University of Glasgow

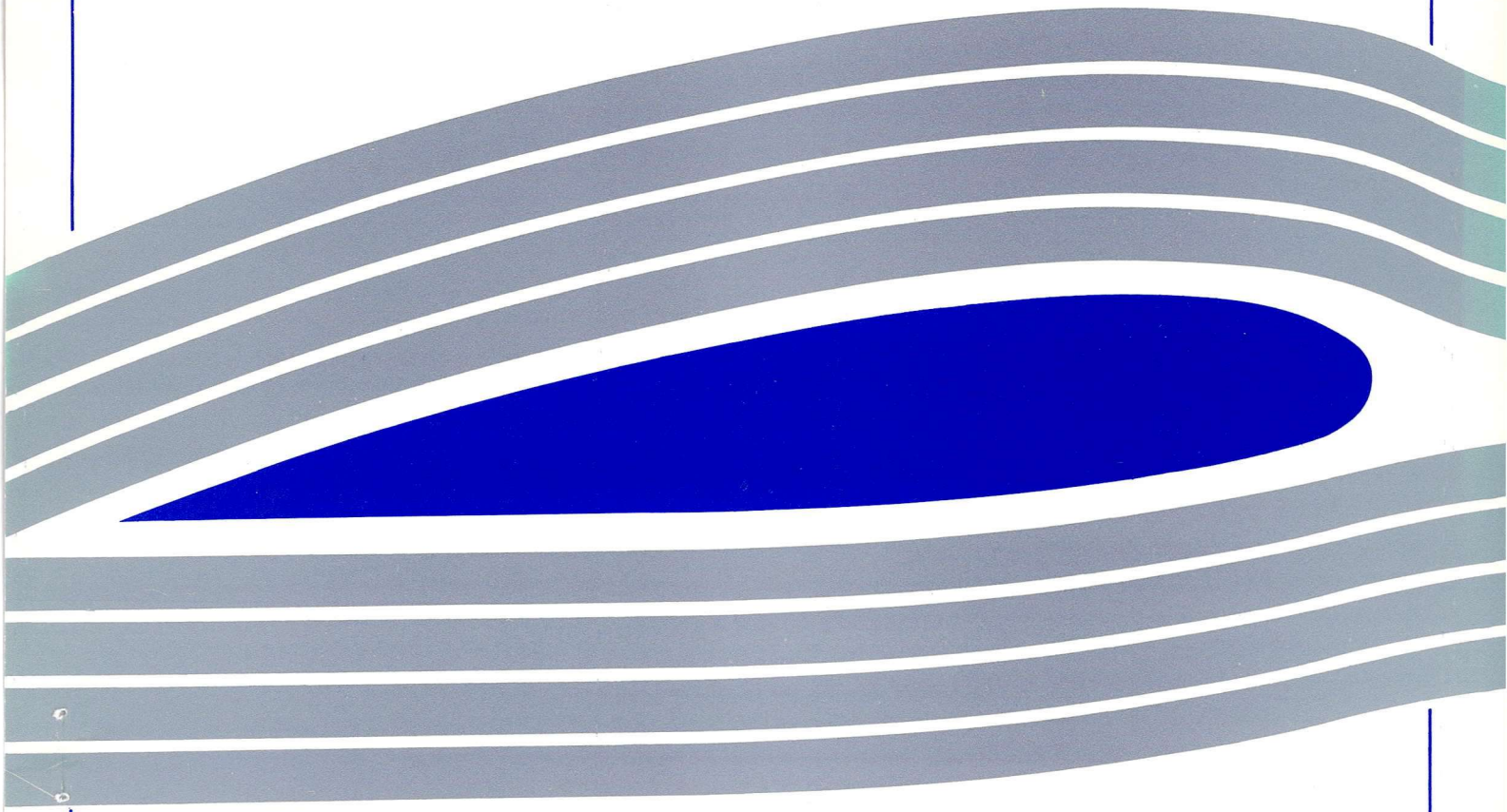
DEPARTMENT OF

AEROSPACE ENGINEERING

Porting parallel codes to various parallel environments

X. Xu

GU Aero Report 9417, October, 1994



Porting parallel codes to various parallel environments

X. Xu

GU Aero Report 9417, October, 1994

Department of Aerospace Engineering
University of Glasgow
Glasgow G12 8QQ

Porting Parallel Codes to Various Parallel Environments

Xiao Xu [1]

Abstract. Some existing parallel N-S codes, including both explicit and implicit algorithms, have been ported on different parallel environments. The application of the PVM software package to a cluster of workstations and on the EPCC Cray-T3D have also been illustrated.

