LARGE COMPUTER SYSTEMS AND NEW ARCHITECTURES

T. Bloch

CERN, Geneva, Switzerland.

1. Super-computers

1.1 Use and classification of super-computers

Up till quite recently the highest speed commercially available computers such as the IBM 7090, the CDC 6600 and the IBM/360-91 in the 1960's, and the CDC 7600 and IBM/360-195 in the early 1970's had a few attractive characteristics which were taken for granted:

- i) They represented the most cost effective way of providing computing (if you could afford them); Grosch's "law" that the price only grows with the square root of the speed (within the same generation of computers) was conveniently observed by the computer manufacturers' pricing manuals.
- ii) The software systems provided were general purpose: COBOL, FORTRAN, multiprogramming systems, sophisticated data set handling, terminal support, etc.
- iii) The speed gains were transparent to the end-users who simply saw their programs running 50 times faster on the CDC 7600 than they did a decade earlier on the IBM 7090.

For economic and technological reasons these qualities of very high speed computing are all practically non-existent today. The fastest general-purpose computers available from CDC and IBM are the CDC Cyber 176 (about the same speed as the CDC 7600 first delivered in Autumn 1969) and the IBM 3033 which is somewhat slower than the IBM 360-195 (first delivered in 1970). The Amdahl 470/V7 and the Fujitsu M-200 are reputed to be somewhat faster than the IBM 3033, but even so, there has not been any significant increase in the speed of the largest available general-purpose computers for about eight years. Most computer users have found this quite tolerable and have covered their needs for additional capacity by installing multiple mainframe systems. With today's software and hardware facilities allowing sharing of data sets, and remote access from terminals and batch stations, it is not too difficult to provide a good service on such systems and the effect of one (out of three) mainframe going out for repair or software tests is (mostly!) easy to bear

compared with having your one and only super-computer going down for the rest of the prime shift.

The interest in other high speed computing than this sort of multiple mainframe installation comes mainly from two types of computer users:

- a) those with a total need for computing power five times and beyond a CDC 7600 or with very large problems which cannot be split and have to be solved within a fixed time frame (e.g., weather forecasting);
- b) those with a relatively modest general-purpose installation, but with a need for real high speed computing at a low acquisition price.

1.2 The super-computers of recent years

For the first category of users mentioned above there have been four different computers marketed over the last five to seven years:

- the CDC String Array processor (STAR-100);
- the Texas Instruments Advanced Scientific Computer (ASC);
- the Cray Research Inc. computer (CRAY-1);
- the Burroughs Scientific Processor (BSP).

These computers are all capable of producing floating point results ten times faster than a CDC 7600, given the absolutely right problem, and have been costing \$10-15 million for a system complete with front-end computers and indispensable peripherals. Another common "feature" is that less than 10 of each have been built till now. Here, however, the similarity stops and although some of the technical differences are discussed in detail later, it seems useful to make a few remarks on each one here to situate them better.

1.2.1 CDC STAR-100¹⁾ (Figs. 1-13). CDC started to design the STAR-100 in 1964 and delivered the first machine in 1974. Altogether three systems have been installed outside CDC so far [Lawrence Livermore Laboratory (2), NASA Langley Research Centre]. The STAR-100 has a multiprogramming operating system, virtual memory, and handles peripherals and terminals using rather independent "station computers" linked to the main computer via high speed channel-to-channel links. The STAR-100 runs on a 40 ns CPU cycle and has up to 1 Mwords (64 bits

plus parity) of core memory (1.28 μ s cycle time). The scalar part of the CPU works mainly on a set of 256 general-purpose registers using a three-address order code. The vector part of the CPU typically streams one or two operand vectors from memory through a floating point "pipe" generating a result vector in memory. The elements of the vectors all have to be consecutive in the memory owing to the fact that the relatively long access time of the core memory is compensated for by having it organized in superwords of 512 bits each. Once such a "stream" is started up, however, a 64-bit result may be generated every 40 ns. Since there are two floating point pipes and they can be used in overlapped mode, and since each pipe can generate 32-bit results at the rate of two per cycle, we get a top speed of one floating point result per 10 ns -- or 100 Millions of FLOating Point operations per Second (MFLOPS).

1.2.2 TI ASC ²⁾. Texas Instruments started the ASC project in 1968 and delivered the first machine in 1973. Three systems have been installed outside of TI (Geophysical Fluid Dynamics Laboratory in Princeton, Systems Development Corporation, and Naval Research Laboratory) and TI terminated the project in 1976. Several machines are in use within the TI service bureau specialized in seismic data processing. The ASC has a complete multiprogramming operating system somewhat similar to the IBM OS/360 MVT and it uses a mixture of direct channels and eight integrated peripheral processors (somewhat akin to those of the CDC 6600) to control the input/output equipment.

The ASC runs on an 80 ns CPU cycle and has up to 1 Mwords (32 bits plus error correction bits) of semiconductor memory (256 bits/chip) with a 160 ns cycle time. The scalar part of the CPU works with 16 base address registers, 16 general arithmetic registers, and eight index registers. The vector part of the CPU typically streams one or two operand vectors from memory through a general-purpose "pipe" generating a result vector in memory. ASCs may have one pipe, two pipes or four pipes with either one instruction processor or, in the case of a four-pipe machine, two instruction processors each tied to two pipes. The elements of the vectors have to be spaced regularly in memory but one is not limited to an increment of 1. The actual vector instruction is a rather complex affair that can handle indexing corresponding to three nested FORTRAN DO-loops, provided the inner loop has an increment of 1 (or -1!). Each pipe generates a 32-bit result every 80 ns cycle, thus achieving a maximum of 50 MFLOPS on a four-pipe machine.

1.2.3 CRI CRAY-1 ³⁾ (Figs. 14-27). Cray Research Incorporated was established in 1972 with the initial goal of designing a large scientific computer, the CRAY-1. The first CRAY-1 was delivered in March 1976 to Los Alamos Scientific Laboratory. At present about six CRAY-1s are installed and several more are on order. Among the more recently installed systems the following sites can be mentioned: National Centre for Atmospheric Research (Boulder, Col., USA), United Computing Services (Kansas City, Miss., USA), Magnetic Fusion Energy Project (Livermore, Calif., USA), European Centre for Medium Range Weather Forecasting (Reading, UK).

The CRAY-1 runs under the control of a simple multiprogramming operating system and is limited by software to using only high speed disks as its direct peripheral equipment. It requires a general-purpose "front-end" computer with a channel-to-channel link to pass to it jobs, permanent files from disks and magnetic tape files, and to absorb the various output files. All these external files are copied to and from the CRAY-1 disks before and after job execution; this copying operation is normally referred to as "staging".

