PB83-176065

Survey of New Vector Computers: The CRAY 1S from CRAY Research; the CYBER 205 from CDC and the Parallel Computer from ICL Architecture and Programming

Deutsche Forschungs- und Versuchsanstalt fuer Luft- und Raumfahrt e.V.
Goettingen (Germany, F.R.)

Prepared for

National Aeronautics and Space Administration
Washington, DC

Jan 82

PB83-176065

A SURVEY OF NEW VECTOR COMPUTERS: THE CRAY 1S FROM CRAY RESEARCH; THE CYBER 205 FROM CDC AND THE PARALLEL COMPUTER FROM ICL - ARCHITECTURE AND PROGRAMMING

by

Wolfgang Gentzsch

Translated from the German

TRANSLATION PREPARED FOR THE INTERNATIONAL TECHNOLOGY EXCHANGE DIVISION, OFfice of International Affairs, NATIONAL TECHNICAL INFORMATION SERVICE.

1983

FT

REPORT DOCUMENTATION PAGE	1. REPORT No. DFVLR-FB-82-02	2.	3. Recipient's Accession No. PB83 176065

4. Title and Subtitle A Survey of New Vector Computers: The CRAY 1S from CRAY Research, the CYBER 205 from CDC and the Parallel Computer from ICL. The Architecture and Programming.	5. Report Date Jan 82
	6.
7. Author(s) W. Gentzsch	8. Performing Organization Rept. No.
9. Performing Organization Name and Address Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Göttingen West Germany	10. Project/Task/Work Unit No.
	11. Contract(C) or Grant(G) No. (C) (G)
12. Sponsoring Organization Name and Address National Aeronautics and Space Administration Washington, D.C. United States of America	13. Type of Report & Period Covered
	14.

15. Supplementary Notes Translation of "Ein Überblick über die neuen Vektorrechner CRAY-1S von CRAY Research, CYBER-205 von CDC und den Parallelrechner DAP von ICL - Architektur und Programmierung", Jan 82, 50pp. Original German available as N82-30961.

16. Abstract (Limit: 200 words)

Problems which can arise with vector and parallel computers are discussed here in a user oriented context. Emphasis is placed on the algorithms used and the programming techniques adopted. Three recently developed supercomputers are examined and typical application examples are given in CRAY FORTRAN, CYBER 205 FORTRAN and DAP (distributed array processor) FORTRAN. The systems performance is compared. The addition of parts of two N x N arrays is considered. The influence of the architecture on the algorithms and programming language is demonstrated. Numerical analysis of magnetohydrodynamic differential equations by an explicit difference method is illustrated, showing very good results for all three systems. The prognosis for supercomputer development is assessed.

17. Document Analysis a. Descriptors

*Architecture(Computers); *Computer Programming; *Computer Systems Performance; *Parallel Computers; *Pipelining (Computers); *CRAY 1 Computers; *CYBER Computers; Computation; Computational Fluid Dynamics; Fortran; Magnetohydrodynamics; Technology Assessment

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group(s) 9B NTIS Field/Group(s) 62A, 62B, 62D, 62F

18. Availability Statement Release unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page) UNCLASSIFIED	22. Price

See Instructions on Reverse

/

GERMAN RESEARCH AND TEST CENTRE FOR AIR AND SPACE TRAVEL

Research Report FB 82-02

A SURVEY OF NEW VECTOR COMPUTERS

The CRAY 1S from CRAY Research
The CYBER 205 from CDC and
The Parallel Computer from ICL

ARCHITECTURE AND PROGRAMMING

by

Wolfgang Gentzsch

DFVLR

Institute for Theoretical Fluid Mechanics

Goettingen

January 1982

Contents

Preceding page blank

1. Introduction

The importance of electronic computers in manifold branches of science and technology has continued to grow since the development and induction of the first German electronic computer in the Astrophysics Division of the Max Planck Institute for Physics in Goettingen. Although most computers have been installed for purely commercial reasons, the largest and fastest computers have been and still are being used primarily in the solution of various physical problems.

The rapid development of semiconductor and switching circuit technologies and of signal communication techniques has resulted in substantial improvements in computation times, mainframe sizes and access times for central processing units (CPUs). In fact, the performance of computers has been enhanced so much that for some ten years now serially operating high performance mainframes have been on the market. Such machines can easily perform one million computations per second, vide the CDC 7600 in 1969, the IBM 360 195 in 1971 and the AMDAHL 470/V6 in 1976.

During this same period, increasing numbers of physicists have immersed themselves in a third branch of physics which came into being with the advent of the computer. Thus, in addition to the domains of experimental and theoretical physics, there is now a discipline known as "computational physics". This field involves the transformation of complex physical problems into equations - often in idealised form - which are solved approximately on the computer using numerical algorithms. Physicists are much interested to discover how the approximate solution of a problem will change when a variety of different parameters are varied. By such techniques it is possible nowadays to save the millions of marks which extremely expensive test units would entail. One typical instance involves the numerical solution of magnetohydrodynamic differential equations used to determine stable plasma equilibria at the Max Planck Institute for Plasma Physics in Garching near Munich. These studies are providing insights which will enable fusion reactors to be built in the future. Another instance involves the numerical solution of Navier-Stokes differential equations to determine the physical properties of the flow around objects having specific configurations, which is being carried out at the Institute for Theoretical Fluid Mechanics within the DFVLR (German Research and Test Centre for Air and Space Travel) at Goettingen. Computations of this type make possible a reduction in the number of very costly wind tunnel tests which need to be performed.

The other side of the coin is that the huge strides made in these fields over the past two decades - which are due primarily to technological advances - have also substantially raised the demands now being made as to what computers might be expected to accomplish.

Since little improvement in the performance of serially operating computers, i.e. so-called von Neumann computers, has been achieved over the past ten years, several computer companies in the seventies started to interest themselves in new architectures. Machines known as vector and pipeline computers have been on the market since 1976. These include the CRAY 1/1S, the STAR 100 and the CYBER 203/205. Parallel computers represent another group and significant examples are the ILLIAC IV from Burroughs, the DAP from ICL and the HEP from Denelcor.

We shall concern ourselves here exclusively with the latest systems from CDC, CRAY and ICL, namely with the mainframes CYBER 205, CRAY 1S and DAP. After a brief outline in Chapter 2 of the principal differences of the individual machines as compared to serial computers, Chapter 3 will be devoted to the languages CRAY FORTRAN, CYBER 205 FORTRAN and DAP FORTRAN and reference will be made to a simple yet typical problem for many applications, viz. the addition of two data subsets. In Chapter 4 we consider the numerical solution of modified magnerohydrodynamic differential equations in two dimensions using the easily programmable single step procedure which works very well on all three systems. This procedure may be adapted without any special problems as a rapid super step procedure. Extracts of the computer program are discussed for all three computers. Our concluding Chapter 5 presents a survey and, in the opinion of the author, a brief overview of the development of computer systems and programming languages up to the end of the eighties.

At this point we should like to express our sincere thanks to those companies and institutions involved in any way in the appearance of this report. In particular, we should like to thank Dr. Schäfer of CDC, Mr. R. Übelmesser of CRAY, Mr. W. Erhard of the University of Erlangen-Nuremburg and Mr. Sutherland of ICL. Thanks are also due to Dr. Müller-Wichards of the DFVLR in Goettingen for his valuable comments.

2. Vector and Parallel Computers

The most important arithmetical instructions, such as addition and multiplication, are carried out in several steps by a conventional computer. For instance, floating point addition comprises the following steps (see Figure 1):

The loading of two elements
Comparison of the exponents
Transfer of the mantissas
Addition of the mantissas
Normalisation to floating point normal format
Storage of the result.

For the serial processing of two vectors by a von Neumann computer, the floating point addition in the above case would require four time cycles for each pair of elements plus the time taken for the loading and storing. For a machine having a processing cycle time of (say) 32.5 nsec (such as the AMDAHL 470) and a main store cycle time of 325 nsec, we obtain in very simplified terms

$$4*32.5 \ + \ 2*325 \ = \ 780 \text{ nsec.}$$

This corresponds to a performance of 1.3 MFLOPS (Million FLOATING POINT Operations per Second) in our example. If two vectors are added, 83% of the time taken for a pair of elements is for loading and storage operations and only 17% for the actual addition.

However, if addition of two vectors from an addition pipe is performed on a vector computer, there will be in each segment of the pipe a pair of operands in a differing state of processing. After a certain start-up time (in our example 325 nsec), a result is produced after

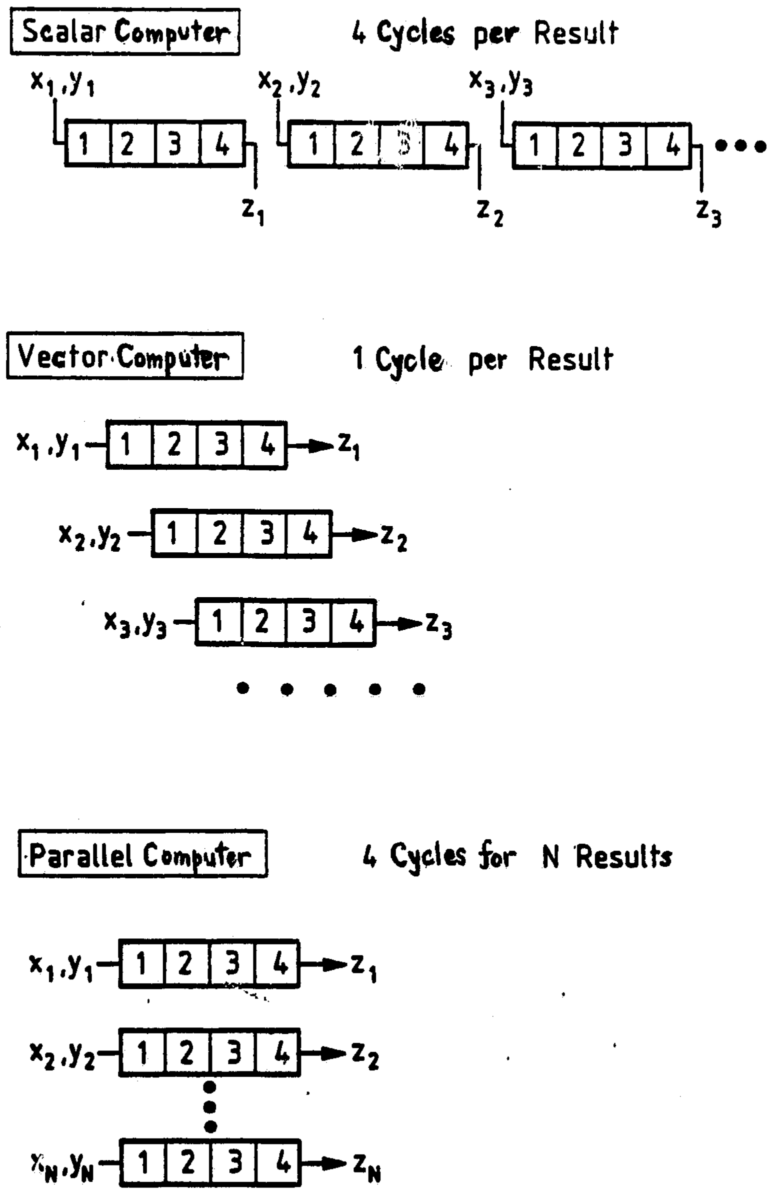


Figure 1. The addition of pairs of numbers $z_i = x_i + y_i$, $i = 1,2, \ldots ,N$ on scalar, vector and parallel computers

each processing cycle. In the above case, for a vector length of 64 we obtain

$$325 + 4*32.5 + 63*32.5 + 325 = 2827.5 \text{ nsec}$$

which is equivalent to

$$44.18 \text{ nsec} \approx 22.6 \text{ MFLOPS}$$

for a single addition. The percentage involving loading and storage is now reduced to 23%.