1 INTRODUCTION

In the computational fluid dynamics (CFD) area, in order to simulate complex three dimensional flows, numerical solutions of the Navier-Stokes (N-S) equations need much CPU time and memory. One state of the art technique is extending original sequential codes to operate in parallel. Some parallel codes have been developed for solving both algebraic non-linear systems and two and three dimensional high speed viscous flows (see [6],[7],[8]). Algorithms presented in the above papers achieved high parallel efficiency and could decompose memory requirements to each processor efficiently.

However, the way that hardware and software of parallel environments develop will be a key factor in the emergence of CFD codes. Some message passing parallel computers available are the Meiko Computing Surface, Intel iPSC/860 Hypercube, and Cray-T3D. Versions of the latter two belong to the Engineering and Physical Science Research Council (EPSRC) in Daresbury Laboratory and the Edinburgh University Parallel Computing Centre respectively. Since the Meiko Computing Surface in the University of Glasgow was composed of nearly 40 of the slower T800 transputers, it was used only for testing algorithms. Concerning software, the PVM package is a successful message passing software available in the public domain, which allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. By using PVM, the cluster of workstations can be used as a parallel computer. However, the speed of message passing on the network could be critically dependent on the application.

[1] Department of Aerospace Engineering, University of Glasgow, Glasgow, G12 8QQ, UK

2 NUMERICAL METHODS

2.1 Discretised equations

Using a numerical method to solve a system of differential equations often includes the discretizations of the equations and the domain where the equations are defined. Discretization of the domain can be carried out by a structured or unstructured grid. After spatial discretization a semi-discretised system of ordinary differential equations in time can be defined as

$$\frac{\partial V}{\partial t} + R(V)=0 \tag{1}$$

where $R(V)$ is a N-dimensional non-linear *algebraic* system in the global domain, and N is the number of all unknown variables on discretised cells (nodes).

2.2 Linearisations

For Eq. (1) various linearization methods are available to transfer the original problem to an iterative procedure. Using the explicit Euler method we have

$$V^{k+1} = V^k - \Delta t R(V^k) \tag{2}$$

A four-step Runge-Kutta method is

$$\begin{aligned} V_1 &= V^k - \alpha_1 \Delta t R(V^k) \\ V_2 &= V^k - \alpha_2 \Delta t R(V_1) \\ V_3 &= V^k - \alpha_3 \Delta t R(V_2) \\ V^{k+1} &= V^k - \Delta t R(V_3) \end{aligned} \tag{3}$$

where α_1=1/4, α_2=1/3, α_3=1/2. Using a fully implicit method, e.g., the backward Euler implicit method,

$$\begin{aligned} (\tfrac{1}{\Delta t} + (\tfrac{\partial R(V)}{\partial V})^k)\Delta^k V &= -R(V^k) \\ V^{k+1} &= V^k + \Delta^k V \end{aligned} \tag{4}$$

As the time step approaches infinity the method approaches the Newton's method

$$\begin{aligned} (\tfrac{\partial R(V)}{\partial V})^k \Delta^k V &= -R(V^k) \\ V^{k+1} &= V^k + \Delta^k V \end{aligned} \tag{5}$$

Therefore, generally speaking, in each step of the iterative procedures the following linear system is included

$$J^k \Delta^k V = -R(V^k) \qquad (6)$$

where J^k is a diagonal matrix for the explicit linearization method, for which the solution of the linear system can be obtained directly, and a non-diagonal matrix for the implicit linearization method. Therefore, the computation falls into three phases, i.e. the generation of the right hand side; the derivation of the left hand side Jacobian matrix; and the solution of the linear system. For the explicit method the latter two phases are not needed.

In this paper the Newton's method is used for solving the 2-dimensional and pseudo 3-dimensional high speed viscous flows and the explicit method is used for the 3-dimensional flows. A cell centred finite volume method and the Osher's upwind scheme with the high order MUSCL interpolation are used to discretise the N-S equations.