The CPU cycle time of the CRAY-1 is 12.5 ns and its memory can be up to 1 Mwords (64 bits plus 8 error correction bits) of 50 ns semiconductor memory (1024 bits/chip). The scalar part of the CPU works in three-address mode with eight general-purpose arithmetic registers and eight address or index registers. A second level of register memory, only available for transfer instructions to the above-mentioned registers, provides 64 64-bit and 64 24-bit registers. The vector part of the CPU operates on eight "vector registers" each of 64 words (of 64 bits). These vector registers are loaded from memory and restored to memory when necessary and the typical vector instructions use two of them as input and one as the destination for operations dealing with vectors of up to 64 words. Longer vectors must be broken down into pieces for processing. The memory locations which are brought into a vector register need not be consecutive but must have constant spacing.

In vector mode the CPU can produce a result every cycle, or 80 MFLOPS, but, owing to parallel instruction execution and a large degree of autonomy between the scalar part of the CPU and the vector part, certain inner loops have achieved up to 138 MFLOPS or beyond.

1.2.4 Burroughs BSP ⁴⁾. (Figs. 28 and 29). The Burroughs Scientific Processor was announced in

early 1977 and deliveries are only expected by 1979 at the earliest. It is conceived as an integrated "back-end" processor for a Burroughs 7800 (or 7700) with which it communicates almost uniquely in terms of files (or rather "chapters" of files) which reside on a file memory of up to 64 million 48-bit words. This file memory is constructed using 64 Kbit charge-coupled device (CDC) technology and is accessible both from the B7800 and the BSP. There is no other input/output device for the BSP and it runs a single user program at a time.

The BSP CPU consists of a scalar processing unit (SPU) which operates, using an 80 ns cycle, on 16 48-bit general-purpose registers. The SPU also uses 16 120-bit wide vector description registers. The parallel processor contains 16 Arithmetic Elements (AEs), which execute floating point operations in "lock step" (i.e. each element is executing the same instruction but on different data) at the speed of one result per 320 ns, giving a top speed of 50 MFLOPS. The vector instructions for the AEs are microprogrammed and very complex; up to five operands in one instruction, several different operators, field manipulation and editing operators and, even, special instructions for FORTRAN FORMAT conversion!

The BSP is the only announced super-computer which is based on the ILLIAC IV ⁵⁾ experience. ILLIAC IV was conceived in the early 1960s at the University of Illinois as a machine with four "quadrants" of 64 identical processors each of which would execute in lock step on the same memory. Only one machine was built and with only one quadrant which has been in use at the NASA Ames Laboratory (Calif.) for several years ⁶⁾.

1.3 A couple of recent array processors

For the second category of users mentioned in Section 1.1 there is a reasonably important number of "attachments" available for specialized high speed computation. The relative inexpensiveness (ranging from \$100,000 and upwards) of such "array processors" normally results in a high degree of specialization with either very limited applications in mind or very special programming requiring a substantial investment. Seen from an architectural point of view we can split them into two classes:

- i) those connected via a channel interface to the main computer,
- ii) those which access the main memory of the host computer directly.

Each of these types will be discussed under the heading of an existing processor.

1.3.1 Floating Point Systems Inc., Array Processor AP-120B ⁷⁾ (Fig. 30). In the mid-1960s, almost every large scale "scientific" computer such as the CDC 6000 series, IBM System/360 and UNIVAC 1100 series could be equipped with an "array processor" from the manufacturer to speed up simple floating point operations such as vector inner products. These processors were in considerable use for certain types of calculations such as, for example, seismic data processing. As the speed of large central processors has increased without the channel speeds increasing correspondingly it is now, however, getting too slow to send a couple of vectors out over a channel for an inner product operation; on a normal 1.5 Mbyte IBM S/370 channel it takes over 5 μ s per pair of single precision numbers to send them out; much slower than to perform the operation on an S/370-168.

But, at the same time, semiconductor technology and microprogramming architectures have made it very attractive and cheap to build small, specialized processors which can generate 5-10 million results per second. The results have been that such array processors are now offered to perform relatively complete calculations in the form of, for example, a FORTRAN subroutine call: matrix inversion, complex to complex Fast Fourier Transformation, etc. Attached to a minicomputer this can be a very cost effective way of performing quite extensive calculations, in particular in the general area of "signal processing".

The FPS AP-120B is such an array processor of which several hundred units have been installed since 1975 and which is available with a channel interface to many minicomputers as well as to IBM S/370. It runs with a cycle time of 167 ns and has two independent arithmetic units in the CPU: A floating point multiply unit and a floating point adder. Both units are fully pipelined (i.e. can start a new operation every cycle) and can finish in three and two cycles, respectively. The operand memory is available in up to 500 Kwords (38 bits) with 167 ns cycle time, and the separate program memory is 50 ns cycle time, 4 Kwords of 64 bits each; a micro-instruction which controls both arithmetic units in a zero address mode needs 64 bits. There are 64 floating point registers and 16 operand address registers. An additional "table memory" of up to 64 Kwords of 38 bits is composed of read only memory and normal memory of 167 ns cycle time. When both arithmetic units are active the FPS AP-120B delivers two results every 167 ns or a

result rate of 12 MFLOPS -- quite an acceptable performance for a "box" which in a small configuration is comparable in price to that of a large minicomputer.

1.3.2 International Computers Ltd. (ICL), Distributed Array Processor (DAP)^{8,9)} (Figs. 31-35). Another approach to make "array processors" cost effective with today's large-scale general-purpose computers is to have them operate directly on the main memory and thus to avoid the overheads involved in transmitting the data across the input/output channels. An important problem with this design is that of memory access conflicts: either the array processor has to be able to handle a delay in operand access thus greatly complicating the design and diminishing the efficiency or a way has to be found whereby the operand memory is left to the exclusive use of the array processor when it is active.

International Computers Ltd. have, since 1972, worked on such an integrated design inspired by the SOLOMON concept¹⁰⁾ and revived by J.K. Iliffe: the Distributed Array Processor (DAP). DAP was announced in early 1978 and will be available on the ICL 2900 series from 2960 and upwards. So far, one DAP is scheduled for installation at Queen Mary College, University of London, in 1979.