The operating principle is analogous to the conveyer belt principle used for filling beer bottles in a brewery. While empty bottles enter the "pipe" at one end of the belt, full beer bottles emerge at the other end ready for transportation. Between the starting and finishing points there are at the same time numerous other bottles in varying states of processing (in the segments).

Thus, in pipelining a stream of operand pairs flow through the pipe. In this way, the CRAY 1S can attain up to 23 MFLOPS for vectors whose length is a multiple of 64. The CYBER 205 has a performance of just 40 MFLOPS for addition of vectors of length 64, whereas for vectors of length 1000 even 88 MFLOPS can be achieved on a two pipe version using 64-bit arithmetic. The reasons for this disparity will be investigated in the next chapter. Such performance data for special, individual operations tell us nothing about the overall performance of the particular computers during the operation of extensive production programs found in practice.

Another feasible extension of serial von Neumann computers would be the linking of many such computers to a giant mainframe, though the problems arising therefrom are all too evident. In addition to the high financial expenditure (the flow model processor developed for NASA by Burroughs cost some 100 million dollars), difficult communication problems between the units would have to be solved. Moreover, the interchange of data and control information would be expensive in terms of the time and hardware involved.

A more sensible approach might be the linking of many smaller processors operating under one single control unit. Each of these processors would then carry out simultaneously the same computations.

The maximum performance of such parallel computers is proportional to the number of processors; in the case of DAP from ICL this yields a factor of 64 x 64 = 4096. Although a single processor in the DAP requires 135 μsec to perform an addition (≈0.0074 MFLOPS), the DAP performance for the addition of two 64 x 64 matrices is 30 MFLOPS. A compromise was made for the mutual communication between the processors. Each processor is directly linked to its four neighbouring processors, in particular to their 4096-bit memories. If data are required from the memory of a processor further away, they have to be sent via so-called highways (see Figure 6).

3. Adding Data Subsets on the Three Systems

By means of a simple example we now propose to present a brief review of the three programming languages CRAY FORTRAN, CDC FORTRAN and DAP FORTRAN. This will automatically involve mention of the three architectures used.

Suppose that a scientific problem in the form of a set of differential equations (as, for instance, in Chapter 4) cannot be solved in closed form using the methods of analytical mathematics. It is often sufficient for an approximate solution to be obtained which gives numerical values to certain points within the region of interest. This type of approach is not uncommon in everyday life. If an oil deposit is to be determined, for instance, the surveying engineers often manage with (very costly) drillings made at intervals of 5 km.

The solution of many problems in nature are thus approximated by making use of lattices. If possible, the probable error arising from such a "discrete" solution is also given. We are interested here in the solution of a problem on an N*N lattice with N^2 lattice or nodal points (i,j) with $1 \le i \le N$ and $1 \le j \le N$ as follows:

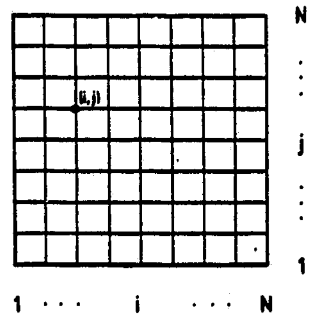


Figure 2. The region for the calculations

The solution is known along the edges of the lattice i = 1, i = N, j = 1 and j = N, so that computations need be performed only for points within the lattice. For instance, the deflection of a plate firmly fastened along its edges is known. The addition of two test series A and B would then be carried out for the subsets

```
A(2,2) - - - A(2,N-1)
'                   '
'                   '
A(N-1,2) - - A(N-1,N-1)
```

and

```
B(2,2) - - - B(2,N-1)
'                   '
'                   '
B(N-1,2) - - B(N-1,N-1)
```

To solve this problem on a serially operating computer, it first has to be translated into a programming language. In FORTRAN, the most commonly used language in the world for scientific and technical problems, the program would assume the following form:

```
   DIMENSION A(N,N), B(N,N), C(N,N)
   N1 = N-1
   READ (3,10) A,B
   DO   1   J = 2,N1
   DO   1   I = 2,N1
1  C(I,J) = A(I,J) + B(I,J)
   WRITE (6,10) C
10 FORMAT (8.5)
   STOP
   END
```

Initially, therefore, space in the computer memory of dimension $3*N^2$ should be reserved for the fields A, B and C, where N is a fixed whole number. After reading in A and B and assigning them to their reserved fields, the addition is performed by columns starting from the lattice point (2,2) and the result returned to memory as C. Finally, C is expressed in a fixed format.

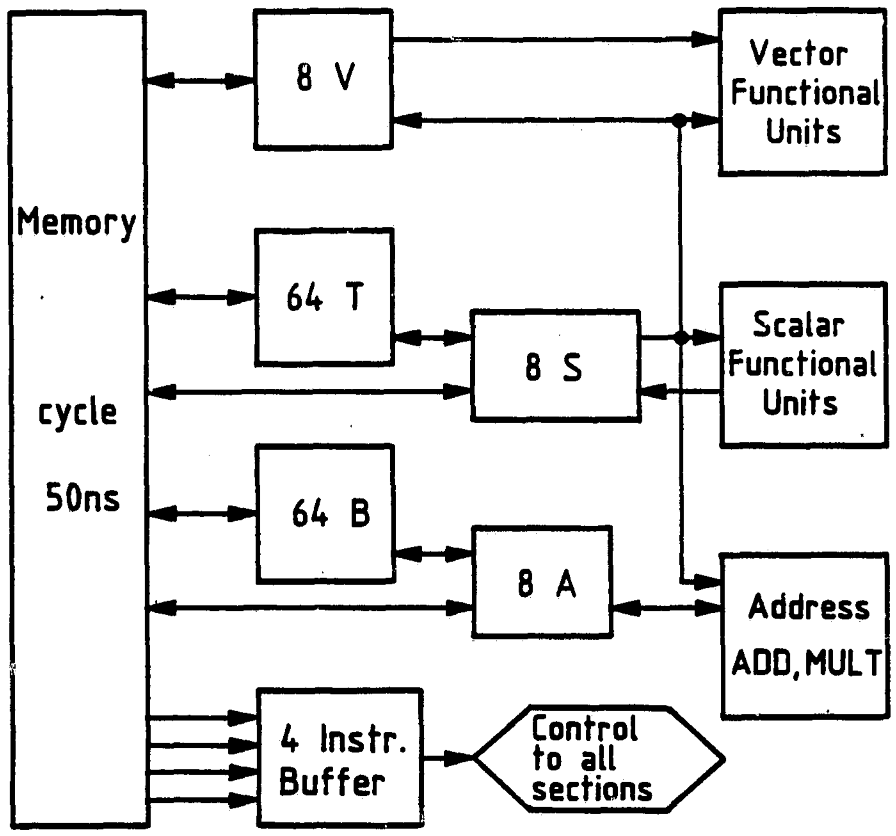
There are no special problems associated with the programming of this for the user of the CRAY 1S. He need not translate his program into some quite different language. Apart from a few extensions, CRAY FORTRAN is virtually identical to normal FORTRAN. The CRAY FORTRAN compiler can read any normal FORTRAN program and automatically vectorises simple DO loops. For multiple DO loops, as in our example, it vectorises in each case only the innermost loop. This is primarily because the CRAY 1S is a computer which carries out vector operations from register to register (see Figure 3). In each of eight vector registers there is space for 64 words from the memory (where they need not be stored as continuous strings). The vector elements move out of the register directly through the corresponding functional units such as addition, multiplication, etc. When 64 elements have been worked through, the relevant register is empty and it is then filled again with 64 elements. At this point the vector operation starts from the beginning once more. Even though the automatic switching to and fro of the data between the memory and the register is very convenient, it consumes a great deal of time (up to 76%). Accordingly, the CRAY 1S attains a maximal performance of 23 MFLOPS for the addition in our problem. If data fields are used for differing computations one after the other, the intermediate results should not go back to the memory but can be further processed directly. In this way the CRAY 1S achieves, for instance, for the operation

$$(x - y) * (x + y), \quad x,y \text{ are vectors}$$

a performance of over 50 MFLOPS.

To solve the above problem with the greatest possible efficiency on the CYBER 205, the program has to be written in CDC FORTRAN roughly as follows:

```
DIMENSION A(N,N), B(N,N), C(N,N)
DESCRIPTOR AD, BD, CD, BITD
N12 = N * (N-2)
BIT, BITD, BIT(N12)
ASSIGN BITD, BIT(1;N12)
BITD = Q8VMKZ(2,N;BITD)
ASSIGN AD, A( N ,1;N12)
ASSIGN BD, B(N,1;N12)
ASSIGN CD, C(N,1;N12)
```



The symbols have the following significance:

V = the vector register

S = the scalar register

A = the address register

B = the fast address register (as intermediate memory)

T = the fast scalar register (as intermediate memory)

Figure 3. Simplified version of the CRAY 1 configuration

As the CYBER 205 is a computer which operates from memory store to memory store (see Figure 4), it will reach its best performance for long vectors (from $N > 200$). The latter should be registered in the memory store in the most continuous manner possible. The DESCRIPTOR AD indicates a location in the memory where the field A(N,N) is stored in columns as an N^2 vector. The information comes from the associated ASSIGN instruction in line 7 of the program. Since only the test values are required for the inner lattice points, the construction of a control vector BIT becomes necessary. The elements of this vector are 0 or 1 BIT depending respectively on whether the lattice point in question is an edge point or an inner point. The appropriate Assembler routine

Q8VMKZ (2,N;BITD)

constructs the following BIT vector

$$\underbrace{001\ldots1}_{N}\underbrace{001\ldots1}_{N}0\ldots\ldots\underbrace{01\ldots\ldots1}_{N-1}\ ,$$

consisting in all of N*(N-2) elements. If the descriptor AD indicates the element (N,1) of the last line in the first column (which still belongs to the edge), the addition will begin only two elements further along because of BIT, i.e. addition begins with the element (2,2). The addition can now be carried out under the control of this BIT vector by means of the Assembler routine

CALL Q8ADDNV (,,AD,,BD,BITD,CD)

Without the BIT vector only vectors of length N could be added. This would give in the case of N = 52, for instance, a performance of around 33 MFLOPS for the CYBER 205 (2 pipe version; 64-bit arithmetic). BIT controlled addition with vectors of length N12 = 48*50 = 2400 enables the CYBER 205 to attain around 95 MFLOPS. It is thus the prime task of the CYBER 205 programmer to arrange his data in continuous vectors. Assistance in the solution of this and similar problems is provided by CDC FORTRAN which contains some 250 novel instructions.

