INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 17, 1115–1133 (1993)

THE CALCULATION OF SEPARATED FLOWS USING A DISTRIBUTED MEMORY MIMD COMPUTER

N. HARDY, B. A. MURRAY, M. J. DOWNIE AND P. BETTESS

Department of Marine Technology, University of Newcastle upon Tyne, Queen Victoria Road, Newcastle upon Tyne, NE1 7RU, U.K.

SUMMARY

A brief description of distributed memory MIMD computers, and in particular, a transputer based computing surface, is presented. The factors to be considered in their application to computationally intensive CFD problems are discussed with reference to the discrete vortex method used for the solution of separated flow about bluff bodies. Three parallel algorithms for its implementation are presented, and the results are discussed in the context of the speed-ups obtained using up to sixty-four parallel processors.

KEY WORDS Parallel processing MIMD Discrete vortices Distributed memory Separated flow

1. INTRODUCTION

Over the past decade, the processing power of computers has increased dramatically, but at the same time, scientists and engineers are producing more data to be processed and larger problems to be solved. The conventional approach to higher performance is to increase the speed of sequential, or von Neumann, architectures. However, an ultimate limitation exists with the speed at which electrical signals can be transmitted over physical connections between processor components.

An alternative approach is to distribute data and algorithms over many processors and thus achieve parallelism. Parallel architectures are usually classified according to Flynn's taxonomy[1]—Multiple Instruction Single Data (MISD), Single Instruction Multiple Data (SIMD), or Multiple Instruction Multiple Data (MIMD).

Earlier parallel computers were SIMD machines, such as the ICL Distributed Arrary Processor (DAP), which simultaneously executes the same instruction on separate data items. An example of a MISD architecture is a pipeline system, where processors execute different instructions on a single stream of data. A MIMD machine is characterized by independent processors executing different programs using different data and communicating with each other. There is now a wide range of MIMD machines which utilize different processors and communication architectures. The field is too large to investigate here and the reader is referred to Reference 2 for a comprehensive survey. A transputer based architecture is an example of a distributed memory MIMD machine and is described in Section 3.

The advent of such revolutionary computer architectures has significant implications for the solution of fluid flow problems. Whilst the Navier–Stokes equations provide a complete description of fluid flows of engineering interest, they can be solved analytically for a very limited range of geometries and boundary conditions. However, considerable progress has been made, where

0271–2091/93/241115–19$14.50

Received October 1992
Revised August 1993

the equations can be simplified, and the solution to many problems in which it is permissible to ignore viscous effects have been obtained. Solution of potential flow problems by grid methods, such as that of Clarke *et al.*,[3] or using boundary integral formulations, such as Hearn *et al.*,[4] are good examples of this. The parallelization of a boundary integral calculation is the subject of another publication.[5]

The use of grid methods for the solution of problems where the viscous terms have to be retained, for the separated flow about bluff bodies for example, have been less successful in engineering applications. The discrete vortex method was originally developed in response to this deficiency. Further progress has been made with finite difference and finite element models and many models that have evolved from the original discrete vortex method now make use of grids. The two approaches are now converging with the former extending their range of applicability, and the latter including viscous diffusion effects.

The feature that all numerical models used for this class of flows have in common is that they are highly computationally intensive. In the case of grid methods this arises from the requirement of appropriately fine meshes required to capture the details of flows with extremely high-velocity gradients. This compounds the problem of providing sufficient computer storage and processing time to carry out the computation of time-dependent flows over a long enough period to be meaningful. These problems can be ameliorated by MIMD computers, which are capable of achieving large reductions in elapsed processing time through the employment of parallel algorithms.

2. THE DISCRETE VORTEX MODEL

The flow about bluff bodies can be characterized by the Reynolds number and, if it is also oscillatory as in wave flows, by the Keulegan–Carpenter number. The Reynolds number, $Re = D\hat{U}/\nu$, is a measure of the relative importance of the inertial and the viscous effects in a flow of typical velocity, $\hat{U}$ and kinematic viscosity, ν, about a body of typical dimension, D. At low Reynolds numbers viscous effects, such as the diffusion of vorticity generated at the fluid/body interface, are more important. At high Reynolds numbers boundary layers are thin and are important chiefly as generators of vorticity which is transported through a flow in which high-velocity gradients are common and in which convection dominates. Also, as the Reynolds number increases the presence of turbulence in the flow becomes increasingly significant, eventually precipitating transition to turbulence in the boundary layers and inducing a drag crisis. The application of turbulence models appropriate to unsteady separated flows is, as yet, far from being common place. Throughout the higher Reynolds number range, except through the drag crisis, the vorticity is largely confined to concentrated subregions of an otherwise irrotational flow, the energy spectrum of the whole being narrow banded.

In a unidirectional flow the concentrated regions of vorticity arise from the roll up of the separated shear layers resulting in the formation of the familiar von Karman vortex street. In oscillatory flow the situation is more complicated. The Keulegan–Carpenter number, which characterizes such flows, is defined as $\hat{U}T/D$, where $\hat{U}$ is now the velocity amplitude of the undisturbed flow at the point occupied by the body. It is a measure of the magnitude of a typical fluid particle orbit in the undisturbed flow in relation to the scale of the body over the period, T. If the Keulegan–Carpenter number is very small (less than 3) separation barely occurs and there is no discernable vortex shedding. If it is large, a number of vortices are shed over each half cycle of the flow, and if it is large enough, the flow over each half cycle should resemble that behind a cylinder in a steady unidirectional stream. At intermediate Keulegan–Carpenter numbers a range of flow regimes occur, each associated with its own distinctive vortex shedding pattern.[6,7]

The discrete vortex method was developed for modelling high Reynolds number separated flow during the late sixties. Early proponents of the method applied to circular cylinders include Gerrard[8] and Sarpkaya[9] and to sharp edged bodies, Clements[10] (for a review of more recent developments, see Reference 11).

In the basic discrete vortex method, the Reynolds number is assumed to be sufficiently high for viscosity to play little part except in the initial generation of vorticity at the surface of the body. The vorticity is represented by point singularities in an otherwise irrotational flow. In other words, the right-hand side of the vorticity convection equation,

$$\frac{\partial \omega}{\partial t}+u\cdot\nabla\omega=\nu\nabla^2\omega, \tag{1}$$

is set to zero. The left-hand side is modelled by a Lagrangian approach in which the vorticity, defined as $\omega=\nabla\times u$, where u is the velocity, is convected with the fluid particles. Under these conditions the vorticity of each particle remains unchanged with time. Computation of the flow around a circular cylinder of radius a, follows a gridless time-stepping procedure.

The complex potential is defined in terms of a velocity potential, ϕ, and a stream function, ψ;

$$W=\phi+\mathrm{i}\psi. \tag{2}$$

At each time step this is given by

$$W(\zeta)=\left(\zeta-\frac{a^2}{\zeta}\right)U+\frac{\mathrm{i}}{2\pi}\sum_{j=1}^{N_v}\Gamma_j\left[\ln(\zeta-\zeta_j)-\ln\left(\zeta-\frac{a^2}{\bar{\zeta}_j}\right)\right], \tag{3}$$

where ln is the natural logarithm. The first term represents the unseparated irrotational flow of a uniform stream about a circular cylinder. The second term represents an array of N_v discrete vortices with position vectors, ζ_j, and strengths or circulations, Γ_j, modelling the vorticity in the flow. The third term, in which the over bar represents the complex conjugate, is made up of the vortex images in the cylinder required to maintain the normal boundary condition, $\partial\phi/\partial n=0$, on the surface of the body, where n indicates the normal.