DAP consists, basically, of a matrix of 64×64 1-bit processing elements of which each has access to 4096 bits of storage. Each processing element can communicate with its four nearest neighbours (N, E, S, W) with a choice of "geometry" around the edges of the matrix. DAP is integrated into the host computer as 16 Mbits of normal memory, and it is perhaps conceptually easiest to view it as a box where each main memory word has its bits allocated to different, consecutive processing elements and where each processing element has a column of 4096 bits available representing bit number n in 4096 different 32-bit words. The DAP has to be programmed to perform floating point arithmetic either by having, for example, 32 consecutive processing elements working together on a main storage word, or by having all 4096 processing elements working on words stored in the columns. All processing elements execute in lock step (i.e. the same instruction), but can have different offsets within their "column" of 4096 bits and can be "inactivated" individually. Efficient algorithms exist for the "transposition" of numbers from columns to rows and vice versa.

DAP executes at a cycle time of about 200 ns, which makes it capable of doing 32-bit floating point arithmetic in times between 135 and 330 μ s (addition and division) per processing element, corresponding

to 12-30 MFLOPS. On table look-up or scanning operations it is very, very fast as it can be considered as an associative store in such applications. The interface between a FORTRAN program and DAP is on a double subroutine level: the user program calls subroutines written in DAP-FORTRAN (operating only on data in COMMON blocks which are allocated to the DAP memory) which in turn is practically interpreted using a large set of general-purpose DAP routines written, laboriously, at the level of the individual processing elements.

2. The Architecture

2.1 Technology, software and other limitations

What are, then, the problems in providing cost effective "super-computing" and what are the solutions proposed by the various pioneering vector computers currently available? The three large categories of problems are:

- the technology improvements are no longer sufficient to improve computing speed by significant factors -- a CDC 7600 delivered in 1969 had a cycle time of 27.5 ns and a redesign in today's technology would probably find it very hard to get below 10 ns. Hence, to get speed improvements of factors 5 and 10 one needs to explore new architecture.
- When the architecture cannot be that of a family (e.g. CDC Cyber series or IBM S/370) then the new computer needs a completely new operating system and compilers. The market for these super-computers is relatively small; even of the CDC 7600 and the IBM S/360-195, which were general-purpose super-computers, only about 60 and 20, respectively, were installed. Hence one needs to find ways of reducing dramatically the cost of developing the operating system and the compilers.
- Within the CPU itself so many levels of buffering and "pipelining" have to be explored that interruptions such as context switches or simply strictly unvectorizable sequences of instructions can represent very significant overheads. In certain designs where vector operations are executed at the rate of 5-10 times that of a CDC 7600, scalar work may be 2-5 times slower than on a CDC 7600, such that the time of the previous inner loop becomes negligible in comparison with, for example, the task of formatting some data for output in graphical form.

In the following sections these various problem areas are discussed and an attempt is made to analyse

the solutions adopted in order to isolate significant trends.

2.2 CPU and memory architecture

2.2.1 Memory speeds and sizes. The progress in memory technology has been most decisive for getting higher speeds at a reasonable cost and reliability. Today, the CRAY-1 is delivered with up to 1 Mwords (64 bits) of 50 ns bipolar memory and it will probably be possible quite soon to reach 4 Mwords using 4096-bit memory circuits. With 16 independent memory banks, the memory bandwidth is 320 million 64-bit words per second -- quite sufficient to feed the CPU and keep the input/output going without conflicts.

On the CDC STAR-100 of much older conception a memory bandwidth of 200 million 64-bit words per second on sequential accesses is achieved by using very long words (512 bits) in the memory itself and by having 32 banks. The core memory cycle time is 1.28 μ s, making it very slow just to access one operand at random. In both of these machines other limitations prevent the CPU from fully exploiting the memory bandwidth under normal circumstances.

In order to treat large problems efficiently on vector oriented machines it is necessary to have very large memories for the vectors treated, the intermediate result vectors and the logical condition vectors which often have to be used as a substitute for branches in order to gain speed. Hence a memory size of 1 Mwords has to be seen as the strict minimum of what is necessary today.

In order to increase the memory sizes to four, or even 16 Mwords without increasing the cost too much it will probably be necessary to go to Metal Oxide on Silicon (MOS) memories with a cycle time of between 100 and 200 ns. There will be no problem with bandwidth if there are enough banks and the problem of access time is not very severe owing to all

the overheads which are involved anyway in getting a word, checking it, and storing it in a register; a total of 137.5 ns is needed on the CRAY-1 to access a word. So even with a memory of, for example, 150 ns cycle time, or three times slower, the real access time would only increase by 73%, to 237.5 ns.

The preceding discussion has concentrated on the various main memories, but it should not be forgotten that they all have a hierarchy of memories ranging from the operand registers and instruction buffers in the CPU through the main memory to larger backing memories which are still mostly disks, but where the BSP CCD memory of 1 ms access time, 75 Mbytes per second transfer rate and 64 Mwords maximum size may be the best indicator of what will happen in the future. It is not impossible either that the concept of a relatively small main memory ($\sim 100,000$ words) might re-appear almost to be treated as a register memory by the program and that the "main memory" would be situated "behind" that one with a few million words.

2.2.2 Vector processing

A basic common design assumption for the supercomputers discussed is that a large proportion of the algorithms to be executed can be expressed as "vector operations", i.e. relatively simple operations to be executed (conceptually) simultaneously on a number of elements with fixed spacing in memory to allow prefetching of operands. Thus a single instruction may execute the equivalent of simple FORTRAN DO LOOPS like:

```
DO 1 I=1,N      or even   DO 1 I = 1,N
1 C(I) = A(I)*B(I)         DO 1 J = 1,M
                           1 C(I,J) = A(I,J) + B(J,I)
```

By expressing such operations in one instruction one eliminates the (large) overheads involved in all the instruction issues, decoding and branches ($\sim 15\%$ of instructions executed in many programs) normally

Table 1
Memories

	ASC	BSP ^{a)}	CRAY-1	STAR-100	CDC 7600
Cycle time (ns)	160	160	50	1280	275
Word length (bits)	256	48	64	512	60
Banks	8	17	16	32	32
Max. size (Mbytes)	4	48	8	8	0.5
Bandwidth total (Mbytes)	1600	600	2560	1600	270

a) Parallel memory, only for vector operands.

necessary for such a loop and one tells the hardware well in advance where the operands are to be found. This predictability allows the vector processors to prefetch operands from memory and thus achieve the rate of tens of millions of results per second.

The two major techniques used in processing operands of a high rate in an arithmetic unit are:

- streaming operands through a pipelined arithmetic unit capable of producing a result every CPU cycle, and
- making multiple processors work on the same instruction (lock step) but on different operands.