Although the CRAY 1S operates only in parallel, when vectors are subjected to various computer operations, i.e. are channelled through various functional units, just the basic operations of addition and multiplication of vectors are carried out in parallel in 2 or 4 pipes in the case of the CYBER 205. Thus it is that, for instance, the vector elements with uneven location numbers in the first pipe and those with uneven location numbers in the second pipe are processed.

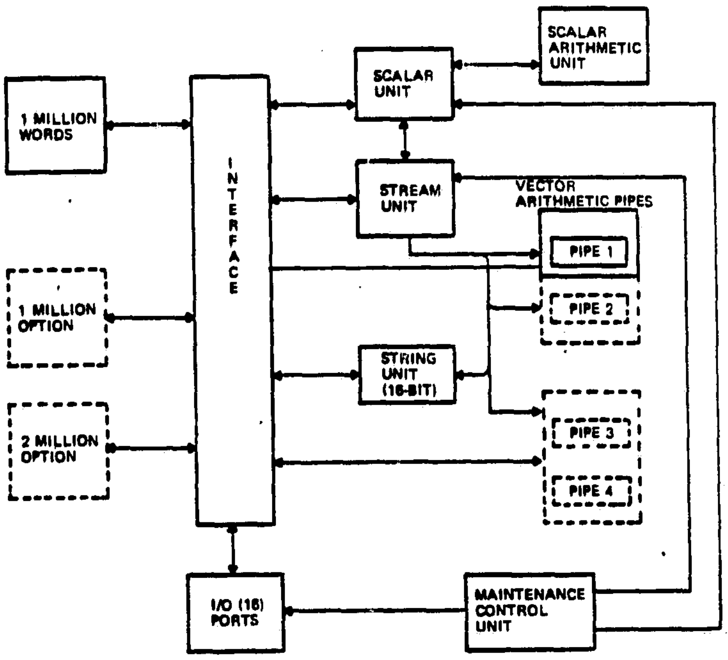


Figure 4. Configuration of the CYBER 205

In contrast with the so-called vector computers from CRAY and CDC, the DAP (Distributed Array Processor) from ICL is a parallel computer. This performs operations for arrays, e.g. 64*64 fields, in parallel on 64*64 identical processors (so-called processing elements) at the BIT level. However, this necessitates a totally new approach which is far removed from that for serial operations used especially in FORTRAN programs. Since many efficient algorithms contain purely serial structures (iterations, recursions, element comparisons, intermediate enquiries, etc.), they need to be restructured or even rejected and new ones developed. The above example assumes roughly the following form in DAP FORTRAN for N $\leq$ 64:

```
   COMMON/ADD1/N,A(64,64),B(64,64),C(64,64)
   READ (3,10) A,B
   CALL ADD
   WRITE (6,10) C
10 FORMAT (8.5)
   STOP
   END
```

```
SUBROUTINE ADD
COMMON/ADD1/N,A(,),B(,),C(,)
LOGICAL INTERIOR (,)
CONVFME (A)
CONVFME (B)
CONVFSI (N)
INTERIOR = ROWS(2,N-1).AND. COLS(2,N-1)
C(INTERIOR) = A+B
CONVMFE (C)
RETURN
END
```

A DAP program always consists of two parts: the host part, e.g. for an ICL 2980, and the DAP part. In the host part endeavours should be made to include all the scalar operations, the input/output and the overall controls for the program. Those parts of the program requiring the longest computing times should be assigned to the DAP part as sub-programs.

The link between the host and DAP parts is established by means of a CALL instruction. Data need not be transferred since they are entered by level at the outset (see Figure 5). When required for use in the DAP, they are first converted column-wise into the individual 4096-bit memories. CONVFME(A) thus means here: convert A from the 2900 FORTRAN memory mode into the <u>m</u>atrix mode of the DAP as a REAL(≈<u>E</u>) value.

Since we do not wish to carry out the addition over the whole 64*64 field, we construct a logical matrix INTERIOR composed of the entries TRUE (≈ 1-BIT) for the points (i,j) of the logical intersection

$$\{2 \le i \le N-1\} \cap \{2 \le j \le N-1\}$$

Whenever the conditions

$$i = 1 \text{ or } j = 1$$

$$N \le i \le 64 \text{ or } N \le j \le 64$$

hold, the entry FALSE (≈ 0-BIT) is made. The addition is thereby carried out everywhere, but transferred to C only for the chosen inner elements. As the result is to be printed out in the host part, C is finally converted from the DAP store mode to the 2900 mode.

In DAP FORTRAN too there are numerous extensions to the language which make it possible for a programmer to write programs in a very straightforward manner. The approach adopted is very similar to that for CDC FORTRAN, though in the case of the DAP problems need to be formulated in a novel parallel structure. To balance this, however, a performance of up to 30 MFLOPS is attained for the above example of matrix addition when the 4096 processors are fully employed, in spite of the fact that the individual processors are relatively slow.

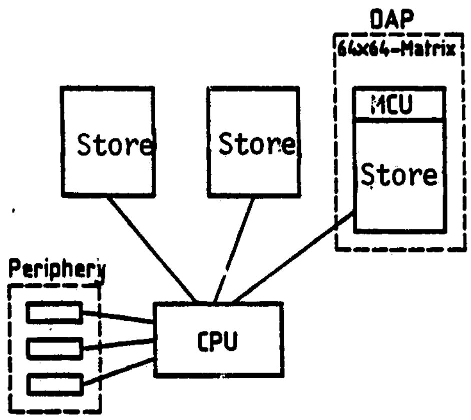


Figure 5. The DAP as part of the 2900 computer

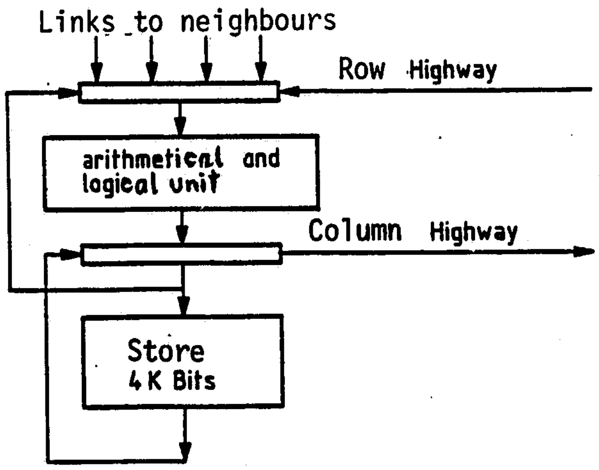


Figure 6. Diagram of an individual processor in the DAP

4. The Numerical Solution of MHD Equations

In order that an even deeper insight may be afforded into the three programming languages CRAY FORTRAN, CDC FORTRAN and DAP FORTRAN, we shall discuss in some detail in this chapter the numerical solution of non-linear, magnetohydrodynamic differential equations. These equations are solved in a cylinder of square cross-section assuming that the ideal gas law holds, that changes of state are isentropic and that the heat conductivity, viscosity and electrical resistance can be neglected. It is then possible to describe a plasma by means of the dynamic equations below:

$$(4.1)\qquad \begin{aligned} \frac{\partial \underline{v}}{\partial t} + (\underline{v}\cdot\mathrm{grad})\underline{v} &= -\mathrm{grad}\ p + \underline{j} \times \underline{B} \\ \frac{\partial \underline{B}}{\partial t} &= \mathrm{rot}\ (\underline{v} \times \underline{B}) \\ \frac{\partial p}{\partial t} &= -\underline{v}\cdot\mathrm{grad}\ p - p\cdot\mathrm{div}\ \underline{v} \end{aligned}$$

with $\underline{j} = \mathrm{rot}\ \underline{B}$ and $\mathrm{div}\ \underline{B} = 0$. Here $\underline{j}$ signifies the current density, $\underline{B}$ the magnetic field, $\underline{v}$ the velocity and p the pressure of the plasma at the point (x,y,z) at time t. The significance of the individual equations is explained, for instance, in {82}. Static solutions, i.e. those representing equilibria for the system, are of especial interest.

It has been shown in {83} that for static solutions the left side of the first equation, i.e. the so-called inertial terms

$$(4.2)\qquad \frac{\partial \underline{v}}{\partial t} + (\underline{v} \cdot \mathrm{grad})\ \underline{v}$$

can be replaced by the frictional term $\underline{v}$. The improved stability of the system then formed is balanced by the less tractable parabolic equations which have to be solved numerically. The original hyperbolic equations are better behaved, though this disadvantage is more or less overcome by the use of implicit procedures or of the rapid, explicit super step procedure {84}.