The discrete vortices are introduced into the flow at each time step, so as to satisfy appropriate boundary conditions at the cylinder surface. The velocity of each discrete vortex is calculated using the Biot–Savart law and the complex potential

$$\frac{\partial\zeta_j}{\partial t}=(u-\mathrm{i}v)_j=\frac{\mathrm{d}W}{\mathrm{d}\zeta_j} \tag{4}$$

and the position of each vortex is advanced during a small time interval, Δt, by a suitable time integration scheme, say,

$$\zeta_j(t+\Delta t)=\zeta_j(t)+\frac{\partial\zeta_j}{\partial t}\Delta t. \tag{5}$$

The complex force can be calculated at each time step using the generalized Blasius theorem

$$\bar{F}=\frac{1}{2}\mathrm{i}\rho\oint\left(\frac{\mathrm{d}W}{\mathrm{d}z}\right)^2\mathrm{d}z+\mathrm{i}\rho\frac{\partial}{\partial t}\oint\bar{W}\,\mathrm{d}\bar{z}, \tag{6}$$

where ρ is the fluid density.

For a steady-unidirectional flow, this may be simplified as

$$\bar{F}=-\mathrm{i}\rho\frac{\partial}{\partial t}\sum_j\left(\zeta_j-\frac{a^2}{\bar{\zeta}^2}\right)\Gamma_j,$$

using the complex potential (3). For an oscillatory flow the U term in equation (3) becomes time dependent and there is an additional inertia term present in the force equation due to the acceleration of the flow.

Repetition of the procedure yields a time history of the flow and the corresponding forces experienced by the cylinder. The most computationally intensive part of the algorithm is the calculation of the velocity of each vortex in the flow at every time step. As can be deduced from equations (3) and (4), this involves $O(N_v^2)$ calculations. The velocity of any given vortex is the sum of the contributions made by the ambient flow and by every other vortex in the flow. It should be noted that the contributions can all be computed independently, a feature that can be exploited by parallel algorithms.

Various sequential strategies have evolved to reduce the time taken to compute flows requiring large numbers of discrete vortices. An example is high Keulegan–Carpenter oscillatory flows in which a large number of time-steps is taken before the flow settles down to a representative shedding pattern. In the model used for the present studies, the well-known procedure of distributing circulation, Γ, of the discrete vortices onto the nodes of a grid using bilinear interpolation is adopted. The contributions from a vortex to the surrounding nodes can be expressed as a relevant ratio multiplied by the vortex circulation. When this is generalized over a distributed vortex flow field the nodal circulations are found by the equation

$$\Gamma_{j,k}=\sum_{v}\frac{A_q(v)}{A_T(v)}\Gamma_v,$$

where the vortices v are contained in an element with $z_{j,k}$ as one of its nodes, q signifies the relevant node number, $A_q(v)$ represents the subelement areas and $A_T(v)$ the total element area (see Figure 1). The velocity field is calculated at the nodes and the vortex velocities are found by a reverse interpolation procedure. This approach requires only $O(N_n \log_2 N_n)$ computations, where N_n is the number of active nodes, and many thousands of discrete vortices can be handled.

Another feature of the present model is that equation (1) is solved in its entirety using operator splitting, in which the vorticity equation is divided into a convective and a diffusive part. On each time step the convective part is solved in the manner just described and the diffusive part is represented by the equation

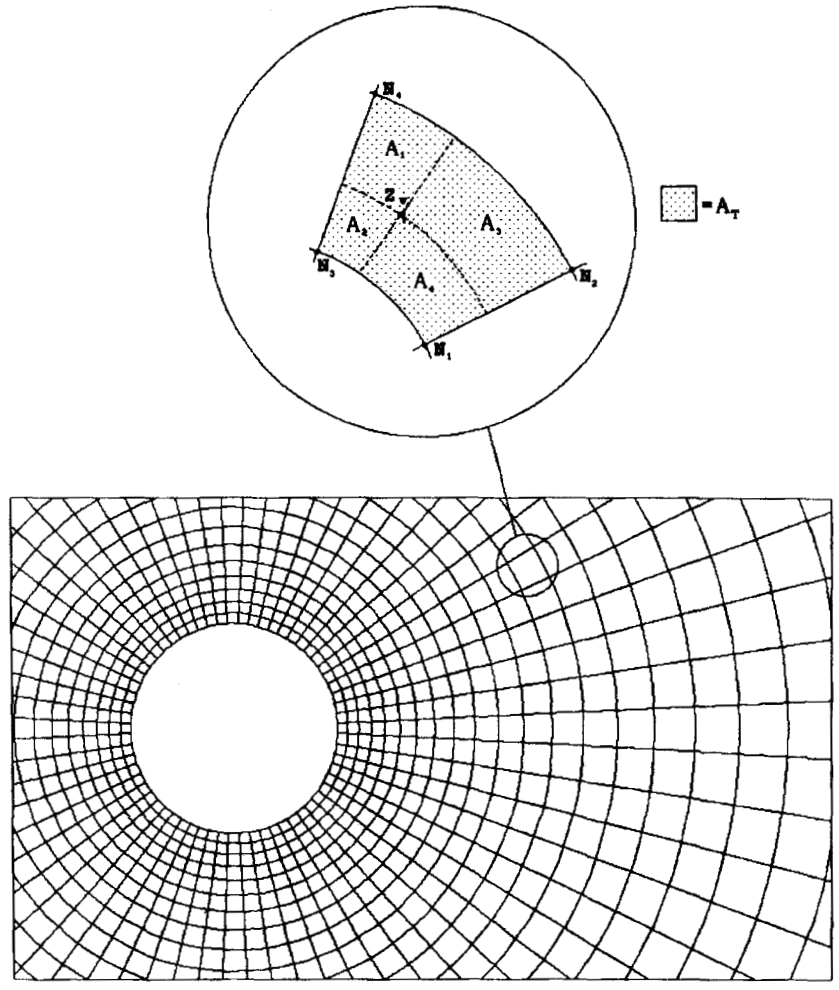
$$\frac{\partial\omega}{\partial t}=\nu\nabla^2\omega.$$

This is modelled stochastically and solved by a random walk approach, as used by Chorin[12] and Lewis and Porthouse,[13] in which each discrete vortex is given a two-dimensional displacement using two independent sets of Gaussian random numbers of zero mean and standard deviation, $\sqrt{(4\nu\,\Delta t/Re)}$. The inclusion of viscous diffusion in the model means that it may be applied over an intermediate Reynolds number range, where it may be more reasonable to ignore the effects of turbulence. However, it does increase the amount of computation at each time step, a factor that becomes more important still if the diffusion step is solved by a finite difference scheme.[14,15]

The following sections describe how the elapsed time of separated flow calculations can be reduced by implementing parallel algorithms on a distributed memory MIMD computer, in this case comprising an array of transputers.

3. PARALLEL HARDWARE AND SOFTWARE

A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another.[16] The concept was developed by the company INMOS. One such transputer, from the INMOS transputer family, is the IMS T800 which incorporates a 64-bit



$N_1 = z_{j,k}$, $N_2 = z_{j+1,k}$, $N_3 = z_{j,k+1}$, $N_4 = z_{j+1,k+1}$

Figure 1. Polar grid

floating point unit which can sustain 1·5 Million Floating Point Operations Per Second (MFLOPS).[17] The Central Processing Unit (CPU) is of a 32-bit architecture. The local memory consists of 4 Kbytes of on-chip RAM with an 80 Mbytes s^{-1} data transfer rate. A memory interface provides an external memory bandwidth of 26·6 Mbytes s^{-1}. Four serial communication links can transfer data from one transputer to another at a rate of 10/20 Mbits s^{-1}. Full specifications of the IMS T800 are given in the T800 product overview[18] (Figure 2).

