: they are true standard candles. They therefore contain cosmological information, including the possibility of measuring the Hubble constant, or the age of the Universe (Schutz 1986). The actual coalescence of the two objects (neutron stars and/or black holes in orbit) may be associated with the mysterious gamma-ray bursts seen by the Compton gamma-ray satellite (Meegan et al 1992). If so, they will be easier to model and understand. Finally, the detection of the radiation from the merger of two black holes will provide a strong test of general relativity itself: modeling this event on computers is an area of considerable current research activity.

1.2.4 Pulsars.

A neutron star that is axially symmetric will not emit GWs when it rotates, but we know that pulsars are not symmetrical: they have off-axis magnetic fields. If they have other asymmetries, perhaps mass deformations that help to pin the magnetic poles in one place, then they may give off detectable radiation. There is an upper limit on the strength of this radiation for any pulsar whose spin-down rate has been measured: GWs probably do not carry away more than the rotational energy loss of the pulsar.

The signal will be steady, so by the same argument we used for coalescing binaries, one's ability to find it increases with the square-root of the number of cycles, or in other words with the square-root of time. Again, a broadband detector is desirable, so that signals of any frequency can be detected. Given a few months' observation, second-stage interferometers are expected to be able to beat the upper limits on several known pulsars by factors of ten or more. It is also possible to conduct searches for unknown pulsars through their GW emission. This is a difficult problem because it involves considerable computing, and we will consider it more later.

1.2.5 Ordinary Binaries.

Binary star systems emit radiation at twice the frequency of their orbit. For all except the coalescing binaries considered before, this is a very low frequency, not attainable from the ground. But space-based detectors would be able to reach as low as $10^{-4}$ Hz, and could detect steady radiation from many binaries.

1.2.6 Cosmological Background.

Just as there is a cosmological background of electromagnetic radiation, one expects a background of GWs. The thermal background may be too weak to see, but many other potential sources have been discussed. Most rely on aspects of high-energy physics that give rise to inflation or to topological anomalies.

Theories that produce cosmic strings have been studied extensively. Strings are massive linear regions of trapped field, which can seed galaxy formation and can emit GWs. If strings do provide the seeds for galaxy formation, then they ought to produce a background of GWs that is detectable by interferometers.
1.2.7 Unexpected Sources.

One of the most exciting prospects is that there might be significant radiation from unpredicted sources. Whenever new windows on astronomy have been opened, such as radio or X-ray astronomy, completely unexpected objects have been found. This will undoubtedly be true for GWs as well, but of course one does not know what sensitivity will be required to reveal them. By going ten times deeper in amplitude (100 times in energy) than the strongest predictions for bursts, second-stage interferometers must stand a good chance of making such discoveries.

1.3 Detectors of Gravitational Radiation

Although detectors for GWs have been under development since J. Weber (1960) built the first one at the University of Maryland in the early 1960s, it is only recently that technology has permitted the design of a detector that meets theoretician's predictions about the likely strengths of expected waves. It is not unreasonable to expect that, by the year 2000, the first direct detection of a GW will have occurred.

Simple detection is not the main goal of the present detector development. GWs carry information about their sources that is obtainable in no other way. In order to extract maximum information from the waves, one needs to be able to infer their amplitude and direction. Since GW detectors are not directional, this can only be done with a world-wide network of detectors, which infer directions from the relative times of arrival of waves at different locations. Three detectors is the minimum for extraction of full information from the waves.

Approximately 6 or 7 detectors around the world are able to make observations today at a sensitivity near $10^{-18}$. In this section we review the techniques.

1.3.1 Bar-Type Detectors.

J. Weber designed and built the first GW detector in the 1960s. It was based on a cylinder of aluminium, suspended and isolated from ground vibrations. When a GW hits a bar broadside, the tidal forces represented in Figure 1 stretch it along its axis. By monitoring the excitation of the fundamental longitudinal mode of vibration of the bar, one can look for GWs.

Weber chose a bar that had a resonant frequency in the kHz region, to look for supernova events. He showed that, even for a burst event, it is desirable for the vibrational mode to have a large $Q$; such a mode is weakly coupled to the thermal bath of the other modes, making it easier to recognise the rapid change in its excitation caused by a GW.

The dominant background is thermal vibrations. Subsequent generations of bars have been cooled to 4.2 K to reduce this background. A further generation of bars now under construction may reach 10 mK (for example, see Astone et al 1991). Where Weber's original bar had a sensitivity limit of $h \sim 10^{-16}$ for a broadband burst around 1 kHz, cryogenic bars today reach about $10^{-18}$, and the new bars may go to $10^{-20}$. It is not clear yet whether they can go much beyond that. For the next five years, while interferometers are being constructed, the ultracryogenic bars will have the best chance of detecting GWs. If a supernova occurs in our Galaxy or in a nearby one, they may well succeed.

1.3.2 Laser-Interferometric Detectors.

An interferometer is designed to measure the relative difference in two optical paths. If one places an interferometer in the center of the ring in Figure 1, with the ends of two perpendicular arms on the ring, then the subsequent motion of the ends relative to the center can be detected by interferometry.

The relative motions are small, however, so the technical challenge is large. For the end mirrors to respond as free masses, they must be hung from supports and isolated from ground vibration. Because the tidal gravitational forces scale with distance, it is desirable to make the arms as long as possible. Present prototypes (see the various articles in Blair 1991) are in the 10–40 m range, but detectors now under construction (see below) will be as long as 4 km. It is this ability to take advantage of the tidal scaling that gives interferometers the edge over bars.
Even over 4 km, a disturbance of $10^{-21}$ translates into a mirror motion of $4 \times 10^{-16}$ cm, less than 0.01 fm. To sense average displacements of the surface of a macroscopic object to this accuracy with optical or infrared photons whose wavelengths are 12 orders of magnitude larger requires many photons. In turn, this requires excellent mirrors and high-power continuous-wave lasers. The physical principles of such detectors have been reviewed by Giazotto (1989).