Streaming. The method most commonly adopted today for the processing of vectors uses as short a cycle time as possible in the CPU and streams operands through generating about one, two or four results per cycle, depending on the computer, in the straightforward case of element-by-element multiplication. The CPUs are constructed such that, although a floating point multiplication may take many cycles (seven on the CRAY-1), each step is independent of the previous one allowing a new set of operands to be passed to the CPU every cycle ("pipe-lining").

An advantage of this approach is that it is conceptually simple to think of it as a sequence of scalar operations and that it can be implemented in such a way that work on short vectors or even scalars can be handled at a relatively small loss in speed -- on the CRAY-1 scalar work is only 4 (addition) to 7 (division) times slower than work on long vectors.

An important disadvantage of the approach taken by the ASC and STAR-100, where the operands are fetched from memory and the results stored back into memory, is that it requires a significant "set-up" time of each vector instruction to get going -- tens of CPU cycles normally. This is partly due to the complicated memory access hardware which has to make sure it can guarantee a pair of operands per cycle out of the memory once the execution of the instruction starts, and the result is that top speed is only obtained on very long vectors and that multiple pipes, like on the ASC, become inefficient to use on relatively short vectors (the break-even point is around 100 elements). Since many programs spend a lot of computing time in calculations on short vectors this represents a major handicap.

The CRAY-1 achieves good performance on short vectors by making the vector operations register-to-register (8 vector "registers", each of 64 words,

are treated exactly like accumulators on conventional computers). This is an interesting approach since it also limits the length of vector operations to a point where one does not need to interrupt them because of input/output activities but can wait for completion instead. Interrupts in the middle of the execution of one instruction create very significant logic problems on a machine like the STAR-100 but cannot be avoided owing to its virtual memory architecture. The weakness with the CRAY-1 approach is, of course, that peak performance in sequences like:

- load first operand vector (≤ 64 words) to a vector register
- load second operand vector (≤ 64 words) to another vector register
- multiply into a third vector register
- store the result vector into memory

require four machine cycles per result rather than one on the STAR-100 or the ASC. But in many cases several arithmetic operations can take place in the registers without having to store the intermediate results. In such cases, and provided the instructions are consecutive, the subsequent operations start as soon as the first element of the result vector is available in the result register, thus minimizing the overheads by a "chaining" process.

Another point to be remembered about pipelined CPU designs is that none of them currently provide for a fully pipelined division; a drop in speed of a factor of four compared to multiplication is more or less the rule. The CRAY-1 provides a fully pipelined approximate reciprocal (14 cycles) which has to be "improved" by one more explicit iteration and the other designs use subgroups of iterations within the pipeline design.

Multiprocessor. The BSP provides for 16 general-purpose arithmetic elements which execute in lock step. The DAP provides 4096 1-bit arithmetic elements executing in lock step -- the equivalent of about 850 of the 48-bit BSP arithmetic elements. This approach allows the duplication of relatively slow hardware, 320 ns to perform an arithmetic operation on a BSP arithmetic element, 200-300 μ s on a DAP one, and still achieves very high performance on vector operations.

The major advantage of this approach is that future speed improvements still seem possible both by having more elements and by technology improvements, since we are in a range of technology where the speed of signal propagation is not yet the limiting

factor. This is a different situation from that of pipeline processors, where one is now close to physical limits (12.5 ns cycle time for the CRAY-1) and where speed improvements will require architectural changes resulting in a significant amount of software development and conversion costs.

The most striking disadvantage is that the speed of calculations which refuse to be "parallelized" will be relatively very slow: a factor 16 on the BSP and orders of magnitude on the DAP. This will certainly limit the immediate applicability of these machines compared to a CRAY-1, which is already more than two times faster than a CDC 7600 on completely scalar problems. On the other hand, unless new super-miniaturized technologies become commercially available for the development of large computers the only way to provide significantly faster computers in the future will be the effective use of multiple processors on single problems.

2.2.3 Scalar processing. It is easy to see that the ultimate performance of a super-computer on a real program depends critically on the speed at which it processes the parts of the code which are non-vectorized. Table 2 compares hypothetical super-computers with the CDC 7600 as a function of the performance gain on vector and scalar processing and the percentage of the code execution time which can be vectorized.

Table 2

Relative performances of hypothetical super-computers depending on degree of vectorization. (Unit is CDC 7600 speed.)

Vector/scalar speed relative to the CDC 7600	Degree of vectorization		
	50%	75%	90%
15/0.2	0.4	0.8	1.8
15/0.5	1.0	1.8	3.8
15/1	1.9	3.3	6.3
15/2	3.5	5.7	9.1

Even for typical super-computer applications it can be hard enough to achieve more than 50% vectorization on complete programs and it is extremely rare that one can get to 90%. The importance of fast scalar processing in achieving a significant performance improvement over a large general-purpose computer like the CDC 7600 is thus quite clear.

In order to situate the numbers used in this table a little bit better it should be noted that

the speed factor of 15 for vector processing is quite optimistic for the super-computers of today; but it is not a very critical parameter. On the scalar performance the STAR-100 is generally thought to be quite close to 0.2, the CRAY-1 about 2 and the ASC about 0.5. It is difficult to give an estimate for the BSP yet, but it seems as if its specialized scalar processor might bring it into the general area of about half a CDC 7600 on scalar work. All these numbers are very rough estimates based on a few applications only.

The way of achieving best possible scalar performance is to have effective instruction buffering and separate hardware for scalar and vector processing so that vector and scalar instruction execution can proceed in parallel.

The instruction buffering is advanced enough on most super-computers to allow several separate but small loops to be kept in the buffers simultaneously but since many loops are quite big and also because context switching invalidates the contents of instruction buffers it is important that the buffers are filled rapidly from the main memory. It is worth noting that the CRAY-1, which is the only super-computer currently to exceed 7600 scalar performance, has its widest memory path to its four independent instruction buffers: 16 full words are transferred in one 50 ns memory cycle. Branch look-ahead with instruction decoding of several possible sequences like on the IBM 3033 is not used in any of these computers; generally instruction issue and decoding stops as soon as one instruction cannot be issued to an operand access or other conflict.

The question of separate hardware for scalar processing on a vector-oriented computer is a particularly difficult one, since it is necessary to decide the level at which the separation between vector and scalar processing takes place: at the one extreme the AP-120B and DAP are array processors only, and scalar work is expected to be handled in the host computer. At the other end of the spectrum the CRAY-1 processes vector and scalar instructions in any mixture using both some common registers and some common functional units; reservation hardware is implemented to make sure that instruction issue stops if there is a conflict. The BSP has a separate register-oriented scalar processor which can handle real calculations and help to prepare the vector instructions for the 16-element array processor.