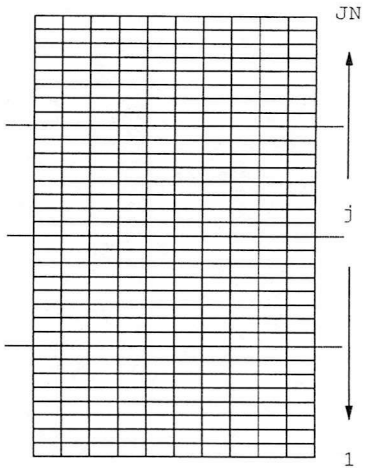


Figure 1. Domain decomposition

3 PARALLEL ALGORITHMS

3.1 Domain decomposition

The benefits of using the structured grid are: (1) it is easy for ordering the cells, (2) it makes the Jacobian matrix of the implicit method structured, (3) it is more suitable for the discretization of boundary layers, and (4) in the parallel calculation it is easy for using the domain decomposition technique.

For the structured grid (i,j) of the 2-dimensional or (i,j,k) of the 3-dimensional domain, one direction or multidirections decomposition can be employed. The order is first to decompose the k direction and then, if it is necessary, to decompose the j direction and finally the i direction. Each array used in the code has the index of (i,j,k) or (l,i,j,k), where l represents another index for grouping a set of relative arrays. An example of the 2-dimensional structured grid with one direction decomposition is illustrated in figure 1.

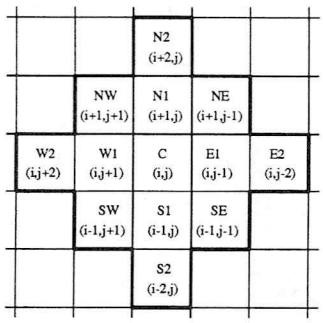


Figure 2. 13 cell stencil

3.2 Parallelizability

It has been presented in [7] and [8] that in the former two operations of the three phases of calculations, the generation of one component only concerns the cell, to which the component is corresponding, and its limited neighbouring cells (e.g. figure 2). This type of calculation is said to have local characteristics. The other approach is said to have global characteristics, an example being the solution of the linear system.

Since the variables on neighbouring cells are needed for the local characteristic, communications are required between the neighbouring subdomains for the message passing near the boundary of them. The extent of cells, for which the variables are needed to be transferred, is dependent on the extent of the local characteristic. Figure 3 shows the real subdomains needed for execution of the parallel code.

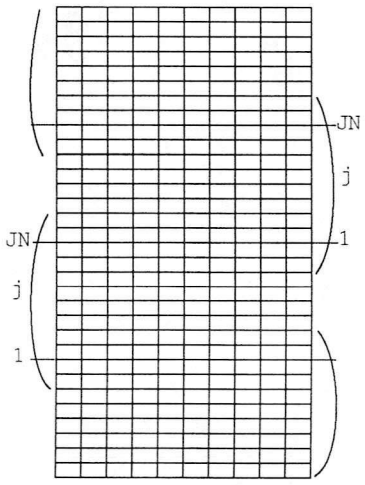


Figure 3. Subdomains

Using the GMRES or CGS linear solver vector inner products and matrix-vector multiplications are included. Assuming there are M processors available. One of the matrix storage methods is through decomposing the matrix along column with no data over-lap. Assuming the matrix A is of order N ($N > M$), the matrix A can be written in columns as $A=[A^1,A^2,...,A^M]$, where A^m is a $N * L$ submatrix. The transposition of vectors x and b can be written as $x^T=[x_1^T,x_2^T,...,x_M^T]$ and $b^T=[b_1^T,b_2^T,...,b_M^T]$ where x_m^T and b_m^T are vectors of order L corresponding to A^m,m=1,2,...M. A^m, x_m^T and b_m^T are stored in each processor m. The calculation of the inner product of two *algebraic* vectors a and b is equal to the sum, within all processors, of the inner products of their corresponding components (eq 7). Therefore the calculation includes single processor and multiprocessor accumulation and broadcast procedures [1].

$$(a,b)=\sum_{i=1}^{n} a_i b_i=\sum_{p1} a_i b_i + ... + \sum_{pM} a_i b_i \qquad (7)$$

The task of calculating Ax to M processors is divided by calculating $A^m x_m$ on processor m. Let y=Ax, i.e.