The detectors will be built in stages. The first stage will aim at a sensitivity to broadband bursts of $10^{-21}$. The second stage will go for $10^{-22}$. Although the detectors are intrinsically broadband, they can also be tuned to narrow frequency ranges if desired. The light must travel along the arms in a good vacuum, better than $10^{-9}$ torr for the second stage. This is not hard in principle, but the volume to be evacuated is large: tubes 4 km in length and 1.5 m in diameter. The cost of this vacuum system is the dominant cost of the detectors.

Data will flow from these detectors at a great rate. Recent short observing runs reported below using prototypes at Glasgow University and the Max Planck Institute for Quantum Optics in Garching, Germany, produced data rates approaching 1 GB per hour.

There are proposals for a number of interferometric detectors around the world. A project called LIGO to build two in the United States (Abramovici 1992) is funded and should begin construction in 1993. If present (1993) schedules hold, LIGO could begin observing by 1998. A French-Italian detector called VIRGO, to be built near Pisa, is likely to get final approval in 1993 (Bradaschia et al 1990).

### 1.3.3 Space-Based Detectors.

As we noted above, the interesting frequency range around 1 mHz is only accessible to space-based detectors. Space-based searches for GWs have already been made using transponding data from interplanetary space probes. A passing GW would affect the time-delays of round-trip signaling to the probes; the signature of this effect is unique, and sensitive searches can be made at very low frequencies. No detections have been reported, but a three-spacecraft experiment was performed in 1993 using NASA's Mars Observer and Galileo probes and ESA's Ulysses solar observatory. When the data are analyzed, they may improve previous limits by a factor of 10 or more.

At the present time, ESA is considering proposals to place purpose-built interferometers into space to look for gravitational waves. Such a detector would open up the low-frequency part of the spectrum; it could detect many binaries and search for giant black holes and pulsars. At low frequency, its data rate would be low, and analysis would not be particularly difficult.

### 2 Analysis of Time-Series Data

The data coming from a gravitational wave detector is a single series of data values, sampled uniformly in time. The analysis of such data draws on a large body of signal-analysis theory, which we will sketch here.

The primary problem of the detection of gravitational radiation consists of identifying a gravitational waveform in a noisy signal. The noise is intrinsic to the detectors themselves: thermal noise in bar detectors, vibrations or photon-counting fluctuations in interferometers. If nothing is known ahead of time about the signal, then the only way to detect it is to look for events that are very strong, so strong that one would not expect noise to duplicate the event during the observation period. But it is possible to detect much weaker signals if we know what to expect. However, even if the form of the wave is known, it may still depend on a number of parameters whose values in any particular event are unknown. Typical parameters could be the masses of the stars in the binary system emitting the radiation. Therefore to detect the waveform we have to estimate its parameters at the same time.

At least in part because this problem is very similar to the problem of the detection of a target by radar and of the estimation of target parameters like range and velocity, the theory of detection and estimation is well understood. Standard textbooks on the theory of signal detection include the monographs by Helstrom (1968), Van Trees (1968), Whalen (1971), and Weinstein and Zubakov (1962). These texts are all oriented towards applications to radar. The first introduction to the subject with the detection of
gravitational waves in mind is a review article by Davis (1989). This article has the additional advantage of being written in the contemporary language of stochastic processes.

2.1 Signals in Noise

As a result of noise a datum from the detector is a value of a certain random variable. Since we take measurements at regular intervals of time, the data from a detector form a sample of a certain (discrete) stochastic process. Regardless of whether a signal is present or not, the data will still be stochastic, but the presence of a signal will affect the probability distribution of the stochastic process.

Let \( x_1 \) and \( x = (x_1, x_2, ..., x_n) \) be a sample of the random variable and of the stochastic process, respectively (i.e. a particular measurement). In general, there will be some probability density function (abbreviated pdf) \( p(x) \) that describes the distribution of data values: the probability that \( x_1 \) lies between a value \( x \) and a value \( x + dx \) is \( p(x)dx \). The probability for the whole set of values (process) is the joint pdf of the process. If the points are statistically independent, the joint probability is the product of the individual probabilities. But data values are not always independent.

Now, if there is no signal we call the joint pdf \( p_0(x) \); if the signal is present we call it \( p_1(x) \). These are different, because the distribution of random data values will be affected by the presence of the signal. To decide which pdf applies to a particular measurement \( x \), we have to devise a rule called the test which divides the range of values of \( x \) into two sets \( R \) and its complement \( R' \) in such a way that we decide the pdf is \( p_1(x) \) if \( x \in R \) and the pdf is \( p_0(x) \) if \( x \in R' \).

The detection probability \( P_D(R) \) is then given by the probability that a data set \( x \) that contains the signal will pass our test:

\[
P_D(R) = \int_R p_1(x) \, dx.
\]  

(8)

The false alarm probability \( P_F(R) \) is the probability that a data set that contains no signal passes our test:

\[
P_F(R) = \int_R p_0(x) \, dx.
\]  

(9)

The most appropriate way to test for the detection of gravitational waves seems to be the Neyman-Pearson approach. In this approach we seek a test that maximises the detection probability subject to a preassigned false alarm probability \( P_F(R) = \alpha \). Given that gravitational waves have not yet been detected, the primary consideration at least at first will be to be sure that one has detected one. By choosing the false-alarm probability \( \alpha \) sufficiently small, one ensures that the chances of falsely identifying a noise event as a gravitational wave are as small as one wants. In this way one misses many true gravitational wave events, but one has considerable confidence that those that are identified are real.

The solution for the “detection region” \( R \) in the Neyman-Pearson approach is given in terms of the likelihood ratio, defined by

\[
\Lambda(x) = \frac{p_1(x)}{p_0(x)}
\]  

(10)

If we let \( k \) be the likelihood threshold associated with the probability \( \alpha \),

\[
P_F[\Lambda(x) \geq k] = \alpha,
\]  

(11)

then the detection region of the space of all possible sample sets is

\[
R = \{x : \Lambda(x) \geq k\}.
\]  

(12)

Thus if for a particular observed set \( x \) we find that \( \Lambda(x) \geq k \), then we say that the signal is present; otherwise we say that the signal is absent.