In what follows we shall restrict ourselves to two dimensions, i.e. concentrate on the numerical solution of the problem in the square cross-sectional area of the cylinder

$$\underline{j} = \begin{pmatrix} O \\ O \\ JZ \end{pmatrix}, \underline{B} = \begin{pmatrix} BX \\ BY \\ O \end{pmatrix}, \underline{v} = \begin{pmatrix} VX \\ VY \\ O \end{pmatrix} .$$

The MHD equations (4.1) then assume the form:

$$\begin{aligned} JZ &= \frac{\partial}{\partial x} BY - \frac{\partial}{\partial y} BX \\ VX &= - \frac{\partial p}{\partial x} - BY \cdot JZ \\ VY &= - \frac{\partial p}{\partial y} + BX \cdot JZ \\ \frac{\partial}{\partial t} BX &= \frac{\partial}{\partial y} (VX \cdot BY - VY \cdot BX) \\ \frac{\partial}{\partial t} BY &= - \frac{\partial}{\partial x} (VX \cdot BY - VY \cdot BX) \\ \frac{\partial}{\partial t} p &= -(VX \frac{\partial p}{\partial x} + VY \cdot \frac{\partial p}{\partial y}) - p(\frac{\partial}{\partial x} VX + \frac{\partial}{\partial y} VY) \quad . \end{aligned} \tag{4.3}$$

The initial and edge conditions are derived f. om the following flow function (see {82}):

$$\psi(x,y) = \frac{1}{4\pi^2} \sin \pi(x-1) \cdot \sin \pi(y-1) \quad ,$$

which also provides the exact solution to the problem in (4.1) or (4.3). The specific initial conditions for t = 0 are:

$$\begin{aligned} p\ (x,y) &= \pi^2 \cdot \psi^2 \\ BX(x,y) &= - \frac{\partial \psi}{\partial y} \\ BY(x,y) &= \frac{\partial \psi}{\partial x} \end{aligned}$$

and the edge conditions for $t > 0$ are:

$$B_{(n)} = 0 , \quad v_{(n)} = 0 ,$$

where (n) indicates in each case the normal components of the vectors $\underline{B}$ and $\underline{v}$.

Discretising of the individual parameters is now carried out at the following points of the cell (I,J):

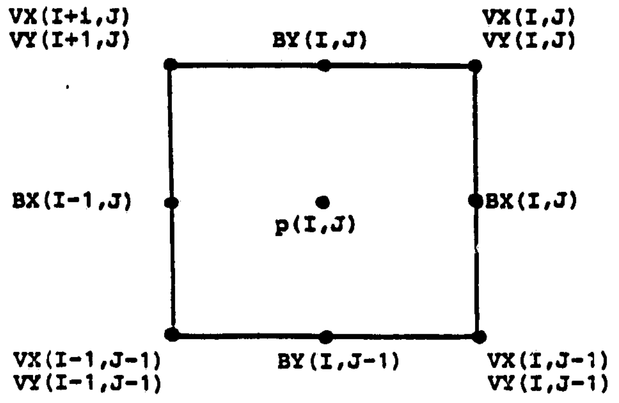


Figure 7

Thus, even for the discrete case, the laws of conservation of flow and mass still hold.

We shall now turn to the detailed discussion of the computer program in the three different languages CRAY FORTRAN, CDC FORTRAN and DAP FORTRAN. The main program has the same form on all three computers:

```
CC          EXPLICIT SINGLE STEP PROCEDURE             CC
CC                                                     CC
CC          WITH CONSTANT STEP WIDTH                   CC
CC                                                     CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C        PROGRAM EINZEL
C   BX,BY  X- OR Y-COMPONENTS OF  B-FIELD
C   VX,VY  X- OR Y-COMPONENTS OF VELOCITY FIELD
C   P,PX,PY PRESSURE X- UND Y-COMPONENTS OF  GRADIENT
         REAL BX(62,62),BY(62,62),VX(62,62),VY(62,62),
     *        P(62,62)
         COMMON /HAIN/ BX,BY,VX,VY,P
         COMMON /CONI/ N,N1,N2,L,N11,NN,LL
         COMMON /CONR/ SDT,DT,DX2,DX,EPS,SV2
C   READING IN PARAMETERS:
C   FIELD SIZE = 5*M, NUMBER OF ITERATIONS  = LL,
C   CUTOFF ERROR= EPS, TIME STEP  = DT
         READ(5,*) M,LL,EPS,DT
         WRITE(6,33) M,LL,EPS,DT
   33    FORMAT(1H1,10X,'M =',I6,/,11X,'LL =',I6,/,11X,'EPS =',
     *E13.6,/,11X,'DT =',F8.4)
C
         N=5*M
         N1=N+1
         N2=N+2
         NN=N*N
         N11=N1*N1
         DX=1./FLOAT(N)
         DX2=DX*DX
```

```
          C
          C     CALCULATION OF THE INITIAL VALUES:
0016            PI=3.141592653590
0017            PSI0=0.02533
0018            P1=PI*PSI0
0019            P2=0.25*PSI0
0020            D2=0.5*DX*PI
0021            DO      1       J=1,N2
0022            DO      1       I=1,N2
0023            X=(FLOAT(I-1)*DX-1.)*PI
0024            Y=(FLOAT(J-1)*DX-1.)*PI
0025            X2=X-D2
0026            Y2=Y-D2
0027            BX(I.J) =-P1*SIN(X)*COS(Y2)
0028            BY(I.J) = P1*COS(X2)*SIN(Y)
0029            P(I.J)=P2*(SIN(X2)*SIN(Y2))**2
0030          1 CONTINUE
0031            DO      2       J=1,N2
0032            DO      2       I=1,N2
0033            VX(I.J) = 0.0
0034            VY(I.J) = 0.0
0035          2 CONTINUE
0036            L=0
0037            SDT=0.
          C
          C     BEGINNING OF THE ITERATION
          C     T=SECOND(T1)
0038            CALL TEXPL
          C     T=SECOND(T1)-T
          C
0039            WRITE(6.2010) L,SV2,SDT,DT
0040            WRITE(6.2000) ((VX(I,J),I=1,N1,M),J=1,N1,M)
0041            WRITE(6.2000) ((VY(I,J),I=1,N1,M),J=1,N1,M)
0042            WRITE(6.2000) (( P(I,J),I=1,N1,M),J=1,N1,M)
0043       2000 FORMAT(1X,6E10.2)
0044       2010 FORMAT(///,11X,'L =',I6,/,11X,'SV2 =',E10.4,/,11X,
               *'SDT=',F10.4,/,11X,'DT =',E10.4,//)
          C     WRITE(6.101) T
          C 101 FORMAT(1X,'CPU-TIME=',F10.5,' SEC')
0045            STOP 1111
0046            END
```

Here are the details of the main program:

After an introductory statement of the variables and reservation of memory space for the fields BX, BY, VX, VY and P, the constants, such as the field size N (N*N = the number of inner points), the local step width, DX = 1/N, and (modified) time step, DT = $\Delta t/\Delta x^2$, are assigned fixed values. Calculation is then made of the initial values from the flow function (lines 16 - 30). The explicit single step procedure is formulated in the sub-routine TEXPL and called in line 38. At the end of the main program there follow instructions for the printout of the results in a given format (L = the number of time steps, SVZ = the cutoff error and SDT = the sum of all L time steps).

The sub-routine TEXPL listed below starts at line 12 by editing the x and y derivatives of the pressure in the right upper corner of the square lattice (I,J) (see Figure 7). The ZJ term in line 29 represents the first of the equations (4.3). After evaluation of the velocity components VX and VY and VB = $\underline{V} \times \underline{B}$, the last three equations from (4.3) are edited numerically in lines 53 - 58 . The edge conditions are taken into account in lines 59 - 68. In the latter part of TEXPL the query is made as to whether a certain mean cutoff error has been reached. If this is the case, there is a return to the main program. Otherwise, the calculation in line 10 of TEXPL is continued.

```
0002          SUBROUTINE TEXPL
        C
0003          COMMON /MAIN/ BX,BY,VX,VY,P
0004          COMMON /CONI/ N,N1,N2,L,N11,NN,LL
0005          COMMON /CONR/ SDT,DT,DX2,DX,EPS,SV2
0006          REAL BX(62,62),BY(62,62),VX(62,62),VY(62,62),
             *      P(62,62),DPX(62,62),DPY(62,62)
0007          REAL VB(62,62),ZJ(62,62),BXM(62,62),BYM(62,62),
             * VX1(62,62),VY1(62,62),DPX1(62,62),DPY1(62,62),
             * PX(62,62),PY(62,62),P1(62,62)
        C
0008          DT1=0.5*DT
0009          DT2=0.0625B*DT1
0010       13 L=L+1
0011          SDT=SDT+DT*DX2
0012                 DO       1        J=1,N1
0013                 DO       1        I=1,N1
0014                 PX(I,J)=P(I,J)+P(I,J+1)
0015                 PY(I,J)=P(I,J)+P(I+1,J)
0016        1        CONTINUE
0017                 DO       2        J=1,N1
0018                 DO       2        I=1,N1
0019                 DPX(I,J)=PX(I+1,J)-PX(I,J)
0020                 DPY(I,J)=PY(I,J+1)-PY(I,J)
0021        2        CONTINUE
0022                 DO       3        J=1,N1
0023                 DO       3        I=1,N1
0024                 BXM(I,J)=BX(I,J)+BX(I,J+1)
0025                 BYM(I,J)=BY(I,J)+BY(I+1,J)
0026        3        CONTINUE
0027          DO       4       J=1,N1
0028          DO       4       I=1,N1
0029          ZJ(I,J)= BY(I+1,J)-BY(I,J)-BX(I,J+1)+BX(I,J)
0030        4 CONTINUE
0031          DO       5        J=1,N1
0032          DO       5        I=2,N
0033          VX(I,J)=(-DPX(I,J)-BYM(I,J)*ZJ(I,J))*0.5
0034        5 CONTINUE
0035          DO       6        J=2,N
0036          DO       6        I=1,N1
0037          VY(I,J)=(-DPY(I,J)+BXM(I,J)*ZJ(I,J))*0.5
0038        6 CONTINUE
0039          DO       7       J=1,N1
0040          DO       7       I=1,N1
0041          VB(I,J)=(VX(I,J)*BYM(I,J)-VY(I,J)*BXM(I,J))*DT1
0042        7 CONTINUE
0043                 DO       8        J=2,N1
0044                 DO       8        I=1,N1
0045                 VX1(I,J)=VX(I,J)+VX(I,J-1)
0046                 DPX1(I,J)=DPX(I,J)+DPX(I,J-1)
0047        8        CONTINUE
0048                 DO       9        J=1,N1
0049                 DO       9        I=2,N1
0050                 VY1(I,J)=VY(I,J)+VY(I-1,J)
0051                 DPY1(I,J)=DPY(I,J)+DPY(I-1,J)
0052        9        CONTINUE
```

```
          DO      10       J=2,N1
          DO      10       I=2,N1
          P(I,J)=P(I,J)*(1.-DT1*(VX1(I,J)-VX1(I-1,J)+
         *                               VY1(I,J)-VY1(I,J-1)))
         *                         -DT2*((VX1(I,J)+VX1(I-1,J))
         *                                *(DPX1(I,J)+DPX1(I-1,J))
         *                                 +(VY1(I,J)+VY1(I,J-1))
         *                                *(DPY1(I,J)+DPY1(I,J-1)))
          BX(I,J)=BX(I,J)+(VB(I,J)-VB(I,J-1))
          BY(I,J)=BY(I,J)-(VB(I,J)-VB(I-1,J))
       10 CONTINUE
                DO       11       I=1,N1
                BX(I,1)   =BX(I,2)
                BX(I,N2)=BX(I,N1)
                BY(1,I)   =BY(2,I)
                BY(N2,I)=BY(N1,I)
                P(I,N2)  =P(I,N1)
                P(N2,I)  =P(N1,I)
                P(I,1)    =P(I,2)
                P(1,I)    =P(2,I)
       11       CONTINUE
          SV2=0.
          DO        12       J=2,N1
          DO        12       I=2,N1
          SV2=SV2+ABS(VX(I,J))
       12 CONTINUE
          SV2=SV2/NN
          IF(SV2.GE.EPS)   GOTO 13
    C     IF(L.LT.LL)GOTO   13
          DT=2.*DT1
          RETURN
          END
```

The program on a CRAY 1S is very similar to the above FORTRAN program. Apart from the removal of the 'C' in the seventh comment line (which is necessary even for certain serial machines), nothing at all changes in the main program. The sub-routine TEXPL too is compiled as it stands by the CRAY compiler and is automatically vectorised up to the DO loop 11 which determines the edge conditions for BX, BY and P. As there are no genuine recursions in this loop, vectorisation can be brought about by means of the instruction

CDIR$ IVDEP.