$$\begin{pmatrix} y^1 \\ y^2 \\ . \\ . \\ y^M \end{pmatrix} = \begin{pmatrix} A^1,A^2,...,A^M \end{pmatrix} \begin{pmatrix} x^1 \\ x^2 \\ . \\ . \\ x^M \end{pmatrix},$$

we have

$$Ax=\begin{pmatrix} A^1,A^2,...,A^M \end{pmatrix} \left(\begin{pmatrix} x^1 \\ 0^2 \\ . \\ . \\ 0^M \end{pmatrix} + \begin{pmatrix} 0^1 \\ x^2 \\ . \\ . \\ 0^M \end{pmatrix} + ... + \begin{pmatrix} 0^1 \\ 0^2 \\ . \\ . \\ x^M \end{pmatrix} \right)$$

$$= A^1x^1 + A^2x^2 + ... + A^Mx^M$$

$$= \begin{pmatrix} * \\ * \\ . \\ . \\ * \end{pmatrix} + \begin{pmatrix} * \\ * \\ . \\ . \\ * \end{pmatrix} + ... + \begin{pmatrix} * \\ * \\ . \\ . \\ * \end{pmatrix}$$

$$=> \begin{pmatrix} y^1 \\ 0^2 \\ . \\ . \\ 0^M \end{pmatrix} + \begin{pmatrix} 0^1 \\ y^2 \\ . \\ . \\ 0^M \end{pmatrix} + ... + \begin{pmatrix} 0^1 \\ 0^2 \\ . \\ . \\ y^M \end{pmatrix}$$

The resulting vector y is again distributed to the M processors. The only communication required in the calculation is in the formation of y. Due to the sparsity of the matrix A, this communication is only of a limited nature.

In using GMRES or CGS linear solver there is no sequential bottle-neck for the parallelization, even if the block diagonal or 5 diagonal blocks incomplete lower and upper factorisation (5BILUF) preconditioner is used [8]. From [8] a BILUF or 5BILUF preconditioner is very efficient only if the factorisation can keep the important information of the Jacobian matrix, and the sequential bottle-neck is present when using BILUF preconditioner, such as BILUF generation and forward-backward substitution. However, the cost of computing these terms has been reduced to only a few percent of the total computation cost. That is, the calculation, which is not parallelizable, will have little effect on the efficiency of the parallelization of the complete Newton's method.

α-GMRES is another fully parallelizable algorithm for solving linear systems, which arise from the representation of very complicated physical phenomenon [6].

3.3 Parallel implementation

There are two types of message passing, i.e. between the neighbouring processors and between all processors.

In one calculation the whole message passing between the neighbouring subdomains along one coordinate direction can be completed within four steps of communications as in figure 4.

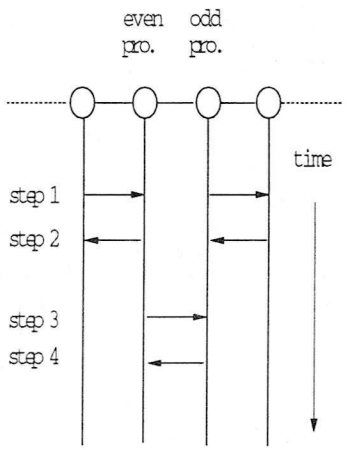


Figure 4. Parallel flow chart

4 PARALLEL CODES

The enrollment of parallel on iPSC860 is as following:

```
c   ------------------
c   Enroll in parallel
c   ------------------
    nun=numnodes()
    if(nun.ne.npro) stop 999
    myn=mynode()
c   ------------------------------
c   Determine neighbors in the ring
c   ------------------------------
```

```
      lan=myn-1
      nen=myn+1
      if(lan.lt.0) lan=lan+npro
      if(nen.gt.npro-1) nen=nen-npro
```

where the decomposition is along one direction k, *npro* is the number of processors, which are used in the calculation and set by the user. *lan*, *myn*, *nen* are the identification numbers of my previous processor, myself, and my next processor, respectively.

Message passing is executed by

```
call csend(type,buf,len,node,pid)
```

and

```
call crecv(type,buf,len)
```

where type is an integer that identifies the type of the message to be sent or received and can be set by user, buf refers to the buffer that contains the message to be sent or received, len is the size of bytes of the message, node is an integer that defines where the message is to be sent, and pid must be set to 0.