In the following we assume that the sampled values of the signal \( h_i = h(t_i) \) are a deterministic function of time that we expect to find in our data. For example because calculations of gravitational wave sources have revealed it.
2.2 Characterisation of Noise

We assume now that the noise \( n(t) \) in the detector is a zero-mean Gaussian stochastic process. (If the noise statistics of a real detector are not Gaussian, as they may well not be, then certain conclusions below will have to be modified slightly. However, the Gaussian model should be close enough to give us a good idea of the accuracy with which parameters can be estimated.) The main way of characterising this noise is by its autocorrelation function \( K_{ij} \):

\[
K_{ij} = E[n_i n_j],
\]

where \( E[\cdot] \) denotes the expectation value. It can be shown for such a process that the logarithm of the likelihood ratio is given by

\[
\ln \Lambda[\mathbf{x}] = \sum_k x(t_k) q(t_k) - \frac{1}{2} \sum_k h(t_k) q(t_k),
\]

where \( q(t) \) is the solution of the equation

\[
h(t_j) = \sum_k K_{jk} q(t_k).
\]

We see that the likelihood ratio \( \Lambda[\mathbf{x}] \) depends on the particular set of data \( \mathbf{x} \) only through the sum

\[
G = \sum_k x(t_k) q(t_k).
\]

This sum \( G \) is called the detection statistic.

A very important special case is that of stationary noise, which is defined as a process which is independent of the origin of time, i.e. of when the experiment started. The autocorrelation then depends only on the difference between the times \( t \) and \( t' \); there exists a \( C_k \) such that

\[
K_{ij} = C_{i-j}.
\]

For the detection of short bursts of gravitational radiation, we may usually assume stationary noise. However, for pulsar and stochastic background measurements requiring long observation times, the assumption of stationary noise may be seriously wrong.

2.2.1 Fourier Transform Techniques.

The great advantage of stationary noise is that, if the noise is stationary and if the whole of the signal is included in the interval \( [0, T] \), then Eq. (15) can be solved by Fourier transform techniques.

Given any data series \( h_j \) of \( N \) points numbered from 0 to \( N-1 \), we define its discrete Fourier transform \( \tilde{h}_k \) by

\[
\tilde{h}_k = \sum_{j=0}^{N-1} h_j e^{-2\pi i j k / N},
\]

its inverse is

\[
h_j = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{h}_k e^{2\pi i j k / N}.
\]

The factor of \( 1/N \) must not be forgotten. There are various conventions for these definitions, according to whether \( 1/N \) is placed in the Fourier transform or in its inverse, or is shared between them.

If the index \( j \) counts sampled points in the time domain, the index \( k \) is the frequency index. Since a given real signal has Fourier components at both positive and negative frequencies (as in \( \cos 5t = \frac{1}{2} (\exp(5it) + \exp(-5it)) \), which has components at frequencies \( \pm 5 \)), the indices \( k = 0 \) to \( k = N/2 - 1 \) denote positive frequency components. With the frequency equal to \( \nu_k = k/(N\Delta t) \), for a sampling interval
\( \Delta t \). The negative frequencies are mapped by periodicity to larger values of \( k \): the negative component associated with a positive frequency \( k \) is at \( N - k \).

One of the most important operations one can perform with Fourier transforms is the convolution. Note that with a data set that is finite, the only way to define a correlation is cyclically: the data wrap around from back to front as they are shifted, or equivalently they extend to indices outside the range \( (0, \ldots, N - 1) \) periodically. Thus, the convolution of two discrete sets \( \{f_j, j = 0, \ldots, N - 1\} \) and \( \{g_j, j = 0, \ldots, N - 1\} \) is defined as

\[
\left(f \circ g\right)_j = \sum_{j'=0}^{N-1} f_{j'} g_{j-j'} = (g \circ f)_j.
\]

(19)

The key result here is the convolution theorem:

\[
\hat{f} \cdot \hat{g} = \hat{f} \cdot \hat{g}.
\]

(20)

2.2.2 Power Spectral Density of Noise.

We need a way of characterising the amount of noise, which is essentially its variance or standard deviation. In the normal language of Fourier transforms, the power spectrum is defined as the squared magnitude of the Fourier transform. In signal analysis, the power spectrum is described by the power spectral density of the noise, or p.s.d., conventionally denoted by \( S_k \). It is defined for stationary noise by

\[
E\hat{n}_k \hat{n}_k^* = S_k \delta_k + k.
\]

(21)

Notice that, for stationary noise, there is zero correlation between the noise at different frequencies, and at a given frequency the psd is the ordinary power spectrum. We say that the noise is white if the psd is flat, so that there is the same power at all frequencies. Many sources of noise are intrinsically white, but most measuring instruments have different responses at different frequencies, so that the noise in their outputs will generally be coloured.

It can be shown that \( S_k \) is the Fourier transform of the autocorrelation function of the noise:

\[
S_k = \sum_{t=0}^{N-1} K_{j,j+t} \exp(-2\pi it/k).
\]

(22)

where the right-hand-side is independent of \( j \) because of stationarity.

Putting all these results together, we find that by the convolution theorem, the solution \( q \) of Eq. (15) has the Fourier transform

\[
\hat{q}_k = \hat{n}_k/S_k.
\]

(23)

This filter is called the matched filter for the expected signal \( h \). Notice that it involves a weighting with \( S_k \): if the noise is high at some frequency, the Fourier component of the filter at that frequency will be reduced, so that it contributes relatively less to the convolution that produces the statistic \( G \) in Eq. (16) that searches for the signal. This shows how the detector’s noise distribution limits its sensitivity. nA bar detector is narrow-band: \( S_k \) is small only in a small range of values of \( k \). Interferometers are broadband, but they can be tuned to narrow ranges if desired.

2.3 Signal-to-Noise Ratio

Depending on whether signal is absent or present, the detection statistic \( G \) has one of the following probability density functions, provided that the noise is a zero-mean Gaussian process:

absent: \( p_0(G) = (2\pi d^2)^{-1/2} \exp(-G^2/2d^2) \)

(24)

present: \( p_1(G) = (2\pi d^2)^{-1/2} \exp(-(G - d^2)^2/2d^2) \)

(25)
Here the variance $d^2$ in each case is given by the filtered signal.

$$d^2 = \sum_{j=0}^{N-1} h(t_j) q(t_j).$$  \hspace{1cm} (26)

We call $d^2$ the (power) signal-to-noise ratio, or snr. The quantity $d$ itself is sometimes called the amplitude signal-to-noise ratio. If the noise is white then $d^2$ is the ratio of powers of the filtered signal and the filtered noise. When the noise is stationary, then the filter is given by Eq. (23), and the signal-to-noise ratio can be expressed as

$$d^2 = \sum_{k=0}^{N-1} \frac{|h_k|^2}{S_k}. \hspace{1cm} (27)$$

This is the best snr that can be produced by any linear filter.