Since the introduction of the transputer in 1985, many transputer based machines and plug-in boards for PCs have become commercially available. These support a plethora of operating systems, toolsets and programming languages. Operating systems for transputers fall into two categories—those designed specifically for the transputer, e.g. the Transputer Development System (TDS) and Helios, and proprietary systems which have been extended with extra facilities or toolsets, e.g. UNIX. Programming languages also fall into these categories. The most popular language designed for the transputer is Occam which allows an application to be described as

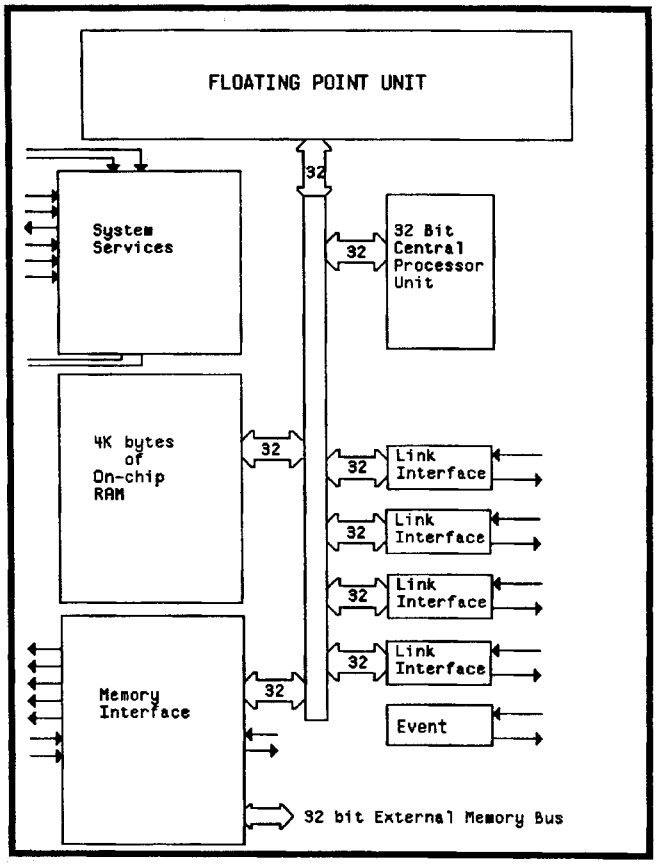


Fig. 2:

a collection of processes which operate concurrently and communicate through channels. Contemporary languages such as C, Fortran 77 and Pascal have been adapted and extended in various ways to allow sequential processes to communicate.

The transputer system used for the implementation of the discrete vortex method is a Meiko in-sun computing surface consisting of 16 T800 transputers, each with 4 Mbytes of external memory. The programming language is Meiko's Fortran 77 in conjunction with Meiko's Communicating Sequential Tools (CSTools). CSTools[19] is a program development toolset running, in this case, under UNIX, and consisting of cross-development tools and run-time facilities. Meiko's Fortran can call CSTools' library routines, enabling processes to communicate. CSTools provides a level of abstraction from details of the hardware, enabling the programmer to view the system as fully connected. CSTools will handle the transmission of data across the network or processors. The sending of an integer from one process to another can be achieved by the following example syntax:

```
INTEGER THISPROC, NEXTPROC, C
CALL CSN OPEN (CSN NULL ID, THISPROC)
CALL CSN REG NAME (THISPROC, 'Proc1')
CALL CSN LOOKUP NAME (NEXTPROC, 'Proc2', .TRUE.)
CALL CSN TX (THISPROC, 0, NEXTPROC, C, 4)
```

In this case, the destination processor need not be directly connected to the sending processor. CSTools will route the data to its destination.

Various sized domains of the Edinburgh Concurrent Supercomputer (ECS)[20] have been available to the authors for collecting results.

The T800 transputer is by no means the most powerful processor available for constructing distributed memory MIMD computers. The purpose of its use here is to investigate the optimum implementation of a problem on a similar architecture, although results may vary depending on the ratio between processing speed and data communication rates. One such processor which will be of interest in the implementation of computationally intensive problems is the INMOS T9000. Although not commerically available at the time of writing, technical specifications are available.[21,22] The T9000 has a peak performance of 25 MFLOPS or 200 MIPS with link speeds of 80 Mbytes s^{-1}.

4. PARALLEL IMPLEMENTATION

In order to find an optimum method of parallelizing the discrete vortex method, it is necessary to analyse the performance of a serial version. The method can be broadly decomposed into ten steps. Definition of the grid is carried out once, whereas the subsequent steps are carried out for each time interval, except for output of data and reordering of vortices which is undertaken less often. The percentage of elapsed time taken by each step of a run of 500 time intervals for a uniform flow at Reynolds number 10^4 is shown in Table I, where the overall elapsed time is 33 613 s.

It is clear that, calculation of nodal and surface velocities account for the high proportion of 85·9% of the total elapsed time. Based on this analysis, a first parallelization scheme shares the work involved in the two steps amongst available processors, while the remaining steps execute serially on one, the root, processor. This technique is often referred to as processor farming and is discussed by Hockney and Jesshope,[23] Pritchard[24] and Quinn.[25] The master process carries out all initialization, including the definition of the grid. Work pertaining to calculation of the surface velocities is shared out to the worker processes and resultant velocities received. New vortices are introduced to satisfy boundary conditions and circulations distributed to the nodes. These data are then allocated to the worker processes which calculate and return nodal velocities. The master process then distributes nodal velocities to the vortices, calculates forces and implements vortex displacements.

Table I. Breakdown of elapsed execution time

Step	Function	Elapsed time (%)
1	Definition of polar grid	0·00
2	Calculation of surface velocities	9·64
3	Reordering vortices	0·27
4	Introduction of vortices	0·02
5	Output of results	2·70
6	Distribution of vortex circulations	0·83
7	Calculation of nodal velocities	76·26
8	Distribution of velocities	1·61
9	Calculation of forces	1·21
10	Displacement of vortices	6·56

The portioning of work to worker processes is carried out in one of two ways. The first method uses a prescheduled partitioning scheme where the work to be done by each worker process has been decided at compilation time. Given n non-zero nodal velocities to be calculated by p processes, then the number assigned to process i is given by

$$n^{(i)}=\text{int}\left(\frac{n+i-1}{p}\right), \tag{7}$$

which has the effect of balancing the load on each processor. Figure 12 (see the appendix) outlines the master and worker process algorithms when using this approach.

A second method uses a self-scheduled partitioning mechanism whereby the work done by each worker process is not defined until run time. Each worker process is initially given one node to evaluate, after which the calculated velocity is returned to the master process, indicating that the worker process is idle and ready to receive another node. The master process receives calculated velocities from the worker processes as they become free and sends data for the next node to be evaluated to the sending worker process. The procedure continues until all nodal velocities have been calculated. This scheme approaches the problem of minimising worker process idle time rather than load balancing. Figure 13 (see the appendix) outlines the algorithm for portioning computation to worker processes. Concepts of pre- and self-scheduled partitioning algorithms are discussed by Day[26] and Quinn.[25]

A second parallelization scheme not only distributes the surface and nodal velocity calculations but also shares the work involved in distributing data to and from nodes (Steps 6 and 8), calculating force coefficients (Step 9), implementing vortex displacements (Step 10) and reordering of vortex data (Step 3). The algorithm requires the distribution of vortex data among worker processes, necessitating larger volumes of data communication between the master and worker processes. However, this disadvantage is offset by the fact that a higher percentage (approximately 96·38%) of the program is carried out in parallel. The master process still undertakes the initialization and definition of the grid and introduces new vortices. However, once the vortices are created, they are distributed among the worker processes. If the number of vortices introduced is v (where v is the number of segments defining the polar grid), the number of worker processes is p and the time-step count is t, then the number of vortices sent to process i is given by

$$v^{(i)}=\text{int}\left[\frac{v+f(i)-1}{p}\right], \tag{8}$$

where

$$f(i)=\text{mod}(i+t-2, p)+1.$$

The purpose of the function f is to balance the load for each process. If omitted, then after t time steps, some worker processes may be processing t more vortices than others. After the distribution of the vortices, each worker process distributes the circulations of its vortices to neighbouring nodes. The nodal circulations are then sent to the master process which accumulates them and distributes them back to the worker processes. Each worker process, i, evaluates $n^{(i)}$ nodal velocities, where $n^{(i)}$ is given by equation (7), and sends results to the master process. The master process collates the nodal velocities and sends all non-zero velocities to the worker processes. Each worker process then distributes nodal velocities to its stored vortices, calculates partial forces and sends them to the master process for accumulation. Each worker process displaces it vortices and the process is repeated for the next time step. After initial introduction and distribution of vortices, the only processing of vortex data undertaken by the master process