The trend seems to be in the direction of relatively well separated processors within the same

super-computer for the different sorts of processing required. This will certainly impose some additional burden on the compiler writers and, most likely, on the end user who will be encouraged to structure his calculations to separate scalar and vector work already at the source code level.

2.3 Input/Output

When considering input/output for a super-computer there are three important problems which need to be resolved:

- i) The production of the software necessary to handle general-purpose input/output (i.e. disks, tapes, terminals, card readers, remote job entry, printers, mass memories) is cumbersome and very expensive.
- ii) The amount of interrupts ("context switches") generated by a large volume of input/output and the scalar-type code in the operating system needed to deal with them represent a danger to the ultimate performance of the super-computer.
- iii) The aggregate input/output rate ("bandwidth") has to be very high owing to the large computing potential.

2.3.1 Software and costs. The total market for computers of the category we are discussing here is, currently, less than 10 per year distributed over several manufacturers. With their very special architectures made necessary by the emphasis on speed it is not possible to develop super-computers to be able to use the same operating system, compilers, etc., as other computers produced in large quantities. It is, on the other hand, necessary to develop a sophisticated FORTRAN compiler which recognizes and vectorizes DO-loops, gives informative hints on code restructuring which would allow more efficient execution, etc. It should perhaps be noted that all manufacturers have, so far, developed FORTRAN compilers rather than pushing new languages.

The only way then to cut costs and limit the time needed for the production of software of a super-computer is to restrict the variety of input/output devices and record formats available. Although both the STAR-100 and the ASC have been delivered with a rather full complement of input/output devices and can support all the classical peripheral devices and a variety of access methods, this may prove impractical in the future. Already the CDC 7600 needs a general-purpose front-end computer for all other input/output than tapes and disks and the CRAY-1 only supports high speed disks. The BSP has its CCD memory

as the only input/output device (which is shared with the B 7700/7800 front-end) and the DAP is integrated as a piece of shared main memory in its ICL 2900-series host. The software effort for monoprogrammed machines like the BSP and DAP can then be restricted almost only to FORTRAN cross-compilers (executing on the respective hosts) and FORTRAN run-time input/output routines. The CRAY-1 is in an intermediate position with a multiprogrammed system and software to communicate with a front-end as well as disk support -- but the less than 50,000 lines of code is still very acceptable cost-wise, compared to the millions of lines in a system like MSV.

2.3.2 Context switching. The overheads involved in context switching and input/output processing can be illustrated by the example of the CDC 7600 installed at CERN which is running a general scientific workload in terms of input/output frequencies. A study in 1976 ¹¹⁾ gave the following figures: 3 ms of CPU time between GET/PUT operations and 250 μ s average CPU time in the supervisor to deal with a GET/PUT. For one particular application, FORTRAN compilation, there was a GET/PUT request for every 360 μ s of CPU time. Thus we find an average of about 330 context switches per second, and about 8% of the CPU time spent on input/output in the operating system. If we extrapolated these figures to a CRAY-1 which is generally rated to be five times faster than the CDC 7600 we would get 1650 context switches per second (assuming an unchanged length of the average GET/PUT). An "exchange jump" takes about 1.5 μ s on the CDC 7600, but because of the almost 700 registers in the CRAY-1 CPU an exchange jump could easily have been 10 times slower thus creating a potential performance problem. The design solution adopted on the CRAY-1 is to do only a partial context switch in the hardware (612.5 ns) the exchange jump instruction exchanges the contents of the 16 scalar registers, the P-counter and a few other indicators, and it is left to the operating system to exchange the contents of the vector registers, the extra buffer registers, etc., if it wants to give control to another user program. The operating system can be programmed to use only the scalar registers (or save/restore any other registers it wants to use) and thus the overhead for an input/output interruption is kept down to a minimum.

On monoprogrammed machines like the BSP or, in some sense, DAP, the time spent in context switching for input/output processing will be small but the overheads will not disappear entirely: setting up the jobs, creating a structure where certain routines are

completely vectorizable, waiting for the host computer to provide the data in memory or on the CCD file, etc.

2.3.3 Maximum speed input/output. The total input/output bandwidth for the super-computers under discussion is, in all cases, very impressive when measured as the maximum number of bytes per second which can be passed between the channels and the main memory (or between the CCD file and the main memory on the BSP). Thus for computers like STAR-100, CRAY-1, ASC and even the CDC 7600, total input/output to memory can be sustained at much higher peak rates than the peripheral devices can supply data. The peripherals are, of course, generally disks which operate at a peak rate of about 4 Mbytes per second but which have latencies of the order of 50 ms). The limitations on peak interrupt rates furthermore restrict the number of parallel disk streams to three on the CDC 7600 and eight on the CRAY-1 for instance. This corresponds to a maximum of about 32 Mbytes per second on the CRAY-1 where the central memory access input/output rate would allow 20 times this rate. The BSP uses a CCD file for input/output which is designed to achieve 75 Mbytes per second transfer rates with 1 ms latency -- this high speed disk replacement is certainly a very interesting feature of the whole BSP design.

3. The Programming

3.1 Vectorization, parallel processing

To write a program which executes fast on an array processor it is necessary to "think parallel" in the first place; algorithms have to be chosen in such a fashion that the bulk of computing can be expressed as identical operations on many elements. If we use the FORTRAN DO-loop as an example one has, basically, to avoid IF statements, not to call EXTERNAL FUNCTIONS or SUBROUTINES (which may have side effects), use simple indexing representing equally spaced elements in memory, and avoid formulations where an iteration depends on the results of the previous one [i.e. $A(I) = f\{A(I - 1)\}$].

Special limitations on certain array processors are:

- STAR-100 demands a spacing of one between elements,
- CRAY-1 must produce a vector (not a scalar) as the result in the inner loop (i.e. the matrix multiplication $C_{ij} = A_{ik} B_{kj}$ must not be written with K as the index in the innermost DO-loop),

- DAP requires two-dimensional matrices to be of the dimension 64×64 ,
- ASC demands that the inner DO-loop increment in the case of a 3-level deep nest is +1 or -1
- etc.

There are also particular strengths of some of the computers which can give rise to very efficient code generation:

- STAR-100 has single instructions for such complicated calculations as polynomial evaluation, square root, difference calculations and even for the transposition of an 8×8 matrix,
- the BSP compiler is planned to be able to vectorize both loops with subscripts of the form $A(I(J))$ and to resolve iterations of the form $A(I) = f[A(I - 1)]$ into expressions in $A(1)$, $A(17)$, etc. Here advantage is taken of the multiple processor approach which is more flexible than pipelining for these cases.