The Neyman-Pearson test amounts to setting a suitably chosen threshold $G_0$ on the detection statistic. Then the false alarm and detection probabilities are given by, respectively,

$$Q_F = \text{erf}(G_0/d) \hspace{1cm} \text{and}$$

$$Q_D = 1 - \text{erf}(d - G_0/d). \hspace{1cm} (28)$$

where erf($x$) is the error function. Thus, the probabilities governing the detection of a known signal buried in Gaussian noise are completely determined by the amplitude signal-to-noise ratio $d$.

### 2.4 Dependence on Parameters

#### 2.4.1 The Maximum Likelihood Estimator.

In general we know the form of the signal $h(t)$ as a function of a number of parameters. In most problems, the amplitude of the signal and its time of arrival are almost always unknown parameters. For example, the waveform from a coalescing binary will also depend on two further parameters: the chirp mass (a certain combination of the masses of the members of the binary), and the phase of the wave at the time of arrival. In such a case in order to detect the signal we must also determine its parameters. For this one uses the maximum likelihood estimator.

Let $\theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ be the set of $m$ unknown parameters of the signal $h(t; \theta)$, and let (as in Eq. (8)) $x$ stand for the observed data function $x(t)$. As in the case of a completely known signal we can consider two probability density functions $p_1(x; \theta)$ and $p_0(x)$ depending on whether the signal is present or absent. From these we form the likelihood ratio

$$\Lambda(x; \theta) = \frac{p_1(x; \theta)}{p_0(x)}.$$

We define the maximum likelihood estimator (MLE) $\hat{\theta}$ of the set of parameters $\theta$ is the set that maximises the likelihood ratio $\Lambda(x; \theta)$. Thus the MLE can be found by solving the set of simultaneous equations

$$\frac{\partial}{\partial \theta_j} \Lambda(x; \theta_1, \ldots, \theta_m) = 0 \hspace{1cm} \text{for} \hspace{0.2cm} j = 1, \ldots, m. \hspace{1cm} (30)$$

For Gaussian noise, the MLE can be obtained by passing the data through a bank of linear filters for suitably spaced values of the parameters, each of the filters being determined by solving the linear equations analogous to Eq. (15):

$$h_j(\theta) = \sum_{t=0}^{N-1} K_{jt} q(t). \hspace{1cm} (31)$$

Once the maximum value of the likelihood ratio has been found for a set of filters, the spacings between parameter values may be further refined to get as close an approximation to the MLE as desired. Note that the errors in determining various parameters can be correlated, and this can make some parameters difficult to determine.
3 Analysis of Data for Bursts

Bursts can in principle be detected by looking at the output of one instrument, but one must always have coincident observations of the same events in different detectors, for several reasons: to increase one's confidence that the event is real, to improve the signal-to-noise ratio of the detection, and to gain extra information with which to reconstruct the wave. The simplest detection strategy splits into two parts: first find the events in single detectors, then correlate them between detectors.

I will illustrate the principles using data taken by the two European prototype interferometric detectors: the Glasgow University and Max Planck Institute for Quantum Optics (Garching) detectors. They took data in coincidence in 1989 for a period of 100 hours, and the data are being analysed in Cardiff.

3.1 Finding Broadband Bursts

A broad-band burst is an event whose energy is spread across the whole of the bandwidth of the detector, which I will take to be something like 100–5000 Hz (although considerable efforts are now being devoted to techniques for extending the bandwidth down to 40 Hz or less). To be detected it has to compete against all of the detector's noise, and the only way to identify it is to see it cross a pre-determined amplitude threshold in the time-series of data coming from the detector.

3.1.1 Threshold-Crossing Bursts.

The idea of setting thresholds is to exclude "false alarms" — apparent events that are generated by the detector noise. Thresholds are set to reduce the number of coincidences to those where there is some chance that a real event will be recognised. Our search is a form of matched filtering. Our model here is of an event that lasts for one sampling time, so that

\[ h_j = \delta_{jt} \]

for an event that arrives at the sampling time \( t_i \). The filter for this produces a correlation of the form

\[ G_j = \sum_{k=0}^{N-1} \frac{\hat{z}_k e^{-2\pi ik t_i/N}}{S_k}. \]  \hspace{1cm} (32)

This is just the inverse Fourier transform of \( x \) after weighting for the noise.

To look for coincident events, we perform a Fourier transform, determine the psd of the noise, invert the weighted noise, and then set a threshold of a signal-to-noise ratio of 4 in both data streams. We then look at the events that crossed the thresholds within 5 ms of each other: this is the light-travel time between the two detectors.

The resulting data and psd's are illustrated in the figures shown during the lectures. Bars and interferometers operate in a similar way here, although of course they have very different psd's in Eq. (32).

3.1.2 Finding coincidences.

The events that are coincident with one another are distributed randomly in signal-to-noise. The figures shown illustrate the pairs of amplitudes that are associated with such coincidences. There are no unusual coincidences, and no points that seem to be difficult to understand on the basis of noise. The data therefore provide only upper limits on the gravitational wave events during the observation period of about \( h \sim 1.5 \times 10^{-16} \).
3.2 Coalescing Binary Searches

Searches for coalescing binaries can be carried out by applying threshold criteria to the correlations produced by matched filtering using the more complex waveform expected from these sources. Only interferometers have the bandwidth to make effective searches for these sources. The false-alarm probabilities for detecting a coalescing binary have to be calculated with some care, however, because we must allow for the fact that we have in general to apply many independent filters, for different values of the stars’ masses, and this increases the chance of a false alarm.

The presence of parameters in this case complicates matters. Not only does one want to determine the time-of-arrival of the wave, but one also needs to determine the masses of the stars. It turns out that the errors in these measurements are not independent, and this makes the expected timing errors considerably larger.