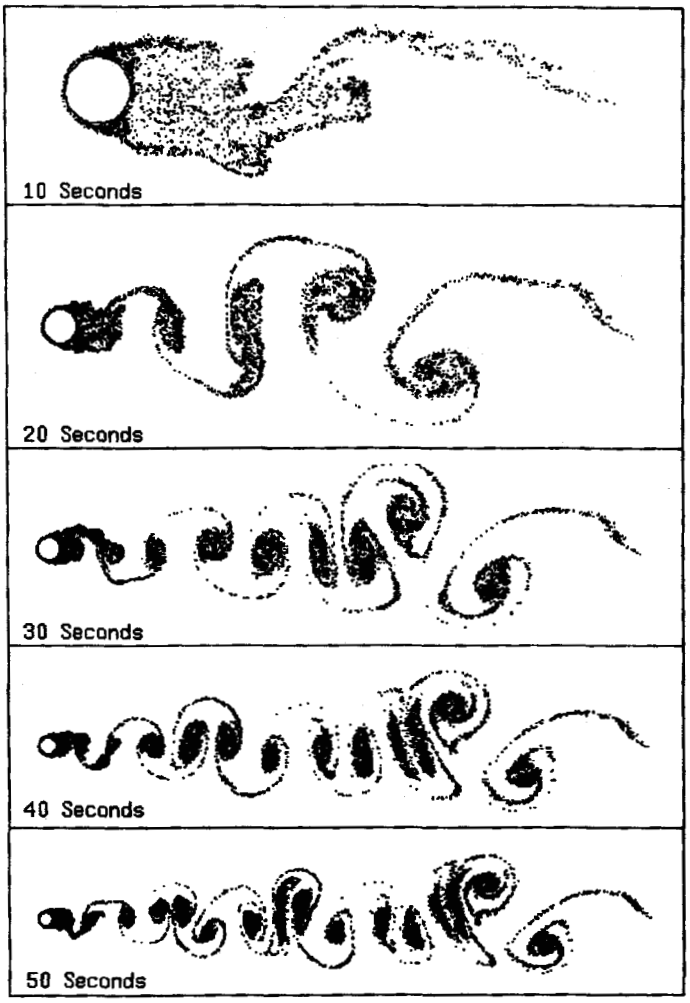


Figure 3. Flow visualization

is that required during output of data. Figure 13 (see the appendix) outlines the master and worker process algorithms for nodal/vortex parallelization.

5. RESULTS

Elapsed execution times for computation of a uniform flow at Reynolds number 10^4 about a circular cylinder have been recorded for 500 time steps of interval 0·1 s. The flow visualization produced by the discrete vortex model is shown in Figure 3 with associated non-dimensional drag (in-line with the direction of flow) and lift (transverse with respect to the flow) forces shown in Figure 4. The results on the 16-transputer Meiko computing surface for the three parallelization schemes are shown in Figures 5–7. The results for the same flow computed on upto 64 transputers using an ECS domain are shown in Figures 8–10.

The figures show the speed-up obtained by running the program with different numbers of worker processes. Speed-up is defined as the ratio between the time taken for one transputer to

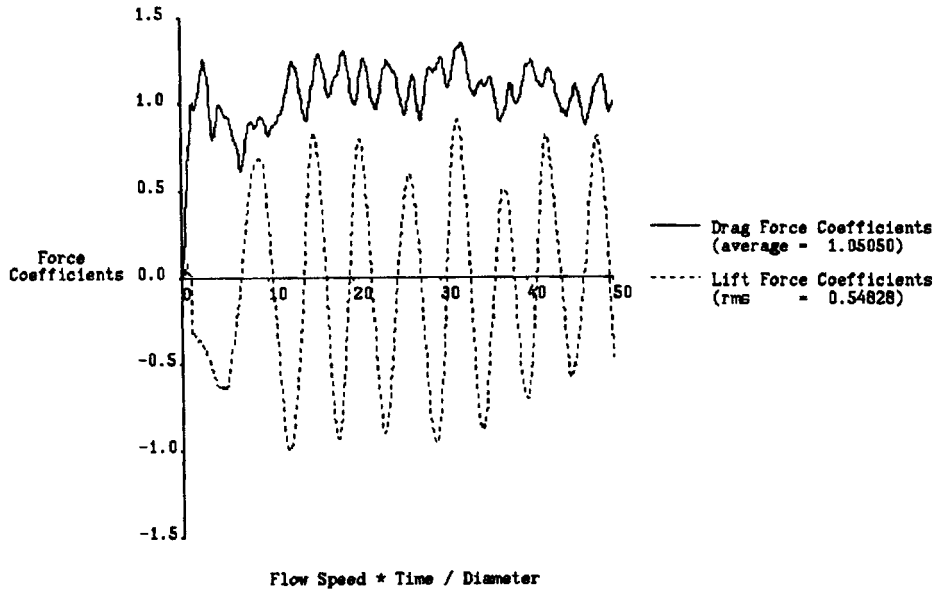


Figure 4. Drag and lift forces

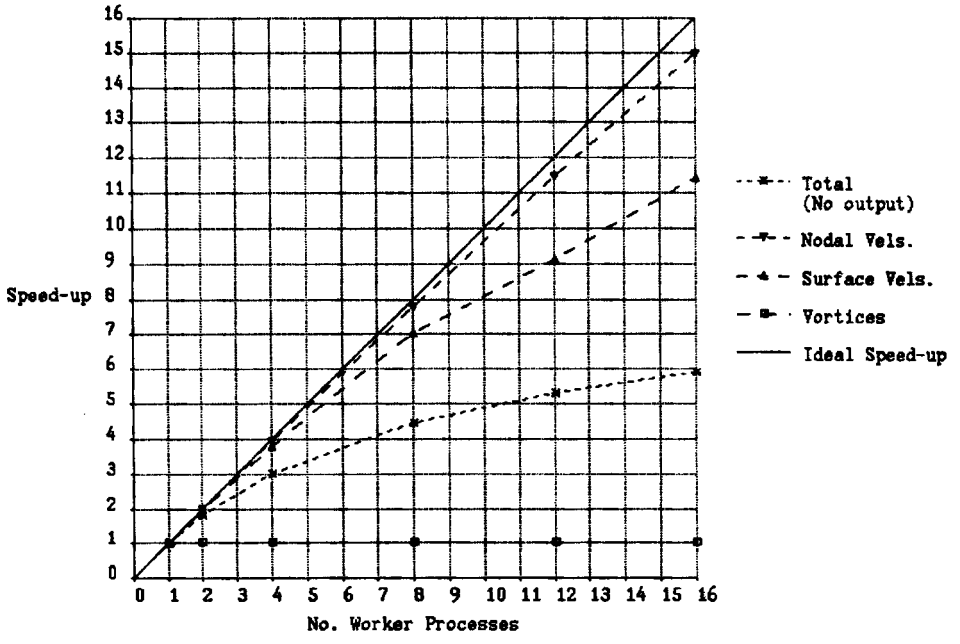


Figure 5. Speed-up on 16-transputer computing surface using prescheduled nodal parallelization