If the sub-routine were to be written rather more elegantly, for instance by splitting up loop 10 into several smaller DO loops, the CPU time required would be increased by up to 30%. In fact, the CRAY compiler optimises better the more complicated the vector expressions in the innermost loop are. In this way the possibility of chainings, i.e. the linking together of several functional units for (say) (A + B)·C with the vectors A, B and C, are more effectively exploited. Finally, loop 12 is replaced by the CAL (CRAY Assembly Language) function SASUM.

As is to be expected, the program is rather more complicated for the

CYBER 205. The main reason for this is that the data fields have to be made continuous to attain maximal performance from the vector computer. Vectors of length 62, for instance, are thus made up into vectors of length 3844. In this way, the performance for addition and multiplication of vectors - in our case for the whole of TEXPL - is improved by a factor of up to 2.5. In all essentials, the main program remains in CDC FORTRAN and in ordinary FORTRAN.

```
      SUBROUTINE TEXPL
      COMMON /MAIN/ BX,BY,VX,VY,P,DPX,DPY
      COMMON /CONI/ N,N1,N2,N11,N22,L
      COMMON /CONR/ SDT,SV2,DT,DX2,DX
      REAL BX(62,62),BY(62,62),VX(62,62),VY(62,62),
     2     P(62,62),DPX(62,62),DPY(62,62)
      DIMENSION VXM(62,62),VYM(62,62),W(61),II(61),III(61),
     1     BXM(62,62),BYM(62,62),VB(62,62),ZJ(62,62),
     2 VX1(62,62),VX2(62,62),VX3(62,62),VX4(62,62),I1(61),I11(61)
      DESCRIPTOR BIT1D,BIT2D,BIT3D,BIT4D
      BIT BIT1D,BIT1(4000)
      BIT BIT2D,BIT2(4000)
      BIT BIT3D,BIT3(4000)
      BIT BIT4D,BIT4(4000)
C
      N12=N1*N2
      NM12=N2*(N-1)
      N121= N12-1
      N3=N*N2-2
      N1X=N12-3
      DT1=0.5*DT
      DT2=0.0625*DT1
      ASSIGN BIT1D,BIT1(1;N12)
      BIT1D=Q8VMKO(N1,N2;BIT1D)
      ASSIGN BIT2D,BIT2(1;N1X)
      BIT2D=Q8VMKO(N-1,N2;BIT2D)
      ASSIGN BIT3D,BIT3(1;N3)
      BIT3D=Q8VMKO(N,N2;BIT3D)
      ASSIGN BIT4D,BIT4(1;N121)
      BIT4D=Q8VMKO(N1,N2;BIT4D)
      II(1;N1)=Q8VINTL(2,N2;II(1;N1))
      III(1;N1)=Q8VINTL(1,N2;III(1;N1))
      I1(1;N1)=Q8VINTL(N1,N2;I1(1;N1))
      I11(1;N1)=Q8VINTL(N2,N2;I11(1;N1))
   13 L=L+1
      SDT=SDT+DT*DX2
C
      VX1(1,1;N121)=P(2,1;N121)-P(1,1;N121)
     *  +P(2,2;N121)-P(1,2;N121)
      DPX(1,1;N121)=Q8VCTRL(VX1(1,1;N121),BIT1D;DPX(1,1;N121))
C
      VX1(1,1;N121)=P(1,2;N121)-P(1,1;N121)
     *  +P(2,2;N121)-P(2,1;N121)
      DPY(1,1;N121)=Q8VCTRL(VX1(1,1;N121),BIT1D;DPY(1,1;N121))
C
      VX1(1,1;N12)=BY(2,1;N12)-BY(1,1;N12)-BX(1,2;N12)+BX(1,1;N12)
      ZJ(1,1;N12)=Q8VCTRL(VX1(1,1;N12),BIT1D;ZJ(1,1;N12))
C
      VX1(1,1;N12)=(-DPY(1,1;N12)+BX(1,1;N12)+
     *     BX(1,2;N12))*ZJ(1,1;N12)*0.5
      VY(1,2;NM12)=Q8VCTRL(VX1(1,2;NM12),BIT1D;VY(1,2;NM12))
C
      VX1(2,1;N1X)=(-DPX(2,1;N1X)-BY(2,1;N1X)-BY(3,1;N1X))*ZJ(2,1;N1
     *X)*0.5
      VX(2,1;N1X)=Q8VCTRL(VX1(2,1;N1X),BIT2D;VX(2,1;N1X))
```

```
C
      VX1(2,2;N3)=VX(2,2;N3)
     *   +VX(2,1;N3)+VX(1,2;N3)+VX(1,1;N3)
      VXM(2,2;N3)=Q8VCTRL(VX1(2,2;N3),BIT3D;VXM(2,2;N3))
      VX1(2,2;N3)=VY(2,2;N3)
     *   +VY(2,1;N3)+VY(1,2;N3)+VY(1,1;N3)
      VYM(2,2;N3)=Q8VCTRL(VX1(2,2;N3),BIT3D;VYM(2,2;N3))
C
      BXM(1,1;N121)=BX(1,1;N121)+BX(1,2;N121)
      BYM(1,1;N121)=BY(1,1;N121)+BY(2,1;N121)
      VX1(1,1;N121)=(VX(1,1;N121)*BYM(1,1;N121)
     *   -VY(1,1;N121)*BXM(1,1;N121))*DT1
      VB(1,1;N121)=Q8VCTRL(VX1(1,1;N121),BIT4D;VB(1,1;N121))
C
      VX1(2,2;N3)=DPX(2,2;N3)+DPX(2,1;N3)+DPX(1,2;N3)+DPX(1,1;N3)
      VX2(2,2;N3)=DPY(2,2;N3)+DPY(2,1;N3)+DPY(1,2;N3)+DPY(1,1;N3)
      VX3(2,2;N3)=VX(2,2;N3)+VX(2,1;N3)-VX(1,2;N3)-VX(1,1;N3)
     *   +VY(2,2;N3)-VY(2,1;N3)+VY(1,2;N3)-VY(1,1;N3)
      VX4(2,2;N3)=P(2,2;N3)*(1.-DT1*VX3(2,2;N3))-DT2*(VXM(2,2;N3)
     **VX1(2,2;N3)+VYM(2,2;N3)*VX2(2,2;N3))
      P(2,2;N3)=Q8VCTRL(VX4(2,2;N3),BIT3D;P(2,2;N3))
      VX1(2,2;N3)=BX(2,2;N3)+(VB(2,2;N3)-VB(2,1;N3))
      BX(2,2;N3)=Q8VCTRL(VX1(2,2;N3),BIT3D;BX(2,2;N3))
      VX1(2,2;N3)=BY(2,2;N3)-(VB(2,2;N3)-VB(1,2;N3))
      BY(2,2;N3)=Q8VCTRL(VX1(2,2;N3),BIT3D;BY(2,2;N3))
C
      W(1;N1)=Q8VGATHR(BY(1,1;N12),II(1;N1);W(1;N1))
      BY(1,1;N12)=Q8VSCATR(W(1;N1),II1(1;N1);BY(1,1;N12))
      W(1;N1)=Q8VGATHR(P(1,1;N12),II(1;N1);W(1;N1))
      P(1,1;N12)=Q8VSCATR(W(1;N1),II1(1;N1);P(1,1;N12))
      W(1;N1)=Q8VGATHR(BY(1,1;N12),II(1;N1);W(1;N1))
      BY(1,1;N12)=Q8VSCATR(W(1;N1),II1(1;N1);BY(1,1;N12))
      W(1;N1)=Q8VGATHR(P(1,1;N12),II(1;N1);W(1;N1))
      P(1,1;N12)=Q8VSCATR(W(1;N1),II1(1;N1);P(1,1;N12))
      BX(1,1;N1)=BX(1,2;N1)
      BX(1,N2;N1)=BX(1,N1;N1)
      P(1,N2;N1)=P(1,N1;N1)
      P(1,1;N1)=P(1,2;N1)
C
      VX1(1,1;N22)=VABS(VX(1,1;N22);VX1(1,1;N22))
      SV2=Q8SSUM(VX1(1,1;N22))
      SV2=SV2/N11
      IF(L.LT.1000)   GOTO  13
      RETURN
      END
```

For this reason we shall concentrate here on the sub-routine TEXPL. The BIT vectors mentioned in chapter III are an essential feature of this CDC FORTRAN program. They are generated by means of the function

Q8VMKO

if they begin with a 1 (One) and by means of the function

Q8VMKZ

if they begin with a 0 (Zero). Although so-called implicit descriptors are used for the BIT vectors, all other vectors are formulated explicitly. This means, for instance, that

VX1(1,1;N121)

represents that part of the vector VX1 which is stored in the reserved field VX1 of the memory, begins with the element VX1(1,1) and contains

N121 elements (stored by columns). VX1, VX2, VX3 and VX4 are dummy fields in which the results of vector operations are stored at the outset in continuous fashion. Eventual rearrangement of the storage under the control of a BIT vector with the aid of the function

Q8VCTRL (≃ Q8 vector control)

ensures that the relevant edge values are not overwritten with any "false edge operation".

In this way DPX, DPY, ZJ ($\doteq \underline{j} \times \underline{B}$), VY, VX, VXM (the mean value of VX), VYM and VB ($\doteq \underline{V} \times \underline{B}$) are calculated before the last three equations of (4.3) are formulated in CDC FORTRAN.