We have filtered each individual data stream for coalescing binary events, but we have not yet looked for coincidences.

3.2.1 The Technology of Real-Time Filtering for Coalescing Binaries.

In this section I will discuss the technical feasibility of performing matched filtering on a data stream in “real time”, i.e. keeping up with the data as it comes out of a detector. Since coalescing binaries seem to make the most stringent demands for on-line processing, I will take them as fixing the requirements of the computing system. We need a data stream sampled at a rate of about 2 kHz in order to obtain the best S/N and timing information. If the detectors’ noise can be controlled down to 40 Hz, then the duration of the signal will be less than 30 seconds in almost all cases. This means that a filter need have about 60000 2-byte data points.

The quickest way of doing the correlations necessary for filtering is to use fast Fourier transforms (FFT’s) to transform the filter and signal, multiply the signal transform by the complex conjugate of the filter transform, and invert the product to find the correlation. The correlation can then be tested for places where it exceeds pre-set thresholds, and the resulting candidate events can be subjected to further analysis later. This further analysis might involve: finding the best value of the parameters; looking for unmodelled effects, such as tides or mass transfer; looking for the final burst of gravitational radiation as the two stars coalesce; and of course processing lists of these events for comparison with the outputs of other detectors. Since the number of significant events is likely to be relatively small, the most demanding aspect of this scenario is likely to be the initial correlation with a set of perhaps 1000 coalescing binary filters that span the space of parameters.

Because the correlation using Fourier transforms is circular, there is a danger that the filter will “wrap around”, so that its end will correlate with the earliest points in the sample. This is undesirable, but its effects can be controlled by taking the data set to be much larger than the length of the filter, and then “backspacing” the next set to avoid contaminated regions. For a filter length of $6 \times 10^4$ points, a data set of $N = 3 \times 10^9$ points will be reasonably efficient. Estimates show (Schutz 1991) that a computer working at around 300 Mflops could keep up with the data stream. To provide a margin for other operations, its effective speed should be at least 400 Mflops.

This is not a stringent requirement by today’s standards, and it will be even less difficult when detectors come on line in a few years. The operations are naturally parallelizable, because each processor in the computer can be assigned a different range of parameters to search for.

4 Looking for Pulsars and Other Fixed-Frequency Sources

4.1 Why the Data-Analysis Problem is Difficult

Rotating neutron stars are an important and exciting potential source of gravitational radiation. In some cases, such as nearby known pulsars, we will know ahead of time the frequency to look for and the position of the source. But most continuous sources may have unknown frequencies; indeed they
may only be discovered through their gravitational waves. I will first discuss the detection problem for sources of known frequency, and then consider searches for unknown sources. It is mainly relevant to interferometers, since bars typically do not have the sensitivity at pulsar frequencies to see them.

If we were on an observing platform that had a fixed velocity relative to the stars, and therefore to any pulsar we might be looking for, then finding the signal would be just a matter of taking the Fourier transform of the data and looking for a peak at the known frequency. This is a special case of matched filtering, since the Fourier integral is the same as the correlation in Eq. (16) with the filter equal to a sinusoid with the frequency of the incoming wave. Therefore, the signal-to-noise ratio \( d \) for an observation that lasts a time \( T_{\text{obs}} \) would increase as \( T_{\text{obs}}^{1/2} \), by Eq. (26).

However, the Earth rotates on its axis and moves about the Sun and Moon, and these motions would Doppler-spread the frequency and reduce its visibility against the noise. After a time

\[
T_{\text{max}} \approx 70\left(\frac{f}{1 \text{ kHz}}\right)^{-1/2} \text{min},
\]

a pulsar signal of frequency \( f \) gets smeared out by more than the observational resolution. The effect of the Earth’s orbit around the Sun gives a time roughly 2.8 times as long. The Earth’s motion about the Earth-Moon barycentre also has a significant effect. Since any serious observation is likely to last days or longer, the Doppler effects of all these motions must be removed, even in searches for very low-frequency signals (10 Hz).

### 4.2 Angular Resolution of a Pulsar Observation

The Doppler corrections one has to apply depend on the location of the source in the sky. Since the spin axis of the Earth is not parallel to orbital angular momentum vectors of its motion about the Sun or Moon, there is no symmetry in the Doppler problem, and every location on the sky needs its own correction. The orbital motion of the Earth sets the most stringent limit for observation times \( T_{\text{obs}} \) greater than a couple of days:

\[
\Delta \theta_{\text{orbit}} = 1 \times 10^{-6} \left(\frac{f}{1 \text{ kHz}}\right)^{-1} \left(\frac{T_{\text{obs}}}{10^7 \text{ s}}\right)^{-2} \text{ rad}, \quad \text{for } T_{\text{obs}} < 1 \times 10^7 \text{ s}. \tag{34}
\]

This reaches a minimum of about 0.2 arcsec for a millisecond pulsar observed for 4 months.

Observations can therefore determine the position of a pulsar very accurately, but of course there is a compensating problem. Uncertainties in the position of the pulsar being searched for, orbital motion of the pulsar in a binary system, proper motion of the pulsar (e.g., a transverse velocity of 150 km/s at 100 pc), or unpredicted changes in the period [anything larger than an accumulated fractional change \( \Delta f/f \) of \( 10^{-10} (f/1 \text{ kHz})^{-1} \)] will all require special techniques to compensate for the way they spread the frequency out over more than the frequency resolution of the observation.

### 4.3 The Need for Performing Long Fourier Transforms

If one imagines that the observation lasts \( 10^7 \) s with a sampling rate of 1 kHz, then one must perform an FFT with roughly \( 10^{10} \) data points. This requires roughly \( 3N \log_2 N \) operations for \( N = 2^{24} = 1.7 \times 10^{10} \). This evaluates to \( 1.7 \times 10^{12} \) operations per FFT. Given the 400 Mflops computer we required earlier for filtering for coalescing binaries, this would take just over 1 hour. This is not unreasonable: over 2000 FFTs could be computed in the time it took to do the observation. This would suffice to search for a number of known pulsars, but not to search all possible locations on the sky for unknown ones.