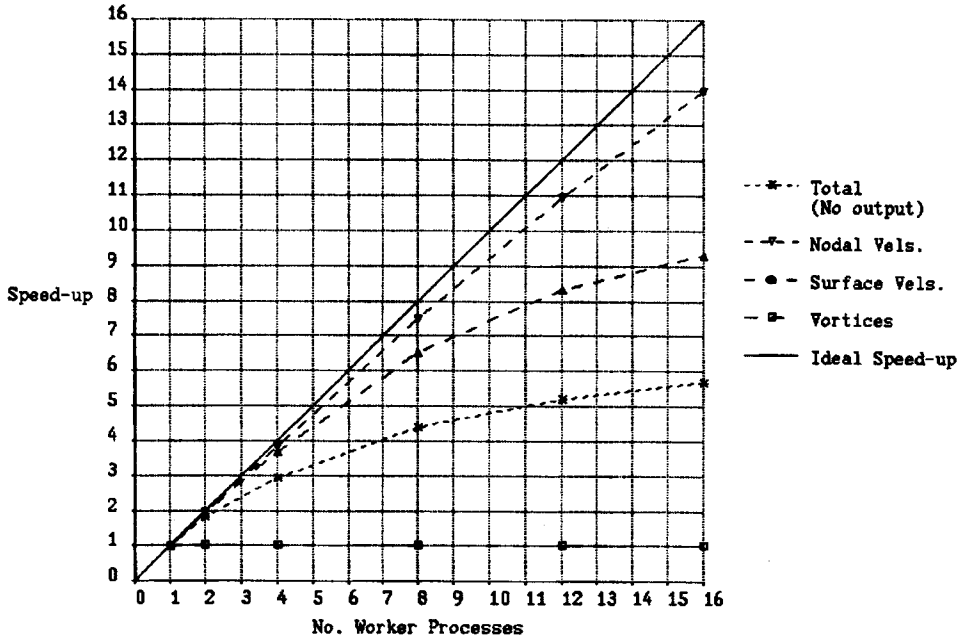


Figure 6. Speed-up on 16-transputer computing surface using selfscheduled nodal parallelization

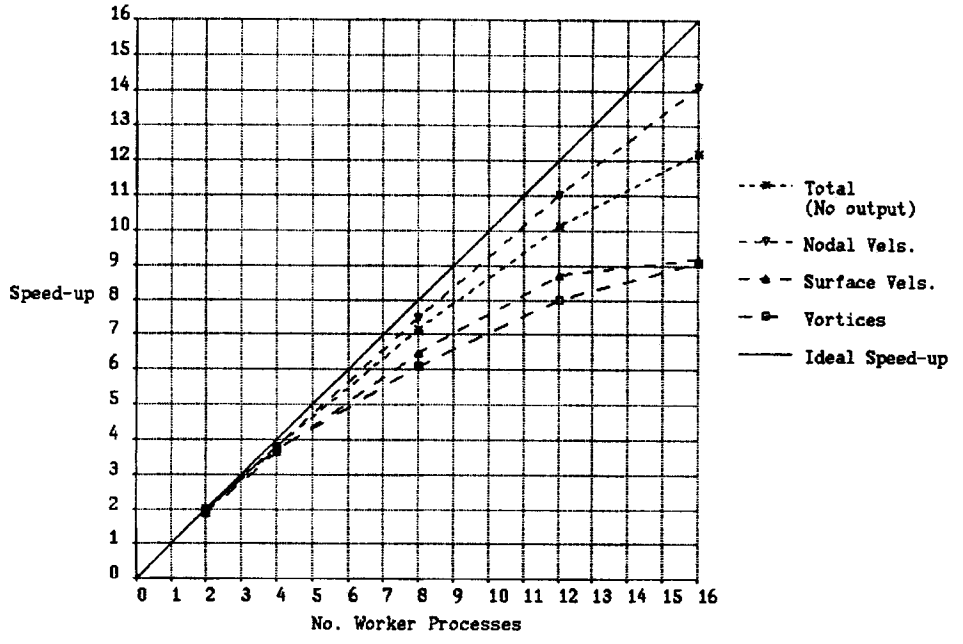


Figure 7. Speed-up on 16-transputer computing surface using nodal and vortex parallelization

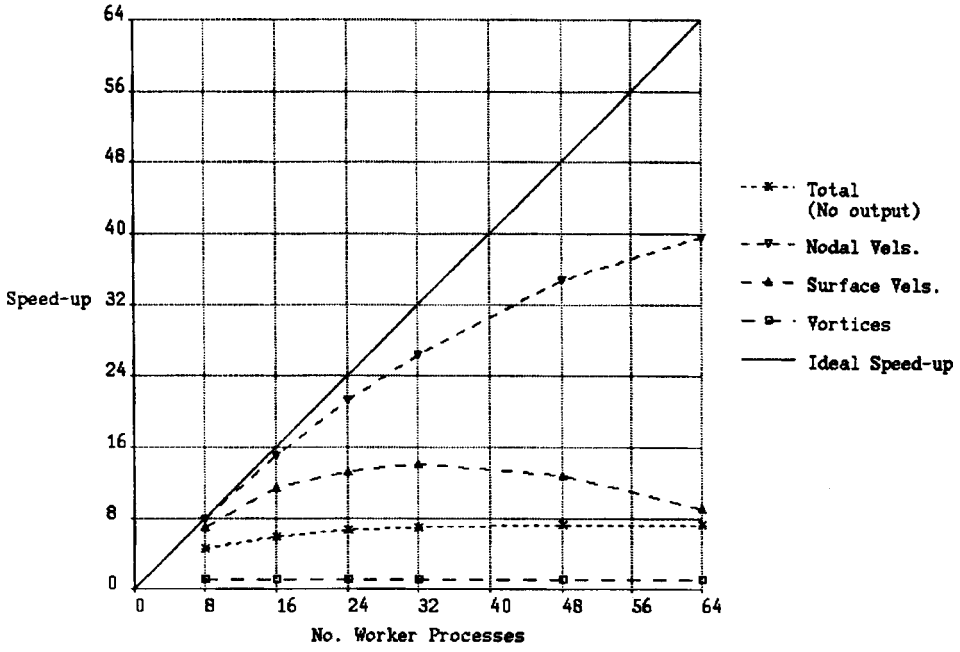


Figure 8. Speed-up on 64-transputer ECS domain using prescheduled nodal parallelization

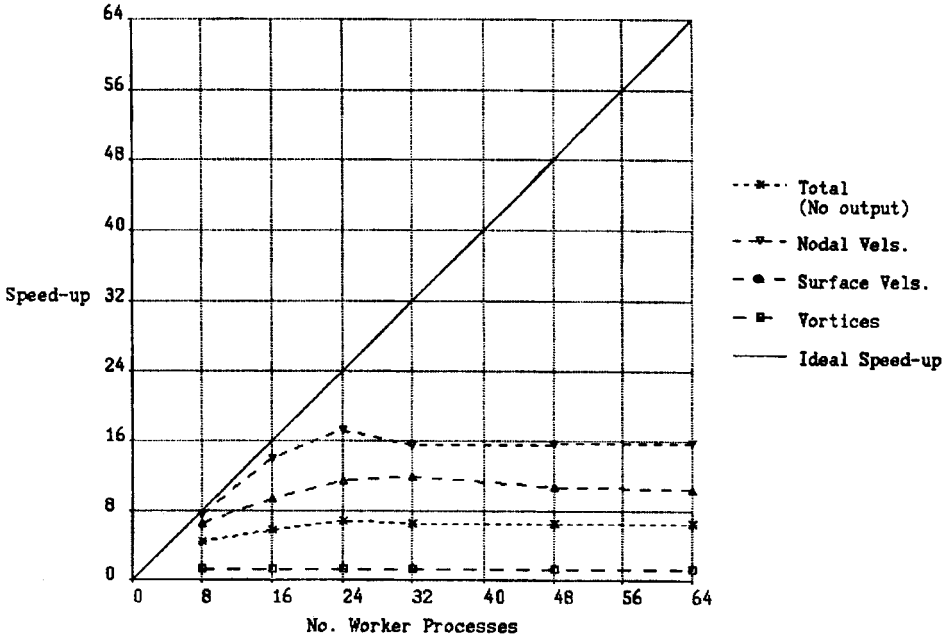


Figure 9. Speed-up on 64-transputer ECS domain using selfscheduled nodal parallelization