The enrollment of PVM 3.3 on Cray-T3D is as following:

```
c     -------------
c     Enroll in pvm
c     -------------
      call pvmfmytid( mytid )
c     --------------
c     T3D inclusions
c     --------------
      call pvmfgetpe(mytid, pe)
      call pvmfgettid(pvmall,0,mtid)
      call pvmfgsize(pvmall,nun)
c     ------------------------------
c     Determine neighbors in the ring
c     ------------------------------
      if(nun.ne.npro) stop 999
      myn=mytid-mtid
      mynj=myn+1-(myn/nproj)*nproj
      mynk=myn/nproj+1
      lank=(mynk-2)*nproj+mynj-1
      lanj=(mynk-1)*nproj+mynj-2
      nenj=(mynk-1)*nproj+mynj
      nenk=mynk*nproj+mynj-1
      mynj=mynj-1
      mynk=mynk-1
```

The enrollment of PVM 3.3 on workstation clusters is as following:

```
c     -------------
c     Enroll in pvm
c     -------------
      call pvmfmytid( mytid )
      name='nscode'
      call pvmfparent( tids(0) )
      if( tids(0) .lt. 0 ) then
        tids(0)=mytid
        myn=0
        call pvmfspawn(name,PVMDEFAULT
     &       ,'*',npro-1,tids(1),info)
        if(info.lt.npro-1) stop 999
        do i=2,npro
         type=50000
         call pvmfpsend(tids(i-1),type
     &     ,tids(0),npro,INTEGER4,info)
        end do
       else
        type=50000
        call pvmfprecv(-1,type,tids(0)
     & ,npro,INTEGER4,ati,ata,len,info)
        do i=1,npro-1
         if(mytid.eq.tids(i)) myn=i
        end do
       endif
c     ------------------------------
c     Determine neighbors in the ring
c     ------------------------------
      mynj=myn+1-(myn/nproj)*nproj
      mynk=myn/nproj+1
      lank=(mynk-2)*nproj+mynj-1
      lanj=(mynk-1)*nproj+mynj-2
      nenj=(mynk-1)*nproj+mynj
      nenk=mynk*nproj+mynj-1
      mynj=mynj-1
      mynk=mynk-1
      latj=tids(lanj)
      netj=tids(nenj)
      latk=tids(lank)
      netk=tids(nenk)
```

where the decomposition is along two directions k and j. *npro*, *nproj*, and *nprok* are the numbers of all processors, j direction processors, and k direction processors respectively, which are used in the calculation and set by the user. *myn* is the identification number of myself in all processors. *lanj*, *mynj*, *nenj* are the identification numbers of my previous processor, myself, and my next processor in j direction, respectively. *lank*, *mynk*, *nenk* are the identification numbers of my previous processor, myself, and my next processor in k direction, respectively. *latj* and *nenj* are the identification numbers of my previous processor and my next processor in j direction, and *latk* and *netk* are the identification numbers of my previous processor and my next processor in k direction, which are used in workstation clusters version.

Message passing is executed by

```
call pvmfinitsend(PVMDEFAULT,info)
call pvmfpack(what,buf,len,stride,info)
call pvmfsend(node,type,info )
```

and

```
call pvmfrecv(node,type,info )
call pvmfunpack(what,buf,len,stride,info)
```

where type is an integer that identifies the type of the message to be sent or received and can be set by user, buf refers to the buffer that contains the message to be sent or received, len is the size of words of the message, node is an integer that defines where the message is to be sent or received, what is equal to REAL4 or REAL8 for the real data to be sent or received (on Cray-T3D, always REAL8), and stride is an integer and will equal 1 for real data to be packed or unpacked.

Note: On Cray-T3D, the node number for the neighbouring nodes is lanj, nenj, lank, or nenk, however on workstation clusters, it should be latj, netj, latk, or netk which is obtained from a calculation, such as *latk=tids(lank)*.

References [3] and [2] also provide useful information.

5 NUMERICAL RESULTS

The foregoing numerical tests have been carried out to solve the LCNS, PNS and 3-d NS equations for compressible flow. One case is a laminar Mach 7.95 flow around a sharp cone at an angle of attack of 24°. Another case is the flow around an ogive cylinder at Mach 2.5 and an angle of attack of 14°. Any one of the test cases produces a flow which has a large separated flow region with an embedded shock wave in the leeward side of the object and strong gradients in the thin boundary layer on the windward side, more details of the physical phenomena involved in these flows can be found in [5] and [4]. The computers used are a cluster of Silicon Graphics workstations, Cray-T3D, and Intel iPSC/860 Hypercube.

Parallel implementations for various procedures in using implicit method in LCNS code have been shown in table 1, where the dimension of the Krylov subspace is 50, and the grid size is 34×66. In the table the first GMRES column is that in conjunction with 5BILU preconditioner and the second is in conjunction with the BILU preconditioner.