There is a difficulty with this: the memory requirement. FFT algorithms require access to the whole data set at once. To achieve these processing speeds, the whole data set would have to be held in fast memory, all 20 GBytes of it. Today, it is possible to get up to half a gigabyte into the memory of ordinary workstations, so it is possible that this much memory will not pose a problem in three or four years' time.
4.4 Detecting Known Pulsars

Let us assume that we know the location and frequency of a pulsar, and we wish to detect its radiation. We need to make a correction for the Doppler effects from the known position, or from several contiguous positions if the position is not known accurately enough ahead of time. If many searches are done all at once, this can be done by filtering. For only a few searches, a more efficient method is a numerical version of the standard radio technique called heterodyning. See Niebauer et al (1993) for details.

4.5 Searching for Unknown Pulsars

One of the most interesting and important observations that interferometers could make is to discover old nearby pulsars or other continuous wave sources. There may be thousands of spinning neutron stars — old dead pulsars — for each currently active one. The nearest may be only tens of parsecs away. But we would have to conduct an all-sky, all-frequency search to find them. We shall see in this section that the sensitivity we can achieve in such a search is limited by computer technology.

The central problem is the number of independent points on the sky that have to be searched. It follows from Eq. (34), the angular resolution increases as the square of the observing time, so the number of patches on the sky increases as the fourth power. For observations longer than 20 hours, Eq. (34) implies

\[ N_{\text{patches}} = \frac{4\pi}{(\Delta \theta)^2} = 1.3 \times 10^{13} \left( \frac{f}{1 \, \text{kHz}} \right)^2 \left( \frac{T_{\text{obs}}}{10^7 \, \text{s}} \right)^4. \] (35)

The various methods of searching these patches have been studied by Schutz (1991).

4.5.1 Depth of a Search as a Function of Computing Power.

The data set will have a length

\[ N_{\text{pts}} = 2 \times 10^{10} \left( \frac{f}{1 \, \text{kHz}} \right) \left( \frac{T_{\text{obs}}}{10^7 \, \text{s}} \right) \text{ points}, \]

provided we interpret \( f \) as the highest observable frequency, so we sample at a rate \( 2f \). No method of searching will scale faster than \( N_{\text{pts}} \). If we assume that our method requires \( 10 \) real floating point operations per data point per place on the sky, then we need to perform

\[ N_{\text{flop ops}} = 2.5 \times 10^{24} \left( \frac{f}{1 \, \text{kHz}} \right)^3 \left( \frac{T_{\text{obs}}}{10^7 \, \text{s}} \right)^5 \]

floating point operations to search the whole sky.

In order to do repeatable searches, it must be possible to analyze the data in roughly the same time it takes to take it. If the computer speed is called \( S \), measured in floating point operations performed per second, then the time it takes to perform \( N_{\text{flop ops}} \) operations is \( N_{\text{flop ops}} / S \) s. Ignoring overheads due to other factors, we therefore find that the time to analyze the data is

\[ T_{\text{analyze}} = 2.5 \times 10^{13} \left( \frac{f}{1 \, \text{kHz}} \right)^3 \left( \frac{T_{\text{obs}}}{10^7 \, \text{s}} \right)^5 \left( \frac{S}{100 \, \text{Gflops}} \right)^{-1} \text{ s.} \]

By equating \( T_{\text{analyze}} \) and \( T_{\text{obs}} \), we obtain the maximum observation time allowed by a computer of a given speed:

\[ T_{\text{max}} = 2.5 \times 10^5 \left( \frac{f}{1 \, \text{kHz}} \right)^{-3/4} \left( \frac{S}{100 \, \text{Gflops}} \right)^{1/4} \text{ s.} \] (36)

This is about 17 hours for a 400 Mflops computer analyzing data for millisecond pulsars (up to 1 kHz). If we lower our sights and try to search for pulsars under 100 Hz (still very interesting), we can run for about 4 days.
Even on these optimistic estimates, the permissible observation time will grow only as the fourth root of the computer speed. Even worse, since the sensitivity one can reach in $h$ scales as the square root of the observation time, the limits on $h$ will scale as the eighth root of the computer speed! Changing from a desktop computer capable of 0.1 Mflops to a supercomputer capable of 10 Gflops improves one’s limits on $h$ by only a factor of 4.

This is the central problem of the all-sky search for pulsars: it is quite possible to run detectors for several months gathering data, and this will probably be done to search for known pulsars, but computing power limits any all-sky, all-frequency search for unknown pulsars to periods of the order of days.

5 Using Cross-Correlation to Discover a Stochastic Background

Another very important observation that interferometers will make is to find or set limits upon a background of radiation. This is much easier to do than finding discrete sources of continuous radiation, because there is no direction-finding or frequency-searching to do. This problem has been discussed in detail by Michelson (1987), who particularly concentrated on bar-type detector experiments. The basic idea is that the background is just a form of noise, but the noise is correlated between two detectors. By cross-correlating their outputs, their intrinsic noise is suppressed but the common noise — the background — stands out. A correlation search for the 100-hour data is underway in Cardiff at the present time.

The most sensitive search for a background would be with two detectors on the same site, with the same polarisation. Current plans for some installations envision more than one interferometer in one vacuum system, which would permit a correlation search. One would have to take care that common external sources of noise are excluded, especially seismic and other ground disturbances, but if this can be done then the two detectors should respond identically to any random waves coming in, and should therefore have the maximum possible correlation for these waves. If separated detectors are used, the essential physical point is that two separated detectors will still respond to waves in the same way if the waves have a wavelength $\lambda$ much longer than the separation between the detectors. Conversely, if the separation between detectors is greater than $\lambda/2\pi$, there is a significant loss of correlation. It is important as well to try to orient the detectors as nearly as possible in the same polarisation state. In order to perform a search at 100 Hz, the maximum separation one would like to have is 500 km. This may be achievable within Europe, but it seems most unlikely that detectors in the USA will be built this close together. The data analysis is exactly the same as for two detectors on the same site.

Detectors need not be of the same type. Under some circumstances, it might be desirable to correlate the output of a bar detector and an interferometer whose output had been filtered to the bar’s bandwidth. This has recently been studied by Astone, Lobo, and Schutz (to be published).