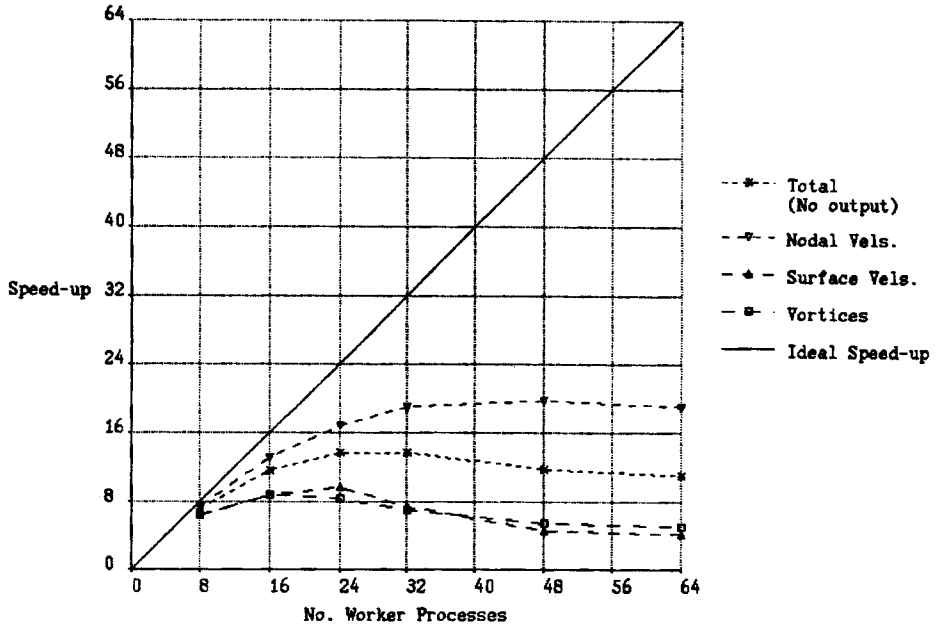


Figure 10. Speed-up on 64-transputer ECS domain using nodal and vortex parallelization

run the serial discrete vortex program, T_s, and the time taken for p transputers to run the parallel program using p worker processes, T_p,

$$S_p = \frac{T_s}{T_p}. \tag{9}$$

This definition of speed-up is given by a number of authors, including Kerry and Martin[27] and Quinn.[25] CSTools' default transputer link configuration has been used for all test cases since the default allows all available links to be connected and as the program requires communication between the master and all worker processes, but not between worker processes, this configuration achieves the best results. The configuration essentially defines a ternary tree with links that would not normally be connected used to increase the system software's choice of data packet routing between processors. The configuration of an 8-transputer domain is shown in Figure 11.

Results shown in Figures 5–10 have been subdivided to show the speed-up obtained from parallelization of nodal velocity calculations (Step 7), surface velocity calculations (Step 2), and processing of vortices (Steps 3, 6, 8–10). The overall performance of the parallel program excluding output (Step 5) is also shown. Output of the vortex data has been excluded from the timings since the operation is dependent on devices external to the transputer network and therefore tends to vary with the load on such devices.

The pre- and selfscheduled nodal parallelization schemes achieve similar results. The pre-scheduled method performs marginally better with a speed-up of 5·88 on 16 processors and a peak of 7·21 on 48 processors. The self-scheduled scheme achieves a speed-up of 5·69 on 16 processors and peaks at 6·69 on 24 processors. Note that the performance of the self-scheduled scheme degrades faster than the pre-scheduled as the number of processors increases. Nodal velocity calculations attain a speed-up of 39·59 on 64 processors using the pre-scheduled algorithm compared to only 17·21 using the self-scheduled algorithm, indicating a high level of idle time. This is mainly due to the difference in grain size of the two algorithms, where grain size,

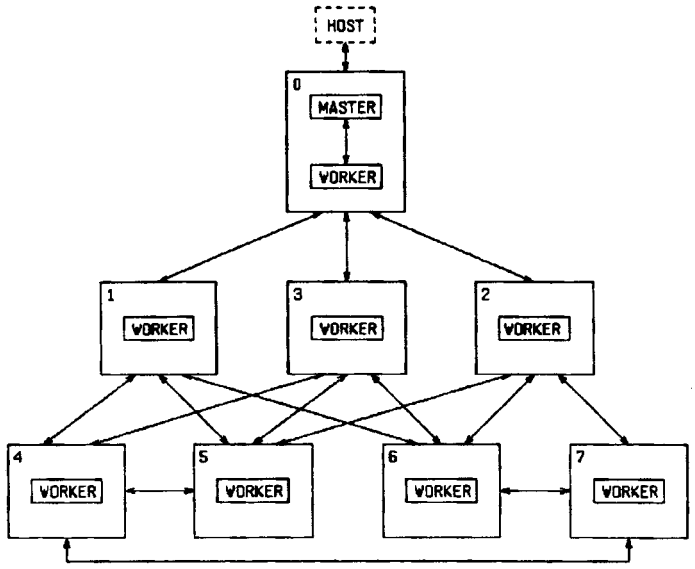


Figure 11. Eight transputer Meiko default configuration

as defined by Almasi and Gottlieb,[28] is the average subtask size measured in instructions executed. The grain size of the self-scheduled algorithm is always the work involved in the processing of one node, whereas the granularity of the pre-scheduled algorithm for processor i is that associated with $n^{(i)}$ nodes as defined by equation (7). After 500 time steps the number of active nodes is 560 and so the grain size is eight or nine times larger on 64 processors than that of the selfscheduled algorithm.

The advantage of this scheme is the simplicity of its implementation. The most computationally intensive parts of the serial program have been identified and parallelized. The code required to distribute work for the parallelization is small and the majority of the code remains unchanged and is executed as part of the master process. However, when only a proportion of the program is parallelized, then the effect of the serial portion becomes more prominent as the number of processors increases and the overall speed-up of the system is asymptotically limited by the serial portion. This limitation is known as Amdahl's Law.[29] By dividing the time to run the parallel program, T_p, into the serial portion, t_s, and parallel portion, t_p, then the speed-up is given as

$$S_p = \frac{T_s}{t_s + t_p} \tag{10}$$

and the maximum speed-up is

$$S_\infty = \frac{T_s}{t_s}. \tag{11}$$

Thus, for the nodal parallelization schemes, where 85·9% of the serial execution time is parallelized, a maximum theoretical speed-up of only 7·09 can be achieved, or 8·54 if output is excluded. As detailed earlier, the pre-scheduled scheme achieves a maximum speed-up of 7·21 when excluding output, which is approaching the theoretical maximum, indicating that the parallel portions are executing efficiently. A second disadvantage of this scheme lies with the memory requirements of the master process. Since much of the code is serial, then this code and associated data must reside on the root processor. Processing of all vortex data is carried out by the master

process and as such, all vortex data arrays must be held in the root processor's memory. This has the effect of limiting the number of stored vortices to approximately 28 000 given 4 Mbytes of external memory. In terms of time steps, the number is limited to around 1250.