Table 1. CPU time for one step implementation

◇	□	△	5BILU	GMRES	BILU	GMRES
1	7.17	39.27	1.23	45.51	—	—
2	3.71	19.23	0.62	23.50	2.03	37.58
4	1.95	9.86	0.31	12.45	1.02	30.04
8	1.04	4.95	0.15	6.61	0.62	27.13
16	0.57	2.35	0.08	4.20	0.50	26.41

◇: Number of processors □: Explicit iteration
△: Generation of Jacobian matrix

Parallel results of the 3-d NS code on the Cray-T3D, the iPSC/860, and the workstation clusters for the explicit method is illustrated in table 2. The grid is (i,j,k).

The CPU time (seconds) of one step iteration of a parallel implementation for explicit and implicit methods in the PNS code have been shown in table 3, where the grid size is 34×66 in one section. The explicit method is the Euler explicit method. The implicit method includes Newton's method, GMRES(30) linear solver, block diagonal precondition, and BILUF preconditioning.

Table 2. CPU time for one step explicit iteration

◇	Grid	□	Cray-T3D	iPSC/860	△
2	33x33x57	k	10.27 sec	—	3.88 sec
4	33x33x57	k	5.33 sec	—	2.05 sec
8	33x33x57	k	2.87 sec	—	1.04 sec
4	33x33x57	j,k	5.15 sec	13.64 sec	—
8	33x33x57	j,k	2.63 sec	7.60 sec	—
32	33x33x57	j,k	0.72 sec	1.89 sec	—
64	33x33x57	j,k	0.36 sec	—	—
64	65x65x113	j,k	2.76 sec	—	—
128	65x65x113	j,k	1.43 sec	—	—

◇: Number of processors □: Decomposition direction
△: Silicon Graphics cluster

Table 3. Comparison of using different environments

	Cray-T3D		iPSC/860		Clusters	
◇	□	△	□	△	□	△
2	0.249	15.93	0.815	43.2	0.116	10.4
4	0.114	10.23	0.425	29.5	0.071	6.0
8	0.063	8.12	0.228	22.9	0.036	3.1
16	0.034	6.12	0.164	20.3	0.022	1.9

◇: Number of processors □: Explicit iteration
△: Implicit iteration

6 CONCLUSION

We have shown some results of using the PVM package on workstation clusters and the EPCC Cray-T3D, and some results on the iPSC/860. The common parallel feature of these environments is message passing. The parallel implementations accelerate the calculations and divide the storage to individual processors.

There are two kinds of jobs can be submitted to the Cray-T3D, interactive and batch under NQS. The interactive job is only used for testing the code. When a job is submitted to the Cray-T3D by NQS, the user needs to wait for the necessary number of processors to become available, i.e. even if the calculation is very quick when using large numbers of processors people cannot have the result quick enough. The situation will worsen in future when the number of users increase. When using workstation clusters, the job can be started immediately, which will share the computer resources with other jobs. There are also big difference between the CPU time used and the real time for obtaining the result. Also another problem is that of load balance, i.e. some workstations are heavily used, which delays the whole calculation. It is apparent that the iPSC/860 has lost it power nowadays, since each processor is of relatively slow speed and the number of processors that can be made available in a calculation is limited.

REFERENCES

[1] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall International, Inc., 1989.

[2] Al. Geist et. al., *PVM 3 User's Guide and Reference Manual*, 1994.

[3] Intel Corporation, *iPSC/2 and iPSC/860 Programmer's Reference Manual*, 1991.

[4] X.S. Jin, 'A three dimensional parabolized Navier-Stokes equations solver with turbulence modelling', Technical report, GU Aero Report 9315, (1993).

[5] N. Qin, X. Xu, and B.E. Richards, 'SFDN-α-GMRES and SQN-α-GMRES methods for fast high resolution NS simulations', in *ICFD Conference on Numerical Methods for Fluid Dynamics*, ed., K.W. Morton ed. Oxford University Press, (1992).

[6] X. Xu, N. Qin, and B.E. Richards, 'α-GMRES: A new parallelizable iterative solver for large sparse non-symmetric linear system arising from CFD', *International Journal for Numerical Methods in Fluids*, **15**, 613–623, (1992).

[7] X. Xu, N. Qin, and B.E. Richards, 'Parallelising explicit and fully implicit Navier-Stokes solutions for compressible flows', in *Parallel Computational Fluid Dynamics '92*, ed., R.B. Pelz ed. Elsevier, (1993).

[8] X. Xu and B.E. Richards, 'Numerical advances in Newton solvers for the Navier-Stokes solutions', in *Computational Fluid Dynamics '94*, ed., S. Wagner ed. John Wiley and Sons, (1994).