6 Data Storage and Exchange

Bars and space-based interferometers present no problems for data volumes, but the amount of data generated by a four-detector network will be huge, and it is important that the data should be archived in a form that is relatively unprocessed, and kept for as long a time as possible, certainly for several years. It may be that new and unexpected sources of gravitational waves will be found, which will make it desirable to go over old data and re-filter it. It may also be that new classes of events will be discovered by their electromagnetic radiation, possibly with some considerable delay after the event would have produced gravitational waves, and a retrospective search would be desirable. In any case, we have already seen that it will be important to exchange essentially raw data between sites for cross-correlation searches for unknown events. Once exchanged, it is presumably already in a form in which it can be stored.

6.1 Storage Requirements

A network of interferometers could generate 5000 optical discs per year. Data compression techniques and especially the discarding of most of the housekeeping data at times when it merely indicated that
the detector was working satisfactorily could reduce this substantially, perhaps by as much as a factor of four. The cost of the storage media is not necessarily trivial. While tape media are inexpensive, optical discs of large capacity could cost $250k at present prices.

6.2 Exchanges of Data Among Sites

We have already seen how important it will be to cross-correlate the raw data streams. At a data rate of some 100 kbytes per second, or even at 30 kbytes per second if the data volume is reduced, one would have difficulty using standard international data networks. But these networks are being constantly upgraded, and so in five years the situation may be considerably different. Alternatively, a cheaper solution would be to exchange optical discs or videotapes physically, accepting the inevitable delay. Lists of filtered events can be exchanged on electronic data networks, giving less urgency about exchanging the full data sets.

7 Grid

7.1 Reconfigurable Data Analysis

During the development of large-scale interferometers, it is clear that there will be a period of "trouble-shooting", when the detectors are working, but badly. This is the period when unexpected sources of noise must be tracked down and eliminated, when control loops don't work as expected and need to be changed, and when other systems are not performing well and their problems have to be diagnosed. There will be a multiplicity of data streams form various points around the detector, and crucial information may lie in correlated behaviour in different streams.

To cope with this period, we have been developing in Cardiff a software system that provides a framework within which data can be analysed quickly and in a flexible way: new data can be brought in quickly, new operations performed, and old ones modified. It is designed to run in a windowing environment on a workstation connected to a data network. We call the system Grid. It has been developed primarily by J. Shuttleworth and T. Barnett, with the cooperation of the groups at Glasgow, Garching, and Caltech.

The figure shows a screen dump of a workstation on which Grid is running. The key window is the graphical editor, where units that are pre-compiled subroutines that perform operations can be linked by "wires" that carry the data from one place to another.

Other software is available that can do things like this; a superficially similar one is called Koros. But Grid differs from other systems in its internals. Its units are subroutines that are linked and called efficiently when the system is started, so there are few overheads in running data through them. The internal code is written in C++, which allows us to use sophisticated data types. Eventually these types will contain information about the physical dimensions of the data and will allow tracking of errors introduced by mathematical operations.

7.2 How Grid Works

Grid presents the user with an easily-grasped graphical metaphor of a data-flow diagram. This diagram represents the digital signal processing algorithm to be executed. The user places processing units onto a work area and connects these processing units together with "cables". The data to be processed flows through the cables, being manipulated by the units it encounters. Thus Grid can be thought of as a virtual "lab-bench". The Grid user-interface employs the now prevalent pointer/icon paradigm as a convenient way of building and manipulating signal processing networks. Grid has been designed to be as portable as possible (we have minimised reference to specific operating system facilities) and is written using state-of-the-art object-oriented techniques (Booch 1991) in an emerging industrial-standard language — C++ (Ellis & Stroustrup 1990). This graphical approach to signal processing combines several advantages over conventional, sequential coding.
1. Coding errors are avoided. As the main units are pre-compiled (and optimised) less time is spent finding and eliminating the causes of erroneous results.

2. The network can be reconfigured quickly and easily. Parameters can be altered and the resultant changes in output can be seen while the network is running.

3. Networks can be saved and restored at any time. All parameters of the network are saved. To extend the lab-bench metaphor, this is like being able to both automatically restore the connections between equipment and restore the settings on each item of equipment.

4. Processing units can be executed on different computers. Grid takes advantage of the networking facilities provided on most workstations to allow the user to request that specific units or collections of units may be executed on any other computer connected to the network. As our current networks provide world-wide connectivity, this means that any suitably authorised machine in the world may conceivably become part of a signal processing network. This remote execution potentially offers an increase in processing speed and permits remote data acquisition.

Grid will soon be tested by the gravitational wave community. We hope to be able to release it more generally within about a year.

Acknowledgements

It is a pleasure to acknowledge the contributions of many colleagues to my understanding of the issues discussed here. They include Tim Barnett, Katherine Compton, Mark Davis, Sanjeev Dhurandhar, Chris Dickson, Jim Hough, Gareth Jones, Andrzej Krolak, Alberto Lobo, David Nicholson, Tim Niebauer, Justin Shuttleworth, Kip Thorne, Harry Ward, and John Watkins.