The nodal/vortex parallelization scheme achieves better overall results with a speed-up of 12·2 on 16 transputers and a maximum speed-up of 13·64 on 24 transputers. Although the performance of the surface and nodal velocity stages are degraded in comparison to the previous parallelization schemes due to the increased volume of data communication and synchronization necessary, the speed-up of processing vortices compensates well. Furthermore, as the flow progresses, the number of vortices increases linearly, whereas the number of active nodes tends to level out after 200 time-steps. Thus, the processing of vortex data becomes more suited to parallelization as the flow progresses.

This scheme is much more complex to implement than the previous algorithms since each stage of the process is dependent on the previous stage and so there is a need for high volumes of data communication between stages. However, the effort involved is rewarded by its results. Since 96·38% of the serial execution time is now parallelized, the potential maximum speed-up (see equation 11) is 27·62 (and 105·26 if output is excluded), although this is not achieved due to a poor communication/computation ratio and also to the number of synchronization points within each time-step. Furthermore, as the vortex data are distributed among worker processes, the limit on the number of stored vortices is increased as the number of processors increases. Given 4 Mbytes of external memory shared by one master and one worker process, space can be allocated for approximately 12 000 vortices. On a 16-transputer system, approximately 192 000 distributed vortices may be stored, allowing up to approximately 8700 time steps to be processed. On a 64-transputer domain, figures are increased to 768 000 vortices and 35 000 time steps.

A refined nodal and vortex parallelization scheme could maximize overall speed-up by examining the peak speed-up of the nodal velocities, surface velocities and vortex processing stages. By executing each of these subdivisions on 32, 24 and 16 processors a potential speed-up on 32 processors of 14·28 is achievable. However, the scheme is difficult to implement since the peak speed-ups will differ depending on the type of flow and for how long it is modelled.

6. CONCLUSIONS

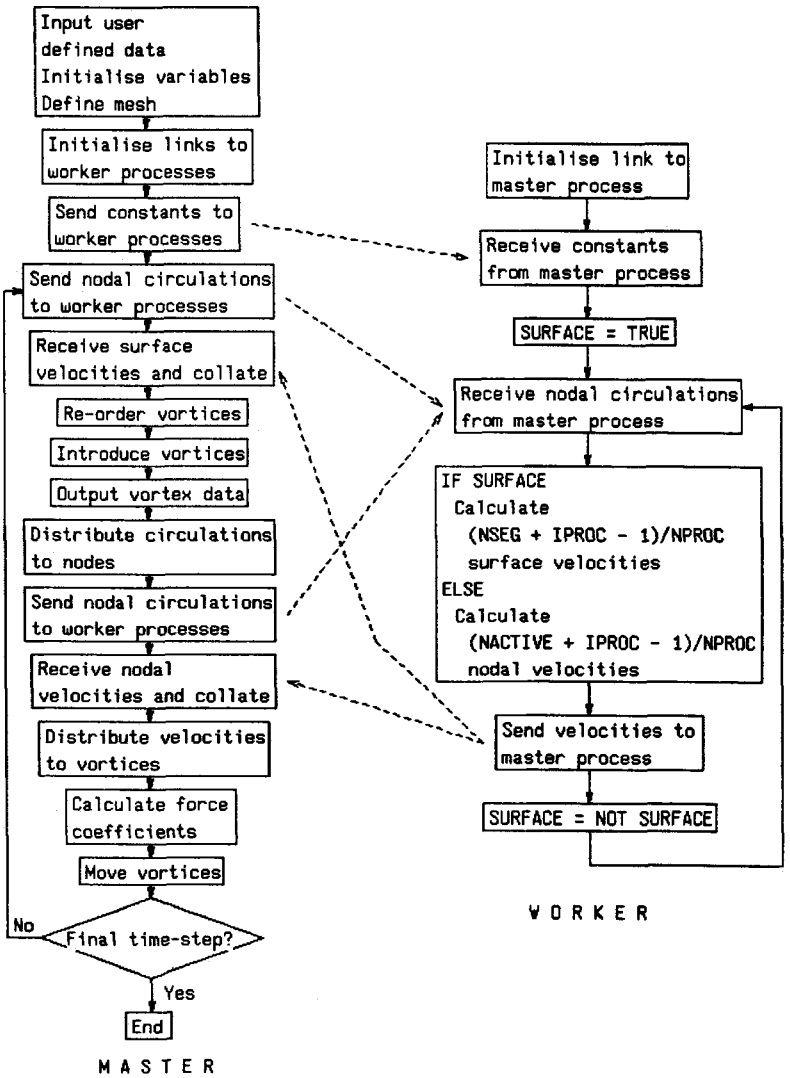
The aim of this study was to examine the feasibility of implementing an existing discrete vortex model on a distributed memory MIMD machine, in this case a transputer system. The advantages of using parallel hardware are obvious. As has been demonstrated, when there is a large amount of computation to be carried out then it can be accomplished on parallel processors in a fraction of the time necessary for conventional processors of a similar MIP or MFLOP rating. However, when data dependency between units of work is high, adding processors to the system will not necessarily increase its speed. Thus, given an optimal parallelization scheme, the financial cost of a parallel system and the effort involved in modifying the existing software must be balanced against the resulting increase in speed. Bearing in mind the foregoing, the physical limitations of single processors, and the continuing demand for the solution of ever larger problems, distributed parallel processing may well be judged the best approach to achieving the required results within practical time constraints.

Within its limitations, the discrete vortex method is very powerful for treating an important range of practical flow problems. It has been shown that it is an excellent candidate method for parallel computation, perhaps to a greater extent than most competing CFD methods. The advent of parallel hardware is likely to lead to a much wider usage of the method applied to flows which, for all practical purposes, are inaccessible to treatment by conventional computer hardware.

ACKNOWLEDGEMENTS

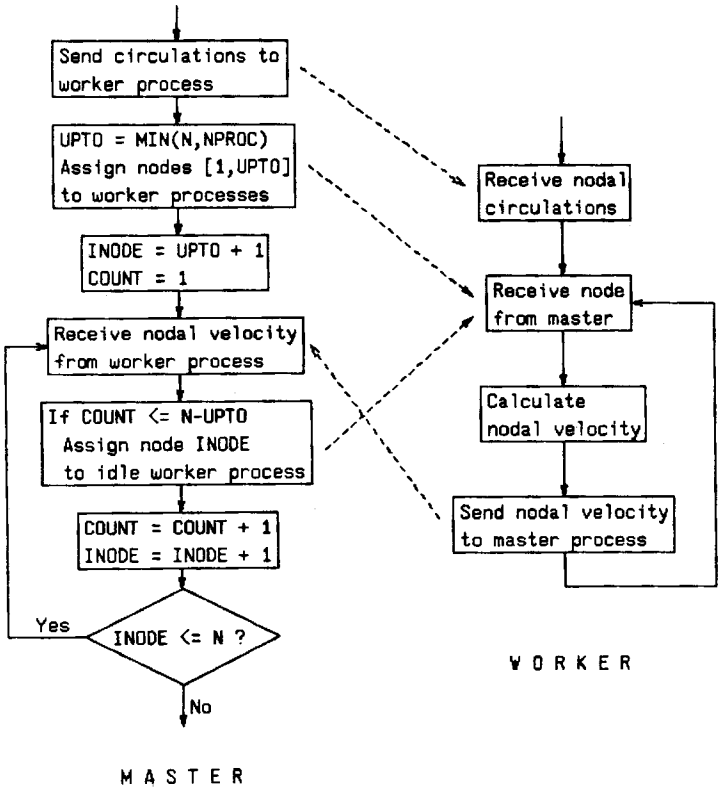
This report is part of a research project funded by the Science and Engineering Research Council and the Marine Technology Directorate, grant number GR/F/07903. We are grateful to Professor J. M. R. Graham of the Department of Aeronautics, Imperial College of Science, Technology and Medicine, for useful discussions.