One has certain problems associated with the determination of the edge values on the upper and lower edges. Although in principle the procedure outlined here with Q8VMKO and Q8VCTRL functions as indicated, from each of the N1*N1 computer operations performed now only N1 results (N1 = N + 1) are required to store the second line of BY in the first line, using for instance

```
      DO 11 I = 1,N1
   11 BY (1,I) = BY (2,I)
```

It is better in such cases to collect the BY(2,I) terms together as in the serial LOOP with the function

Q8VGATHR

and to distribute them in the first line of BY with the function

Q8VSCATR.

For this purpose an index list is needed which in our case will be the vector

II(1;N1) = { 2,N2+2,2·N2+2,..,N*N2+2 }

having N1 elements (N2 = N + 2). This is produced by the function

Q8VINTL(2,N2;II(1,N1).

The GATHER function collects together all the elements in the second line of BY and stores them in the dummy field W. The index list for the elements in the first line of BY is thus given by:

II1(1;N) = { 1,N2+1,2·N2+1,...,N*N2+1 }

which has N1 elements and is constructed by means of

Q8VINTL(1,N2,II1(1;N1)) .

The elements of W are now stored in the first line of BY under the control of the index vector II1 by Q8VSCATR.

In an analogous way,
  the second line of P is stored in the first line of P;
  the N1th line of BY is stored in the N2th line of BY; and
  the N1th line of P is stored in the N2th line of P.

Finally, the absolute magnitudes of the components of VX are written on VX1 and to carry out the precision query the sum of all N2*N2 elements of VX1 is formed by use of

Q8SSUM .

A major problem confronting particularly those inexperienced with the CYBER 205 is the rearrangement of a two or three dimensional data field into a one dimensional field, i.e. a vector. It is suggested that the user of a DAP first familiarise himself with three dimensional fields which can reasonably be stored in the 4096-bit memory of the 64*64 processors. For this reason the two dimensional MHD program here is more or less ideal for the DAP.

In this instance too, the main program is practically unaltered. Only the "C" on the seventh comment card is removed. Since the ICL 2980 at Queen Mary College in London is a relatively slow host computer, the calculation of the initial values is carried out in the DAP part which is constructed in the following way:

```
      ENTRY SUBROUTINE TEXPL 3
      COMMON /HAIN/ VX,VY,P
      COMMON /CONI/ NX,NX1,NX2,L,NX11
      COMMON /CONR/ SDT,SV2,DT,DX2,DX,DT1,DT2
      COMMON /ANFA/ BX(,),BY(,),DPX(,),DPY(,),DPX1(,),DPY1(,),
     *              VX1(,),VY1(,),BXM(,),BYM(,),VB(,),ZJ(,),
     *              PX(,),PY(,)
      REAL VX(,),VY(,),P(,)
      REAL SDT,SV2,DT,DX2,DX,DT1,DT2
      LOGICAL LMAT1(,),LMAT2(,),LMAT3(,),LMAT4(,),LMAT5(,),LMAT6(,)
C
C     KONVERTIERUNG VON 2900 IN DAP
C
      CALL CONVFSI(NX,5)
      CALL CONVFSE(SDT,5)
      CALL CONVFME(VX)
      CALL CONVFME(VY)
      CALL CONVFME(P)
C
```

```
C     AUFBAU DER LOGISCHEN MATRIZEN
C    .TRUE. FUER .GEBIET.
C    .FALSE. FUER .AUSSENRAUM + RAND
C
      LMAT1=ROWS(1,NX1).AND.COLS(1,NX1)
      LMAT2=ROWS(2,NX).AND.COLS(1,NX1)
      LMAT3=ROWS(1,NX1).AND.COLS(2,NX)
      LMAT4=ROWS(2,NX1).AND.COLS(2,NX1)
      LMAT5=ROWS(1,NX1).AND.COLS(2,NX1)
      LMAT6=ROWS(2,NX1).AND.COLS(1,NX1)
C
C
      CALL ANFB
C
   11 L=L+1
      SDT=SDT+DT*DX2
C
      PX(LMAT1)  = P+P(,+)
      PY(LMAT1)  = P+P(+,)
      DPX(LMAT1)= PX(+,)-PX
      DPY(LMAT1)= PY(,+)-PY
C
      BXM(LMAT1)= BX+BX(,+)
      BYM(LMAT1)= BY+BY(+,)
      ZJ(LMAT1)  = BY(+,)-BY+BX-BX(,+)
C
      VX(LMAT2)  =  (DPX+BYM*ZJ)*(-0.5)
      VY(LMAT3)  =  (DPY-BXM*ZJ)*(-0.5)
      VB(LMAT1)  =  (VX*BYM-VY*BXM)*DT1
C
      VX1(LMAT5)= VX+VX(,-)
      VY1(LMAT6)= VY+VY(-,)
      DPX1(LMAT5)  = DPX+DPX(,-)
      DPY1(LMAT6)  = DPY+DPY(-,)
C
      P(LMAT4)  = P*(1.-DT1*(VX1-VX1(-,)+VY1-VY1(,-)))
     *              -DT2*((VX1+VX1(-,))*(DPX1+DPX1(-,))
     *                    +(VY1+VY1(,-))*(DPY1+DPY1(,-)))
      BX(LMAT4)= BX+(VB-VB(,-))
      BY(LMAT4)= BY-(VB-VB(-,))
C
      BX(,1)     = BX(,2)
      BX(,NX2)  = BX(,NX1)
      BY(1,)     = BY(2,)
      BY(NX2,)  = BY(NX1,)
      P(,NX2)   = P(,NX1)
      P(NX2,)   = P(NX1,)
      P(,1)      = P(,2)
      P(1,)      = P(2,)
C
      ZJ    = ABS(VX)
      SV2   = SUM(ZJ)
      SV2   = SV2/NX11
      IF(L.LT.1)        GOTO 11
C
      CALL CONVFSI(NX,5)
      CALL CONVSFE(SDT,5)
      CALL CONVMFE(VX)
      CALL CONVMFE(VY)
      CALL CONVMFE(P)
C
      RETURN
      END
C
```

```
 84        SUBROUTINE ANFB
 85   C
 86        COMMON /HAIN/ VX,VY,P
 87        COMMON /ANFA/ BX(,),BY(,),DPX(,),DPY(,),DPX1(,),DPY1(,),
 88       *              VX1(,),VY1(,),BXM(,),BYM(,),VB(,),ZJ(,),
 89       *              PX(,),PY(,)
 90        COMMON /CONR/ SDT,SV2,DT,DX2,DX
 91        INTEGER V(),PLACE
 92        REAL V1(),V2(),VX(,),P(,),VY(,)
 93   C
 94        V=0
 95        PI=3.1415927
 96        PSI0=0.02533
 97        PSI1=PSI0*PI
 98   C
 99   C    MATRIX V MIT ELEMENTEN VON 0 BIS 63
100   C
101        PLACE=1
102        DO    1     K=1,6
103        V(ALT(PLACE))=V+PLACE
104     1  PLACE=PLACE*2
105   C
106        VX=0.
107        VY=0.
108        PX=0.
109        PY=0.
110        DPX=0.
111        DPY=0.
112        VX1=0.
113        VY1=0.
114        DPX1=0.
115        DPY1=0.
116        BXM =0.
117        BYM =0.
118        VB  =0.
119        ZJ  =0.
120   C
121        V1=PI*(EFLOAT(V)*DX-1.)
122        V2=V1-0.5*DX*PI
123        BX=-PSI1*MATC(SIN(V1))*MATR(COS(V2))
124        BY= PSI1*MATC(COS(V2))*MATR(SIN(V1))
125        P =0.25*PSI0*MATC(SIN(V2)**2)*MATR(SIN(V2)**2)
126   C
127        RETURN
128        END
```

Because the DAP memory is a part of the host memory, the calculated or reserved words from the main program are already in the DAP though in the incorrect memory mode for the DAP. Accordingly, they are converted at the outset as described in Chapter III. Logical matrices are then constructed in analogy to the BIT vectors on the CYBER 205. This too has been discussed in some detail earlier on. The initial conditions are called up from the DAP sub-routine ANFB which serves to construct a vector V having elements from 0 to 63 by means of the built-in function ALT. Parallel processing is thus made possible in lines 123 to 125 of the equations for BX, BY and P. Here, for instance, the matrix

MATC(Z)

(C stands for column) signifies that the first element of the vector

Z occupies the whole of the first line MATC(Z), the second element the second line, and so on. (The same holds for MATR as far as the columns are concerned.)

In lines 37 to 59 the equations are then evaluated, initially on the right hand side of the equality sign over the whole 64*64 field. Here, for instance, P(+,) indicates that the whole P field moves up one line. The last line is filled with zeros and the first is discarded. The results are stored only when a TRUE appears in the logical matrix LMAT.

After insertion of the edge values for BX, BY and P, ABS and SUM follow to determine the mean error. To conclude, those magnitudes which are to be printed out in the main program are converted again.

5. Summary and Overview

The investigations described in Chapters 3 and 4 have demonstrated that mastery of even apparently straightforward problems with new vector and parallel computers involves novel approaches. It is often not cost-effective to use traditional FORTRAN programs without any modification on these new computing systems. The inefficiency which can arise by doing this is illustrated by a comparison of the computation times required for ten very different programs in fluid mechanics {44}. In such a test, CRAY 1 and CYBER 205 machines were faster than the AMDAHL 470/V6 by the following factors:

	CRAY 1	CYBER 205
serial	12.5	5.7
vector	25.9	25.8

Serial means here that the programs used underwent scalar tests with autovectorisers in operation. The extent to which individual programs can deviate from these mean values is shown by the results for the very readily vectorised and parallelised MHD algorithm discussed in the previous chapter.

	CRAY 1	CYBER 205
serial	44.7	10.8
vector	44.7	49.7

The rapid serial times for MHD on the CRAY 1 come about largely as a result of the relatively good autovectoriser of the CRAY compiler which has the function of recognising and vectorising inner DO loops capable of vectorisation. This task is much easier than that of the CYBER autovectoriser which is required to arrange given data structures to produce continuous data fields which are then able to flow through pipes as vector flows of maximum possible length. There is no comparable analogy for the DAP since an ordinary FORTRAN program cannot be run on a DAP without meshes. Moreover, there is no "autoparalleliser" available yet. It is the opinion of the author that the DAP occupies the position indicated in a list of well-known von Neumann computers. An MHD run requiring one minute of computing time on an AMDAHL 470/V6 would require on the following computers:

ICL 2960	30 min	*
UNIVAC 1106	15 min	
CDC 6400	10 min	
IBM 370-158	8 min	*
ICL 2980	5 min	
CYBER 173	4 min	
AMDAHL 470/V6	1 min	*
CYBER 175	45 sec	*
CDC 7600	20 sec	*
STAR 100	10 sec	*
CYBER 203	6 sec	*
DAP	5 sec	*
CRAY-1S	2 sec	*
CYBER-205	2 sec	*

The starred times are based on the experiences and results of the DFVLR group {44}.