3.2 Vectorizing FORTRAN (Figs. 36-42)

All the manufacturers of super-computers discussed here have chosen to invest considerable effort in developing FORTRAN compilers which are capable of recognizing DO-loops (and IF-loops) which can be translated to vector instructions. The ASC compiler¹²⁾ was definitely the pioneering effort and it compiles excellent code if the FORTRAN is written in a way "friendly" to the computer. Vectorizing compilers for the CRAY-1 and STAR-100 are also very well developed now.

Another approach which is being tried, partly in order to ease the transition from a non-vector machine, is to have a (FORTRAN) program which analyses all loops and uses a comprehensive set of vector subroutines, about 75 in all, to output another FORTRAN program where loops are, as far as possible, replaced with calls to these subroutines¹³⁾. Implementation of the subroutines is then done in machine language on such target computers as STAR-100, CRAY-1, CDC 7600, ILLIAC IV. This approach allows the development of a very sophisticated general analyser program which is independent of the target computer; on the other hand some inefficiency is introduced by going via SUBROUTINE calls in all cases.

Whichever approach is used to vectorize, the FORTRAN program which is written for efficient execution on a vector-computer is very hard to read. In order to avoid tests within loops, "state-vectors"

of 0's and 1's have to be generated which are then used within the arithmetic to generate the correct results. Non-ANSI intrinsic functions are provided on the CRAY-1, for example, to avoid the specific generation of state vectors. A primitive example:

```
DO 10 I=N,M
  X(I)=A(X)
  IF(B(I).GT.C(I)) X(I)=D(I)
10 CONTINUE
```

is to be rewritten as follows to become vectorizable:

```
DO 10 I=N,M
  X(I)=CVMGM(D(I),A(I),C(I)-B(I))
10 CONTINUE
```

The CVMGM function will, after compilation, only be called once per 64 elements (the maximum number of elements handled in a CRAY-1 vector instruction), will do the test of $C(I)-B(I)$ and the subsequent merge operations with a few vector instructions and a temporary state vector with one bit per element.

Thus we notice that FORTRAN has a number of drawbacks specific to its use for the expression of algorithms amenable to vectorization:

- all SUBROUTINE and EXTERNAL FUNCTION calls may have side-effects;
- purely syntactical choices, such as the sequence of indexing in nested DO-loops, may prevent vectorization in some cases;
- paucity of standard built-in functions for the handling of vectors and matrices;
- no auto-indexing expression (e.g. $A(*)$ in PL/1);
- no encouragement to write code which is easily vectorized -- one "thinks" in terms of IF's and DO-loops.

If parallel (or vector-oriented) machine architectures are to become more widespread, it will be necessary to stop using a sort of distorted FORTRAN when trying to express parallel algorithms and bring a new language into use. The major reason for this is linked to the fact that the area of parallel machine architecture still is a research topic and that each new computer contains significant innovations and changes. In such an environment the importance of transportability and readability of programs is not to be demonstrated if the end user has to avoid repeated and costly program conversions resulting in discouragement with the whole approach of parallel computing. Since FORTRAN is poor in features to express parallel algorithms without information loss users are forced to use coding tricks

and assembly language in order to solve problems efficiently. The end result is that the FORTRAN programs become non-standard, non-transportable and difficult to read. In such an environment it would be better to give up the illusion that one creates transportable programs for these new architectures by writing them using FORTRAN syntax when the semantics is hopelessly inadequate.

4. SUMMARY

The super-computers of today are becoming quite specialized and we can no longer expect to get all the state-of-the-art software and hardware facilities in one package. In order to achieve faster and faster computing it is necessary to experiment with new architectures, and the cost of developing each experimental architecture into a general-purpose computer system is too high when one considers the relatively small market for these computers. The result is that such computers are becoming "back-ends" either to special systems (BSP, DAP) or to anything (CRAY-1).

Architecturally the CRAY-1 is the most attractive today since it guarantees a speed gain of a factor of two over a CDC 7600 thus allowing us to regard any speed up resulting from vectorization as a bonus. It looks, however, as if it will be very difficult to make substantially faster computers using only pipelining techniques and that it will be necessary to explore multiple processors working on the same problem. The experience which will be gained with the BSP and the DAP over the next few years will certainly be most valuable in this respect.

Bibliography

- [A] G.C. Bell and A. Newell, Computer structures: readings and examples (McGraw Hill, New York, 1971) [A classic.]
- [B] D.J. Kuck, D.H. Lawrie and A.H. Smith, High speed computer and algorithm organization (Academic Press, New York, 1977). [Contains several articles on experience with STAR, ASC, CRAY-1, DAP.]
- [C] K.J. Thurber, Large scale computer architecture: parallel and associative processes (Hayden, Rochelle Park, New Jersey, 1976).
- [D] J.E. Thornton, The history of a computer: the CDC 6600 (Scott and Foresman, Glenview, Ill., 1970). [Illuminating case study of an advanced "scalar" computer.]

References

- 1) N.R. Lincoln, Super-computer development - the pleasure and the pain, *Datamation*, pp. 221-226 (May 1977).
- 2) W.J. Watson and H.M. Carr, Operational experience with the TI advanced scientific computer, *AFIPS - Conference Proceedings*, Volume 43 (1974).
- 3) R.M. Russell, The CRAY-1 computer system, *Communications of the ACM* 21, 63-72 (1978).
- 4) C. Jensen, Taking another approach to super-computing, *Datamation*, pp. 159-172 (February 1978).
- 5) G.H. Barnes, P.M. Brown, Maso Kato, D.J. Kuck, D.L. Slatnick and R.A. Stokes, The ILLIAC IV Computer, *IEEE Trans. Comp. Sci. C-17*, No. 8, 746-757 (1978). Reprinted in Bell and Newell (see [A] in Bibliography).
- 6) F. Walkden, H.A.J. McIntyre and G.T. Laws, A user's view of parallel processors, *Proc. 1976 CERN School of Computing*, CERN 76-24 (1976), pp. 213-226.
- 7) Floating Point System, P.O. Box 23489, Portland, OR 97223, USA.
- 8) S.F. Reddaway, DAP - A Distributed Array Processor, *First Annual Symposium on Computer Architecture*, Florida (December 1973), pp. 61-65.
- 9) P.M. Flanders, D.J. Hunt, S.F. Reddaway and D. Parkinson, Efficient computing with the distributed array processor, *in* Kuck et al., (see [B] in Bibliography).
- 10) D.L. Slatnick, W.C. Borck and R.C. McReynolds, The SOLOMON Computer, *AFIPS Proceedings*, Fall Joint Computer Conference, Vol. 22, pp. 7-107 (1962).
- 11) H. Lipps, DD Division CERN, unpublished study.
- 12) D. Wedel, FORTRAN for the Texas Instruments ASC system, *in* Programming Languages and Compilers for Vector and Parallel Machines, Symposium of the Goddard Institute for Space Studies, New York (March 1975).
- 13) M. Myszewski and D. Loveman, Vectorizer System (Massachusetts Computer Associates, Princess Street, Wakefield, Mass., 1978).