APPENDIX



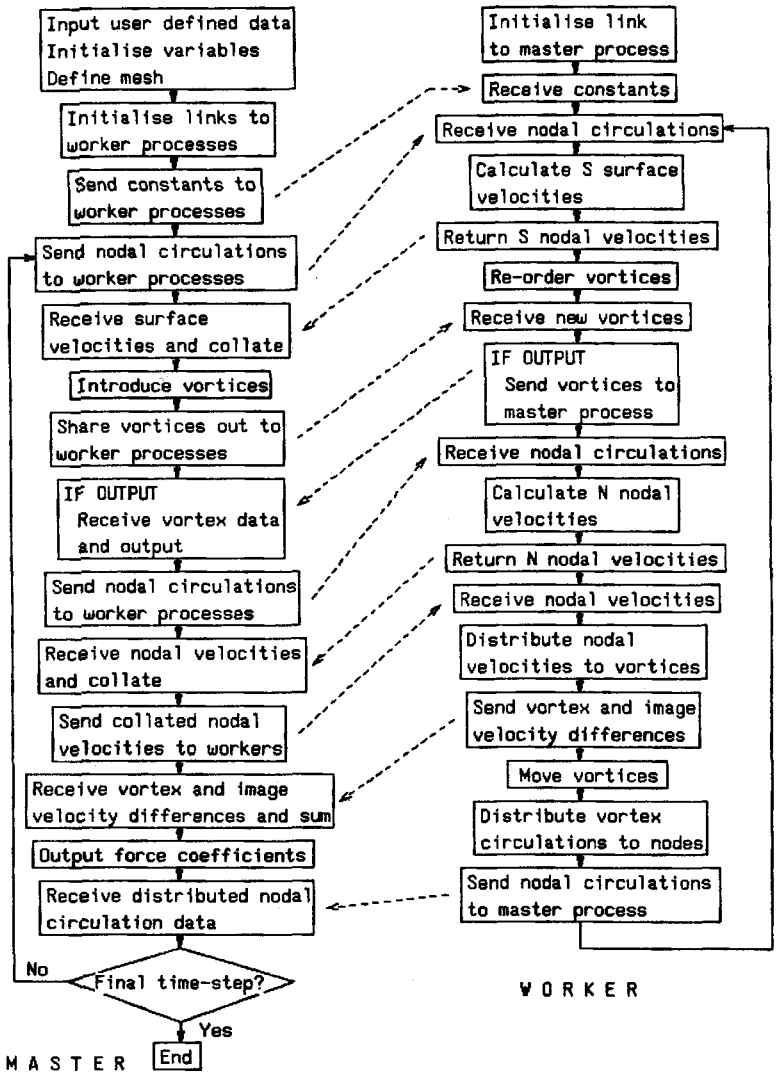
NSEG = Number of segments defining grid
NACTIVE = Number of nodes with non-zero circulation
IPROC = Worker process number
NPROC = Number of worker processes

Figure 12. Flow diagram for prescheduled nodal parallelization



NPROC = Number of worker processes
N = Number of nodes for which velocities are to be calculated

Figure 13. Flow diagram for selfscheduled nodal parallelization



OUTPUT is boolean and true if at a time-step designated for output
S=(NSEG+IPROC−1)/NPROC
N=(NACTIVE+IPROC−1)/NPROC
NSEG=Number of segments defining grid
NACTIVE=Number of nodes with non-zero circulation
IPROC=Worker process number
NPROC=Number of worker processes

Figure 14. Flow diagram for nodal/vortex parallelization

REFERENCES

1. M. F. Flynn, 'Some computer organisations and their effectiveness', *IEEE Trans. Comput.*, **C-21,** 948–960. (1972).
2. A. Trew and G. Wilson (eds), *Past, Present, Parallel: A Survey of Available Parallel Computer Systems*, Springer, Berlin, 1991.
3. P. J. Clarke, P. Bettess, G. E. Hearn and M. J. Downie, 'The application of the finite element analysis to the solution of Stokes wave diffraction problems', *Int. j. numer. methods fluids*, **12,** 343–367 (1991).

4. G. E. Hearn, 'Tank wall influences, higher order boundary elements, irregular frequencies and seakeeping for design', Discussion of ITTC Seakeeping Committee Report, *ITTC Proc.*, Vol. II, 1990.
5. N. Hardy, P. Bettess and M. J. Downie, 'Wave diffraction problems solved with a distributed memory MIMD computer', (to be published).
6. P. W. Bearman, 'Vortex trajectories in oscillatory flow', *Proc. Sep. Flow around Mar. Str.*, Trondheim, 1985, pp. 133–153.
7. C. H. K. Williamson, 'Sinusoidal flow relative to circular cylinders', *J. Fluid Mech.*, **155**, 141–174 (1985).
8. J. H. Gerrard, 'Numerical computation of the magnitude and frequency of lift on a circular cylinder', *Phil. Trans. Royal Soc.*, **261**, 137 (1967).
9. T. Sarpkaya, 'Analytical study of separated flow about circular cylinders', *Trans. ASME, J. Bas. Eng.*, **90**, 511–520 (1968).
10. R. R. Clements, 'An inviscid model of two-dimensional vortex shedding', *J. Fluid Mech.*, **57**(2), 321–336 (1973).
11. T. Sarpkaya, 'Computational methods with vortices—the 1988 Freeman lecture', *J. Fluid Mech.*, **111**, 5–52 (1989).
12. A. J. Chorin, 'Numerical study of slightly viscous flow', *J. Fluid Mech.*, **57**(4), 785–796 (1972).
13. R. I. Lewis and D. T. C. Porthouse, 'Recent advances in the theoretical simulation of real fluid flows', North East Institute of Engineers and Shipbuilders, March 1983.
14. J. M. R. Graham, 'Computation of viscous separated flow using a particle method', *Proc. Conf. Numer. Meth. for Fluid Mechanics*, Oxford, 1988 pp 310–317.
15. B. A. Murray, 'Hydrodynamic loading due to appurtenances on jacket structures', *Ph.D. Thesis*, Dept. Marine Technology, University of Newcastle upon Tyne, 1992.
16. INMOS, 'Transputer development system', Inmos Ltd., 1988.
17. D. May, 'The transputer', in G. Harp (ed.), *Transputer Applications*, Pitman, London, 1989.
18. INMOS, 'Product overview: IMS T800 transputer', INMOS, 1986.
19. Meiko, 'CSTOOLS for SunOS', Meiko Ltd., 1990.
20. N. MacDonald, M. Smith and N. Stroud, 'Introduction to the Edinburgh concurrent supercomputer', Edinburgh Parallel Computing Centre, 1991.
21. R. Rinn, 'H1—The next generation', *Parsytec News*, 1991.
22. 'The Inmos H1 Becomes the Inmos T9000—A Better Balanced Microprocessor', SERC/DTI Transputer Initiative Mailshot, May 1991, pp. 17–23.
23. R. W. Hockney and C. R. Jesshope, 'Parallel computing 2: architecture, programming and algorithms', IOP Publishing Ltd, 1988.
24. D. J. Pritchard, 'Transputer applications on supernode', in L. Freeman and C. Phillips (eds), *Applications of Transputers 1*, IOS Press, Amsterdam 1990, pp. 48–56.
25. M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.
26. A. M. Day, 'Parallel implementation of 3D convex-hull algorithm', *Comput. Aided Des.*, **23**(3), 177–188 (1991).
27. N. Kerry and S. Martin, 'Adapting an oil reservoir simulation package for a transputer array', in L. Freeman and C. Phillips (eds), *Applications of Transputers 1*, IOS Press, Amsterdam 1990, pp. 215–222.
28. G. S. Almasi and A. Gottlieb, '*Highly Parallel Computing*, The Benjamin/Cummings Menlo Park, CA, 1989.
29. G. Amdahl, 'The validity of the single processor approach to achieving large scale computing capabilities', *AFIPS Conf. Proc. Spring Joint Comput. Conf. 30*, 1967, pp. 483–485.