References


D. ALLEN, CERN, Geneva, Switzerland
A. ALMEIDA, LIP, Lisbon, Portugal
M. ANTONIUUK, The University of Mining and Metallurgy, Cracow, Poland
A.S. AYAN, Middle East Technical University, Ankara, Turkey
F. BARALE, European Synchrotron Radiation Facility, Grenoble, France
E. BERGAN, University of Oslo, Norway
H.-M. BOSCH, CERN, Geneva, Switzerland
B.T. BOUWENS, NIKHEF-H, Amsterdam, Netherlands
A. BRANCO, LAFEX/CPBF, Rio de Janeiro, Brazil
(presently at CERN, PPE Division)
C.E.E. BRUSCHINI, Università di Genova, Italy
(presently at CERN, PPE Division)
A. CALDAS FERREIRA, LIP, Lisbon, Portugal
D. CARRON, European Synchrotron Radiation Facility, Grenoble, France
E. D’AMICO, CERN, Geneva, Switzerland
J.C. DE VRIES, CERN, Geneva, Switzerland
J. DOMINGUEZ-SOL, CERN, Geneva, Switzerland
P. DUDA, Warsaw University of Technology, Poland
B.J. FEYEN, RUCA, Universiteit Antwerpen, Belgium
I. FLADMARK, European Synchrotron Radiation Facility, Grenoble, France
G. FOLGER, CERN, Geneva, Switzerland
D. GALLI, INFN, Sezione di Bologna, Italy
M. GENTILE, CERN, Geneva, Switzerland
D. GOLDNER, Dortmund University, Germany
A. GOMES, LIP, Lisbon, Portugal
F. HERNANDEZ, Centre de Calcul de l’IN2P3, Villeurbanne, France
S.A. JAGIELSKI, The Academy of Mining and Metallurgy, Cracow, Poland
A. KHASANOV, LCTA, JINR, Dubna, Russia
J.T. KLEM, CERN, Geneva, Switzerland
B. KOTLINKSI, Paul Scherrer institute, Villigen, Switzerland
M. KRAUTGAERTNER, Weizmann Institute of Science, Rehovoth, Israel
A. KYRIAKIS, N.R.C. "Demokritos", Aghia Paraskevi, Greece
R. LINDBERG, CERN, Geneva, Switzerland
R.S. MAYER, Gesellschaft für Schwerionenforschung, Darmstadt, Germany
S. NICKEL, DESY, Hamburg, Germany
W. NIESSEN, CERN, Geneva, Switzerland
T.E. PETTERSEN, Bergen University, Norway
C. RETHFELDT, DESY, IIH, Zeuthen, Germany
B.R. SCHULZE, LAFEX/CBPF, Rio de Janeiro, Brazil
(at present also with INFN/Rome II and CERN)
R.S. SCHULZE, LAFEX/CBPF, Rio de Janeiro, Brazil
(presently at CERN, PPE Division)
A. SEGRETO, Università degli Studi, Palermo, Italy
R. SINKUS, DESY, Hamburg, Germany
R. SPIWOKS, Dortmund University, Germany
P. STENSTROM, Stockholm University, Sweden
Z. SZKUTNIK, DESY, Hamburg, Germany
J. SZLACHCIAK, Soltan Institute for Nuclear Studies, Otwock-Swierk, Poland
J. VUOSOSKI, CERN, Geneva, Switzerland
E. WILHELMSEN, Oslo University, Norway
H. WITSCH, European Synchrotron Radiation Facility, Grenoble, France
M.P.J. ZINSER, Gesellschaft für Schwerionenforschung, Darmstadt, Germany
E. ZUDE, Gesellschaft für Schwerionenforschung, Darmstadt, Germany
1993 Students

D. ALLEN, CERN, Geneva, Switzerland

A. ALMEIDA, LIP, Lisbon, Portugal

M. ANTONIUK, The University of Mining and Metallurgy, Cracow, Poland

A.S. AYAN, Middle East Technical University, Ankara, Turkey

F. BARALE, European Synchrotron Radiation Facility, Grenoble, France

E. BERGAN, University of Oslo, Norway

H.-M. BOSCH, CERN, Geneva, Switzerland

B.T. BOUWENS, NIKHEF-H, Amsterdam, Netherlands

A. FRANCO, LAFEX/CBPF, Rio de Janeiro, Brazil

(presently at CERN, PPE Division)

C.E.E. BRUSCHINI, Università di Genova, Italy

(presently at CERN, PPE Division)

A. CALDAS FERREIRA, LIP, Lisbon, Portugal

D. CARRON, European Synchrotron Radiation Facility, Grenoble, France

E. D'AMICO, CERN, Geneva, Switzerland

J.C. DE VRIES, CERN, Geneva, Switzerland

J. DOMINGUEZ-SOL, CERN, Geneva, Switzerland

P. DUDA, Warsaw University of Technology, Poland

B.J. FEYEN, RUCA, Universiteit Antwerpen, Belgium

I. FLADMARX, European Synchrotron Radiation Facility, Grenoble, France

G. FOLGER, CERN, Geneva, Switzerland

D. GALLI, INFN, Sezione di Bologna, Italy

M. GENTILE, CERN, Geneva, Switzerland

D. GOLDNER, Dortmund University, Germany

A. GOMES, LIP, Lisbon, Portugal

F. HERNANDEZ, Centre de Calcul de l'IN2P3, Villeurbanne, France

S.A. JAGIELSKI, The Academy of Mining and Metallurgy, Cracow, Poland

A. KHASANOV, LCTA, JINR, Dubna, Russia

J.T. KLEM, CERN, Geneva, Switzerland

B. KOTLINKSI, Paul Scherrer Institute, Villigen, Switzerland

M. KRAUTGAERTNER, Weizmann Institute of Science, Rehovoth, Israel

A. KYRIAKIS, N.R.C. "Demokritos", Aghia Paraskevi, Greece

R. LINDNER, CERN, Geneva, Switzerland

R.S. MAYER, Gesellschaft für Schwerionenforschung, Darmstadt, Germany

S. NICKEL, DESY, Hamburg, Germany

W. NIESEN, CERN, Geneva, Switzerland

T.E. PETTERSEN, Bergen University, Norway

295
C. RETHFELDT, DESY, IfH, Zeuthen, Germany
B.R. SCHULZE, LAFEX/CBPF, Rio de Janeiro, Brazil
   (at present also with INFN/Rome II and CERN)
R.S. SCHULZE, LAFEX/CBPF, Rio de Janeiro, Brazil
   (presently at CERN, PPE Division)
A. SEGRETO, Università degli Studi, Palermo, Italy
R. SINKUS, DESY, Hamburg, Germany
R. SPIWOKS, Dortmund University, Germany
P. STENSTROM, Stockholm University, Sweden
Z. SZKUTNIK, DESY, Hamburg, Germany
J. SZLACHCIAK, Soltan Institute for Nuclear Studies, Otwock-Swierk, Poland
J. VUOSKOSKI, CERN, Geneva, Switzerland
E. WILHELMSEN, Oslo University, Norway
H. WITSCH, European Synchrotron Radiation Facility, Grenoble, France
M.P.J. ZINSER, Gesellschaft für Schwerionenforschung, Darmstadt, Germany
E. ZUDE, Gesellschaft für Schwerionenforschung, Darmstadt, Germany