CDC STAR-100

Development started: 1964
 Delivery: 1974
 5 produced: Lawrence Livermore Lab. (2)
 NASA Langley
 CDC (2)
 Timing: Maximum 100 MFLOPS (32 bits) but vector start-up can be a significant overhead for short vectors
 Approx. equivalent to a CDC 6600 on scalar work.

Fig. 1

STAR-100

CPU 40 ns clock period
 256 64-bit registers
 Two floating point pipelined execution units, one string unit
 Macro instructions
 64-bit, 32-bit, 8-bit, 4-bit, 1-bit vector instructions
 100 MFLOPS maximum (32 bits).

Fig. 2

STAR-100

Core memory 524,288 64-bit words (plus parity) in 32 banks
 Second memory possible
 1.28 μ s cycle time for 512 bits (25 Mwords/s)
 3R,2W doubleword trunks (25 Mwords/s each)
 Virtual addressing, two page sizes, 16 associative registers compared in 40 ns, subsequently 20 ns/entry

Fig. 3

STAR-100 INSTRUCTIONS

32-bit or 64-bit
 Vector instructions have an implied stride of 1
 sparse vector format: 0001101001 order vector 5, 6, 8, -4 data
 Scalar is 3-address register-register
 Unusual instructions:
 Transpose an 8×8 matrix in place
 Compare and generate order vector
 Sparse dot product
 $\sum A_N, \Pi A_N, C_N = A_{N+1} - A_N, C_N = (A_{N+1} + A_N)/2,$
 $C_N = (A_N + B_N)/2, C_N = (A_N - B_N)/2, C_N = \sqrt{A_N}$
 Polynomial evaluation
 Translate A per B into C
 Floor, ceiling, absolute, truncate exponent
 Altogether 231 instructions

Fig. 4

STAR-100

I/O 4-12 16-bit I/O channels
 5 Mbyte/s maximum channel speed
 Intelligent stations attached only

Fig. 5

STAR-100

Context switching

Invisible package (16 words)
 Register file (256 words)
 Monitor $\rightarrow$ Job: Exit force instruction
 Job $\rightarrow$ Monitor: Exit force
 Channel interrupt
 Storage access interrupt
 Illegal instruction
 Interval timer = 0
 Virtual addressing only in job mode

Fig. 6

STAR-100

MCU Computer with disk, line printer, operator display system autoloader, microcode load
 4 16-bit counters for performance monitoring
 For example:
 No. of branches in stack
 No. of CPU memory requests
 No. of vector instruction < 64 words
 No. of times a particular instruction
 No. of minor cycles delay due to operand Result conflicts
 Dew-point, freon pressure, temperature monitoring, compressor, power failure monitoring

Fig. 7

CDC STAR-100, STRING ARRAY PROCESSOR

Fig. 8

STAR-100
CPU OVERVIEW

ALL PATHS ARE DOUBLEWORD WIDE

Fig. 9

STAR-100

FLOATING PIPE 1 +, -, *, /, TRUNCATE, ADJUST EXPONENT, CONTRACT, EXTEND, COMPARE; ADDRESS OPERATIONS

Fig. 10

STAR-100

FLOATING PIPE 2 +, -, *, /, TRUNCATE, ADJUST EXPONENT, CONTRACT, EXTEND, COMPARE
REGISTER DIVIDE, BINARY/BCD CONVERSION
SQUARE ROOT, VECTOR DIVIDE, VECTOR MULTIPLY

Fig. 11

STAR-100

STRING PROCESSING

Fig. 12

STAR-100

INSTRUCTION BUFFERING

1 WORD / 20 ns LOAD SPEED
CLEARS ON A BRANCH OUT
CONSECUTIVE ADDRESSES

Fig. 13

CRI CRAY-1

Development started: 1972
Delivery : 1976
7 orders: Los Alamos Scientific Laboratory
National Center for Atmospheric Research
European Centre for Medium Range Weather
Forecasting
US Department of Defence (2)
United Computing Services
Lawrence Livermore Laboratory (MFE)
4 "chip types", no capacitors, 3 in. wires,
8 gates/cycle, quite small.

Fig. 14

CRAY-1

Instructions 2

Unusual features

Vector chaining
Some constants (0.5, 1, 2, 4)
Approximate reciprocal and 2-SJ*SK instruction
Population count
Branch addressing to parcel (24 bits)
180 instructions

Fig. 18

CRAY-1

CPU

12.5 ns clock period
585 64-bit registers, 72 24-bit ones
12 fully pipelined specialized functional units
80 Mips maximum
80 - 138 - 250 MFLOPS maximum (64 bits)
2's complement, no floating point divide, no
fixed point multiply.

Fig. 15

CRAY-1

Memory

1,048,576 64-bit words (plus secded) in 16 banks
50 ns cycle time, 137.5 ns access time
80 Mwords/s block transfers, 320 Mwords/s
to instruction buffers
Addressing offset by one fixed base address
register (set to zero in monitor mode)

Fig. 16

CRAY-1

Delays

Functional units: only vector instructions

Registers: Operand and result register avail-
ability
(Note vector chaining however)
Only one result path to A, B, S and T
registers

Block memory copies

Vector load/store 8 or 16 word incre-
ment
(Note it prevents chaining)
V, B or T load/store wait for I/O to
stop and other block transfers to
finish

B and T register block copy stop
issue for 5, 6, or 14 CPs, cannot
issue themselves if there is an A
or S register reservation

Memory Scalar or I/O references conflicting
in rank B of the C, B, A memory
access network.

etc.

Fig. 19

CRAY-1

Instructions 1

16-bit or 32-bit
Scalar is 3-address register-register
Vector is 3-address register-register (can use an
S register as one of the input streams),
VL determines number (1-64) of elements
No bit, byte, decimal facilities
Vector loads/stores are fixed strides, but not
only 1
VM register (64 bits) control merges of two vec-
tors (V-V, S-V, O-V, I-V)
VM register is set from S or from one vector
(0, ≠ 0, pos, neg).