The performance curves for von Neumann, vector and parallel computers are also interesting in the case of our problem. Once again the computers are the AMDAHL 470/V6, CRAY 1S, CYBER 205 and the DAP, the problem is the MHD one and N is the number of lattice points in a dimension (the vector length on the CYBER 205 will then be equal to (N-2)*N).

The time steps for the CRAY 1S and DAP curves arise because of the restriction on the content of the vector register to 64 elements or the restriction of the processors to 64*64. Similarly, for the

CYBER 205 vectors will need to be re-scheduled if they are longer than $2^{16} = 65536$ elements, the content of a large page. This would happen in our case for N = 256.

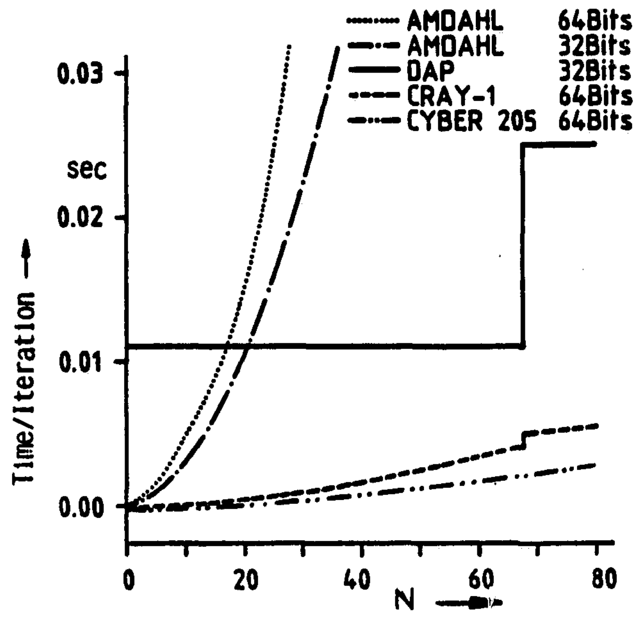


Figure 8

The following picture emerges when the results for individual machines operating the MHD program are compared:

N	10	20	30	40	50	60
CRAY : AMD.	18.7	25.2	30.9	34.6	36.8	38.2
CYBER : CRAY	1.28	1.59	1.55	1.52	1.49	1.46
CRAY : DAP	46.8	18.0	9.4	6.0	4.0	3.0
CYBER : AMD.	23.9	40.0	48.0	52.6	56.4	55.7
CYBER : DAP	60.6	27.9	14.7	9.0	6.0	4.3
DAP : AMD.	0.4	1.4	3.3	5.8	9.1	12.9

It should be borne in mind here that the computations were carried out on the DAP using 32-bit arithmetic and on all other machines using 64-bit arithmetic.

To conclude we shall risk taking a view of the short-term future. In addition to the implementation of 4K-bit chips and 16K-bit ECL chips, CRAY Research, CDC and ICL are now concentrating on refinements in their architectures. In 1981 CRAY extended its already rapid I/O buffer memory up to a maximum of 8 million words. In addition, the transfer rate from this memory to the main memory has been increased by up to 2*850 million bits per second, an important development for possible faster vector operations.

Around 1984 a CRAY 2 machine is expected which should be some 5 times faster than the CRAY 1S. This is to be accomplished by means of new switching circuitry which has been developed in CRAY's laboratories.

Another conceivable improvement would be the attenuation of the so-called "refill" which the CRAY 1S is subject to after each 64 elements when the registers are emptied and filled up again. The underlying cause is not the small size of the vector registers, as is often assumed, but the fact that only one current of vector elements can flow from/to the registers from/to the memory. Enlargement of the registers from 64 to 128 words for vector addition and multiplication would result in an increase of around 10% to just 25 MFLOPS. If it were possible to have two vector flows between the memory and vector registers, some 38 MFLOPS could be attained for addition and multiplication. This, however, would make heavy demands on the hardware.

One of the most interesting developments in the software sector is certainly the production of a Pascal compiler at Manchester University for the CRAY mainframe. This compiler should be able to produce vectorised code.

The CYBER 2XX is expected on the market around 1986. With a station time of 8 ns (CYBER 205 has 20 ns) and 8 vector pipes (CYBER 205 has a maximum of 4), the scalar processor should be 2.5 times faster and the vector processor 5 times faster. The use of 4K-bit chips and 16K-bit ECL chips should extend the memory capacity of the main memory to 8 million 64-bit words.

It can also be assumed that the CYBER software will be considerably enhanced in the next few years. Among other things, the autovectoriser will certainly become more efficient. A simplification of the CDC vector FORTRAN would also be justified, e.g. abbreviation of the Q8 instructions, simplification of the BIT vector construction, simplified statements (such as AD(BITD) = BD+CD), the introduction of certain strictly implemented standard bit vectors, etc.

ICL is primarily concerned with increasing the DAP memory from 2 to 8 million bytes (≈ 2 million 32-bit words) by replacing the 4K chips in the individual processors by 16K chips. A 64-bit arithmetic may then be feasible (roughly double the precision) as this is urgently required for many scientific and technical problems. An improved performance of the individual processors can also be expected (addition now 0.007 MFLOPS), especially after the beginning of October 1981 when an announced joint venture between ICL and Fujitsu will give ICL access to very advanced Japanese microprocessor technology. We can also expect the relatively slow host computer ICL 2980 to be replaced by a much faster machine.

The extension of the 64*64 DAP to a 128*128 DAP should present no problem from the hardware standpoint, though for the end user more complicated programs will mean that full exploitation of the 128*128 = 16384 processors will be even more difficult. The four-fold increase in cost (at least) should be counterbalanced by a performance improvement in the DAP part of around 2.5 times {45}. According to Minsky's hypothesis {89}, which states that multiplication of processors by a factor of p results only in an increase in performance of ln p, the improvement in the performance of the host and DAP parts together should be only 1.4 for the 128*128 configuration. Initially, at least, there will be little improvement on the 64*64 configuration.

After our editorial deadline, the sub-routine TEXPL was revised using the latest CDC FORTRAN. In particular, replacement of the Q8VCTRL statements by so-called WHERE blocks resulted in a further reduction of the computing time by about 20%. Thus, the factor for the comparison of AMDAHL:CYBER 205 is now improved from 55.7 to 69.6. This corresponds to a speed of around 100 MFLOPS for the processing of the production program TEXPL by CYBER 205.

6. Literature

The list below offers but a small selection of the vast number of publications which have appeared to date in the area of vector and parallel computers. Further references are cited in the individual publications listed here.

CYBER 203/205 Software and Hardware

[1] STAR Operating System, Version 1. Reference Manual Vol. 1 and 2. CDC Minneapolis, 1978.

[2] CDC CYBER 200 / Model 203 Computer System. Hardware Reference Manual. CDC Minneapolis, 1979.

[3] CDC CYBER 200 FORTRAN Language 1.4. Reference Manual. CDC Minneapolis, 1979.

[4] CDC CYBER 200 / Model 203. Technical Description. CDC Minneapolis, 1979.

[5] Vector Processing on the CYBER 200. CDC Minneapolis, 1979.

[6] CDC CYBER 200 / Model 205. Technical Description. CDC Minneapolis, 1980.

[7] Kascic, M.J. — Vector Processing on the CYBER 200. Infotech Int. Ltd., Maidenhead, UK 1979.

[8] Kascic, M.J. — Lecture Notes 2+3 of Vector Class. Minneapolis 1978.

[9] Dubois, P.F. u.a. — Operator Splitting on Vector Processors. Lawrence Liv. Lab., UCRL-79316, 1977.

[10] Dubois, P.F. u.a. — Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors. Lawrence Liv. Lab., UCRL-80244, 1977.

[11] Greenbaum, A. u.a. — The Incomplete Cholesky Conjugate Gradient Method for the STAR. Lawrence Liv. Lab., UCRL-17574, 1977.

[12] Madsen, N.K. u.a. — A Comparison of Direct Methods for Tridiagonal Systems on the CDC-STAR-100. Lawrence Liv. Lab., UCRL-76993, 1977.

[13] Rodrigue, G.H. — Operator Splitting on the STAR without Transposing. Lawrence Liv. Lab., UCRL-17515, 1977.

[14] Nolen, J.S. u.a. — Application of Vector Processors to the Solution of Finite Difference Equations. Soc. Petr. Eng. AIME 1979.

CRAY 1/1S Software and Hardware

[15] The CRAY-1S Series of Computers. Pub. 2240008, CRAY Research, Minneapolis 1979.

[16] CRAY-1 FORTRAN (CFT) Reference Manual. Pub. 2240009, CRAY Research, Minneapolis 1979.

[17] CRAY-1 Hardware Reference Manual. Pub. 2240004, CRAY Research, Minneapolis 1979.

[18]	Engeln-Müllges, G.	Systemuntersuchung zur Vektormaschine CRAY-1. Bericht des Rechenzentrums der RWTH Aachen, August 1980.
[19]	Hertweck, F. u.a.	Benchmark-Versuche mit der CRAY-1. IPP-Bericht R/31, Max-Planck-Institut für Plasmaphysik, Garching 1979.
[20]	Rudsinsky, L. u.a.	Evaluating Computer Program Performance on the CRAY-1. Argonne Nat. Lab., Illinois 1979.
[21]	Russel, R.M.	The CRAY-1 Computer System. Comm. of the ACM 1978.
[22]	Johnson, P.M.	An Introduction to Vector Processing. Computer Design (1978), pp. 89-97.
[23]	Higbie, L.	Applications of Vector Processing. Computer Design, April 1978.
[24]	Higbie, L.	Vectorization and Conversion of FORTRAN Programs for the CRAY-1 (CFT) Compiler. Pub. 2240207, CRAY Research, Minneapolis 1979.
[25]	Petersen, W.P.	Basic Linear Algebraic Subprograms for CFT Usage. Pub. 2240208, CRAY Research, Minneapolis 1979.
[26]	Buzbee, B. Golub, G. Howell, J.	Vectorization for the CRAY-1 of some methods for solving elliptic difference equations, High Speed Computer and Algorithm Organization, Ed. D. Kuck, D. Lawrie and A. Sameh, pp. 255-272. New York: Academic Press, 1977.

DAP Software and Hardware

[27]		DAP: FORTRAN LANGUAGE. ICL Technical Publication 6918. 2. Ed. 1980.
[28]		DAP: Introduction to FORTRAN Programming. ICL, TP 6755.
[29]		DAP: Developing DAP Programs. ICL, TP 6920.
[30]		DAP: APAL (Assembly) Language. ICL, TP 6919.
[31]	Hunt, D.J.	Numerical Solution of Poisson's Equation on an Array Processor using Iterative Techniques. ICL, TR 1.
[32]	Hunt, D.J.	Application Techniques for Parallel Hardware. ICL, Technical Report, 1979.
[33]	Reddaway, S.F.	The DAP Approach. ICL, Technical Report, 1979.
[34]	Gostrick, R.W.	Software and Algorithms for the DAP. ICL, Techn. J. Vol. 1 (1979), issue 2.
[35]	Gostrick, R.W.	Supercomputer Diagnostics. Supercomputer Infotech State of the Art Report. Infotech Intl. Ltd., Maidenhead Berks 1979

[36]		DAP Newsletter. Queen Mary College, Computer Centre, DAP Support Unit.
[37]	Erhard, W.	Einführung in die Programmierung des DAP. Arbeitsmaterial. Universität Erlangen-Nürnberg 1981.
[38]	Parkinson, D.	An Introduction to Array Processors. Systems Int., Nov. 1977.
[39]	Parkinson, D. Flanders, P.M.	Fourier Transformation on DAP. ICL, TR 7, 1981.
[40]	Parkinson, D.	The Solution of Sets of Equations. ICL, TE 18, 1981.
[41]	Hunt, D.J.	A Study of Finite Element Analysis on DAP. ICL, TR 2, 1981.
[42]	Reddaway, S.F. Hunt, D.J.	Study of a Finite Element Problem. ICL, TR 10, 1981.

Vector and Parallel Computers and Algorithms (general)

[43]	Bucher, I.Y. u.a.	Comparative Performance Evaluation of two Supercomputers: CDC CYBER-205 and CRI CRAY-1. Los Alamos Scientific Lab., 1981.
[44]	Gentzsch, W. Müller-Wichards, D. Weiland, C.	Möglichkeiten und Probleme bei der Anwendung von Vektorrechnern, dargestellt an der numerischen Behandlung einiger strömungsphysikalischer Fragestellungen. Erfahrungen und Testrechnungen mit den Vektorrechnern von CDC und CRAY. DFVLR-IB 221 81 A 05, Göttingen 1981.
[45]	Hossfeld, F.	Parallelprozessoren und Algorithmenstruktur. Bericht Nr. 87, KFA Jülich 1980.
[46]	Madsen, N.K. u.a.	Matrix Multiplication by Diagonals on a Vector/Parallel Processor. Inf. Proc. Letters 2, Vol. 5, 1976.
[47]	Ramamoorthy, C.V. u.a.	Pipeline Architecture. ACM Computing Surveys (1977), pp. 61-102.
[48]	Heller, D. u.a.	Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers. JACM Vol. 23 (1976) No. 4, pp. 30-41.
[49]	Banerjee, U. Gajski, D. Kuck, D.	Array Machine Control Units for Loops Containing IFs, Proc. of the 1980 Int'l. Conf. on Parallel Processing, Harbor Springs, MI., Aug. 1980, pp. 28-36.
[50]	Baudet, G.M. Brent, R.P. Kung, H.T.	Parallel execution of a sequence of tasks on an asynchronous multiprocessor. The Australian Computer Journal Vol. 12 (1980), pp. 105-112.
[51]	Newman, I.A.	The use of parallel processing systems, CREST Course, Prallel Processing Systems, Loughborough 1980.

[52] Perrott, R.H. A Language for Array and Vector Processors, ACM Trans. on Programming Langs. and Systs. Vol. 1 (1979) No. 2, pp. 177-195.

[53] Robinson, J.T. Some Analysis Techniques for Asynchronous Multiprocessor Algorithms, IEEE Transactions on Software Engineering SE-5(1): 24-31, January 1979.

[54] Chen, S. Kuck, D. Time and parallel processor bounds for linear recurrence systems, IEEE Trans. Comp. C-24, 701-717, 1975.

[55] Dixon, L.C.W. Solution of Navier Stokes equations using finite elements, an optimisation approach using parallel computation, presented at EEC/CNR School, 'Design of Numerical Algorithms for Parallel Processing', Bergamo University, 1981.

[56] Dixon, L.C.W. Patel, K. The place of parallel computation in Numerical Optimisation II, The global problem, Proceedings of the E.E.C./C.N.R. Summer School "Design of Numerical Algorithms for Parallel Processing", Bergamo University, 1981.

[57] Dunbar, J. Analysis and Design of Parallel Algorithms, Ph. D. Thesis, Loughborough University of Technology, 1978.

[58] Enslow, P.H. ed. Multiprocessors and Parallel Processing, Wiley-Interscience, NY, 1974.

[59] Evans, D.J. Hadjidimos, A. Noutsos, D. The Parallel Solution of Banded Linear Equations by the New Quadrant Interlocking Factorisation (Q.I.F.) Method, Int. Jour. Comp. Math. 9, (1981), 151-162.

[60] Flynn, M.J. Very high-speed computing systems, Proceedings of the IEEE, Vol. 54, No. 12, December 1956, pp. 1901-1909.

[61] Heller, D. A survey of parallel algorithms in numerical linear algebra, SIAM Review 20, 740-777, 1978.

[62] Händler, W. Conpar 81, Conf. on Analysing Problem Classes and Programming for Parallel Computing. Berlin: Springer Verlag, 1981.

[63] The Vectran Language: An Experimental Language for Vector/Matrix Array Processing, G. Paul, M.W. Wilson (eds.). IBM Palo Alto Scientific Center, Report G320-3334, Aug. 1975, 343 pp.

[64] Kuck, D.J. The Structure of Computers and Computations, Vol. 1. New York: John Wiley & Sons, 1978.

[65] Kuck, D.J. Kuhn, R.H. Leasure, B. Wolfe, M. The Structure of an Advanced Vectorizer for Pipelined Processors. Proc. of COMPSAC 80, The 4th Int'l. Computer Software & Applications Conf., Chicago, IL, pp. 709-715, Oct. 1980.

[66] Kuck, D.J. Automatic Program Restructuring for High-Speed Computation. Proc. of CONPAR 81, Conf. on Analysing Problem-Classes and Programming for Parallel Computing, Nürnberg, F.R. Germany, Invited paper, June 1981.

[67] Kung, H.T. — The structure of parallel algortihms. In advances in Computers, Vol. 19. New York, Academic-Press, 1979.

[68] Miranker, W.L. — A Survey of Parallelism in Numerical Analysis. SIAM Review 13 (1971), pp. 524-547.

[69] Miranker, W.L. — Parallel Methods for Solving Equations. Technical Report RC 6545, IBM T.J. Watson Research Center, May 1977.

[70] Kozdrowicki, E.W. u.a. — Second Generation of Vector Supercompu-ters. Computer 7 (1980), pp. 71-83.

[71] Brandt, K. u.a. — Parallelrechner, Strukturen und Erfahrun-gen. RWTH Aachen 1978.

[72] Lambiotte, J. — The Solution of linear Systems of Equa-tions on a Vector Computer. Ph. D. Diss. Univ. Virginia 1975.

[73] Jordan, T.L. — A new Parallel Algorithm for Diagonally Dominant Tridiagonal Matrices. Los Alamos Scientific Lab. Report 1974.

[74] Kuck, D.J. — A Survey of Parallel Machine Organization and Programming. ACM Computing Surveys, pp. 29-60, 1977.

[75] Feilmeier, M. — Parallele Datenverarbeitung und Parallele Algorithmen. TU Berlin 1979.

[76] South, J.C. Keller, J.D. — Vector Processor Algorithms for Transonic Flow Calculations. AIAA Journal 18 (1980), pp. 786-792.

[77] Giloi, K. — Rechnerarchitekturen. Vorlesungsskript der TU Berlin, WS 1979/80.

[78] Sameh, A. — Numerical parallel algorithms - A survey, in High Speed Computer and Algorithm Organization, 207-228. D. Kuck, D. Lawrie and A. Sameh (eds.). New York: Academic Press, 1977.

[79] Sameh, A. Kuck, D. — On stable parallel linear system solvers. J. ACM 25 (1978), pp. 81-91.

[80] Smith, B.J. — A Pipelined, Shared Resource MIMD Compu-ter. Proc. of the 1978 Int'l Conf. on Parallel Processing, pp. 6-8, Aug. 1978.

[81] Stone, H.S. — Parallel computers. In: Introduction to Computer Architecture, pp. 318-374. Chicago: Science Research Associated, 1975.

Literature on the MHD Problem

[82] Bateman, G., Schneider, W., Grossmann, W. — Ideal MHD-Instabilities as an initial boundary-value problem. IPP-Bericht 1/145, Garching 1974.

[83] Schlüter, A. — Verfahren zur Behandlung stabiler Magnetohydrodynamischer Gleichgewichte. Sitzungsberichte der Bayerischen Akademie der Wissenschaften 1975.

[84] Gentzsch, W. — Numerical Solution of Linear and Non-Linear Parabolic Differential Equations by a Time-Discretisation of Third Order Accuracy. Notes on Numerical Fluid Mechanics Vol. 2 (1980), pp. 109-117.

Reports of the Hanover Regional Computer Centre on this Topic

[85] Materialsammlung zum 2. Kolloquium "Neue Rechnerarchitekturen: Anwendungsgebiete und Realisierungen" (CDC), Nr. 18, 1981.

[86] Vorträge zur Parallelverarbeitung, August, Nr. 19, 1981.

[87] Materialsammlung zum 3. Kolloquium "Neue Rechnerarchitekturen: Anwendungsgebiete und Realisierungen" (CRAY), Nr. 22, 1981.

[88] Materialsammlung zum 4. Kolloquium "Neue Rechnerarchitekturen: Anwendungsgebiete und Realisierungen" (ICL), Nr. 23, 1981.

Very Recent Publications

[89] Händler, W. — Thesen und Anmerkungen zur künftigen Rechner-Entwicklung. In: GMD-Rechnerstruktur-Workshop. Bericht der GMD Nr. 128. Regenspurg, G. (edit.). München, Wien: R. Oldenbourg, 1980.

[90] Engeln-Müllges, G., Sommer, A. — Parallel-Rechner, Parallele Algorithmen, Parallele Programmierung. Bibliographie. Bericht des Rechenzentrums der RWTH Aachen, 1980.

[91] Hossfeld, F. — Parallele Algorithmen. Bericht der KFA Jülich, 1981.

[92] Book, D.L. — Finite-Difference Techniques for Vectorized Fluid Dynamics Calculations. New York: Springer Verlag, 1981.

[93] Hockney, R.W., Jesshope, C.R. — Parallel Computers. Bristol: Adam Hilger Ltd., 1981.