Fig. 17

CRAY-1

I/O

12 16-bit I channels and 12 O channels
One 64-bit register per channel
Maximum rates:
one channel 80 Mbytes/s
six channels 160 Mbytes/s
24 channels 640 Mbytes/s

Disk controllers or intelligent stations
attached only.

Fig. 20

Fig. 21

Fig. 22

Fig. 23

CRAY - 1

FUNCTIONAL UNITS 2

CRAY - 1

FUNCTIONAL UNITS 1

Fig. 24

Fig. 25

CRAY - 1

INSTRUCTION BUFFERING

- NO BRANCH IS 2 CP's .
- BRANCH IS 5 OR 14 CP's, EXCEPTIONALLY 25 CP's WITH INSTRUCTION SPLIT OVER TWO BUFFERS.

- 4 BUFFERS USED IN A CIRCULAR FASHION.
- EACH BUFFER STARTS ON A PARCEL ADDRESS DIVISIBLE BY 200
- EACH BUFFER IS INDEPENDENT
- NO LOOKAHEAD

Fig. 26

Fig. 27

BURROUGHS BSP

Fig. 28

BSP CONTROL PROCESSOR

80 ns CYCLE, 3-4 MIPS
STACK FOR LINKAGE (RETURN ADDRESS, REGISTER ADDRESS TRANSLATION TABLE)

Fig. 29

AP-120B (Floating Point Systems)

"ARRAY PROCESSOR"

Cheap, 167 ns cycle, 3 cycle MPY, 2 cycle add
Specialized with separate instruction and operand memory, double instructions for both pipes.
Scalar/vector no difference within the AP
500 Kwords data, 4 Kwords instruction memory,
6 Mips, 1-2 (compiler) - 8 (hand) -
12 (theory) MFLOPS.

Fig. 30

ICL DAP			
Development started:	1972		
Delivery	: 1979 (64 × 64)		
Timing	: 200 ns cycle time, 4096 processors		
	32-bit floating point:		
+	135 μs approx.	30	MFLOPS
*	280 μs "	15	MFLOPS
**2	140 μs "	30	MFLOPS
/	350 μs "	11.5	MFLOPS
✓	200 μs "	20	MFLOPS
Memory	: Two storage modes, variable word length.		

Fig. 31

ICL DAP, DISTRIBUTED ARRAY PROCESSOR

Fig. 32

Fig. 33

ICL DAP MASTER CONTROL UNIT SCHEMATIC

Fig. 34

ICL DAP

Fig. 35

VECTORIZATION

FORTRAN or Assembler in practice

FORTRAN:

complicated compiler analysis necessary to restructure. Even so depends on specific high speed computer

Gains : 25-50% vectorization often easy; 50-70% often possible, hard work; > 75% exceptional

Fig. 36

VECTORIZING A FORTRAN LOOP

OBSTACLES AND TOOLS (1)

1. Lack of knowledge

Call subroutine
External function
Unknown index: $M = J(I)$
 $R(M) = \dots$

Tools: Fork and join (?)

2. Logical structure

GO TO
IF

Tools: Avoid GO TO's
Replace IF's by "state vector"
Type constructions ("DO for all except when")

Fig. 37

VECTORIZING A FORTRAN LOOP

OBSTACLES AND TOOLS (2)

3. Use of variables

Scalars used as temporaries
Vector results overlap with operands
Inner loop result not a vector

Comment

This is getting specific to the CRAY-1
Features on ILLIAC IV, ASC, STAR-100 would allow vectorization in some of these cases

Tools

Restructure using temporary vectors
Reorder the sequence of calculations
Split off the hopeless parts of loops

Fig. 38

EXAMPLES (1)

Initial

```
DO 1 I = 2,N
1 R(I) = R(I)+R(I-1)

DO 1 I = 1,N,2
T = A(I)+B(I)*C(I)
1 D(I) = T+1./T
```

Vectorizable

No
Replace T with T(I)
or
cut out T
or
make T a statement function

```
DO 1 I = 1,N
A(I) = ...
1 B(I) = A(I+1)
```

```
DO 1 I = 1,N
B(I) = A(I+1)
1 A(I) = ...
```

Fig. 39

EXAMPLES (2)

Initial

```
DO 1 I = 1,N
X(I) = A(I)
1 IF (B(I).GT.C(I)) X(I) = D(I)

SUM = 0
DO 1 I = 1,N
1 SUM = SUM+A(I)
```

Vectorizable

```
DO 1 I = 1,N
1 X(I) = CVMGM(
D(I),A(I),C(I)-
B(I))

M = MIN0(64,N)
DO 1 I = 1,M
1 T(I) = A(I)
K = (N-M)/64
L = (N-M)-64*K
DO 3 I1 = 1,K
DO 2 I2 = 1,64
2 T(I) = T(I) +
A(I+M)
3 M = M+64
DO 4 I = 1,L
4 T(I) = T(I)+A(N+
1-I)
SUM = T(I)
DO 5 I = 2,M
5 SUM = SUM + T(I)
```

Fig. 40

EXAMPLES (3)

Initial

```
DO 1 I = 1,N
DO 1 J = 1,M
A(I,J) = 0.
DO 1 L = 1,K
1 A(I,J) = A(I,J) +
B(I,L)*C(L,J)
```

Vectorizable

```
DO 2 I = 1,N
DO 1 J = 1,M
1 A(I,J) = 0.
DO 2 L = 1,K
DO 2 J = 1,M
2 A(I,J) = A(I,J) +
B(I,L)*C(L,J)
```

Fig. 41

THE VECTORIZER

(Research Development Associates)

FORTRAN - FORTRAN using calls to

DYADS: $S = \text{SUM}(U(*) \cdot V(*))$
 $R(*) = S/U(*)$
 $R(*) = U(IV(*))$
 $R(*) = U(*) + V(*)$ etc.

TRIADS: $R(*) = S \cdot U(*) \cdot V(*)$
 $R(*) = U(*) - V(*) \cdot W(*)$ etc.

MONADS: $S = \text{SUM}(U(*))$
 $R(*) = S$
 $R(3) = \text{COS}(U(*))$ etc.

MERGES: $\text{TEST}(W(*).GT.0), R(*) = U(*) \text{ OR } V(*)$
 $\text{TEST}(V(*).NE.0), R(*) = S \text{ OR } U(*)$ etc.

Fig. 42