ISSN 0141 3818
SSSU 63

# PARALLEL ALGORITHM FOR SURFACE PANEL ANALYSIS

by S.R. Turnock

Ship Science Report 63

August 1993

ISSN 0141 3818
SSSU 63

UNIVERSITY
OF
SOUTHAMPTON

DEPARTMENT OF SHIP SCIENCE

FACULTY OF ENGINEERING

AND APPLIED SCIENCE

# PARALLEL ALGORITHM FOR SURFACE PANEL ANALYSIS

by S.R. Turnock

Ship Science Report 63

August 1993

# PARALLEL ALGORITHM

# FOR

# SURFACE PANEL ANALYSIS

by

S.R.Turnock

Ship Science Report No. 63

University of Southampton

August 1993

## SUMMARY

This report describes the development of a parallel algorithm to carry out surface panel analysis. The algorithm is designed to run across variable sized arrays of transputers. All software was written in Occam2. The parallelism strategy was based on geometric parallelism for both the data and code. A flexible geometry definition allows a wide range of flow geometries to be investigated. A perturbation potential formulation is used with an iterative Kutta condition to give zero pressure loading at the trailing edge. An investigation was carried out to compare the relative merits of iterative and direct methods for solving the influence coefficient matrix. An interative block Jacobi was found to give the best performance when the block size was equal to the number of panels in a chordwise strip. The overall performance of the PArallel LIfting SUrface PANel code (PALISUPAN) was measured and although an implicit method it gave a high code effficiency.

## TABLE OF CONTENTS

## LIST OF FIGURES

## 1 Introduction

The implementation of a lifting surface panel method to run on an array of transputers using the developed communications harness is described in this report. The theoretical basis is given in Ref. [1]. A suite of procedures for carrying out the various stages of the analysis has been written and is referred to as the PALISUPAN (**PA**rallel **LI**fting **SU**rface **PA**nel) code. A geometric parallelism was used for the data distribution and the numerical formulation of PALISUPAN. The parallelism is based on equally dividing the total number of lifting surface and wake panels amongst the numbers of transputers available on a given parallel computer.

All the software was written in Occam2. Where necessary, important programming techniques have been included directly in the text. The overall software design philosophy was to minimise the development time and subsequent debugging by the use of simple geometric algorithms. A structured approach making full use of the procedures and channel communications of Occam2 allowed this to be successfully carried out.

An interesting part of the work described in this report deals with techniques for the solution of simultaneous linear equations. In the context of a geometric parallelism over a square array of transputers both direct and iterative methods for solving matrices were investigated. The speed performance of these methods for different sizes of problems, numbers of transputers, and memory storage requirements were considered.

A final section gives the overall performance of the complete lifting-surface panel analysis.

## 2 Implementation of PALISUPAN on a Square Array of Transputers

A variety of methods can be used to produce parallel algorithms to solve a lifting surface panel problem with a total of N panels using T transputers. A parallel geometric algorithm where each transputer is assigned (N/T) panels is the simplest method. This helps ensure an even computational load-balance between the transputers. Also, problems with different total number of panels can be easily scaled without the need to alter the software simply assigning a different number of panels to each transputer.

The coarse-grained parallelism of transputers naturally lends itself to a geometric approach in computational fluid dynamics. This and the fact that it is the simplest approach to develop were the reasons for choosing geometric parallelism. It should be noted that the communications harness allows any topology of transputer network or parallelism strategy and this did not act as a restriction on the choice of method.

The use of a square array of transputers is attractive for matrix type operations which involve setting up and solving N simultaneous linear equations. As information about each panel has to be sent between all the transputers in the array the distance travelled by information (number of transputer links crossed) should be minimised and ideally a minimal diameter array should be used. The diameter of a transputer network is defined as the minimum number of transputer links which a message has to cross when travelling the shortest distance between two transputers. Work has been carried out to investigate possible topologies for minimal diameter networks. For example, Allwright[2] showed that a network of 41 transputers can be connected with no more than 3 links to be crossed for a communication between two transputers. At present, it is not a simple matter to make the software scalable for changes in the number of transputers if the topology of the network changes as well. A square array of transputers which have spare links joined to form a closed surface was considered a satisfactory approximation to a minimal diameter network. For this case the diameter of the network is equal to the number of transputers along the side of the square network. It should be noted that on average that the number of links most messages have to pass across will be less than this.

The choice of a square array of transputers restricts the number of transputers which can be used to 1, 4, 9, 16, and so on. However, the programming effort is significantly reduced. There will be less need for complex routeing strategies and the sub-division of panels and influence matrices can be directly mapped onto a square array of transputers.

## 3 Geometric algorithm for PALISUPAN

The data distribution and the numerical algorithm used geometric parallelism. The overall structure of the program was the same as that used for the successful implementation of the Euler solver. At the outermost level, the program consists of the harness and guest (or host) process running in parallel. The method of initialisation of the array of guest processes from the host process and of termination are identical to that described in Ref. [3].

The development of the complex lifting surface panel code requires the successful implementation of a number of separate stages. Each stage, for example the reading of the geometry input file, generating body panel co-ordinates, calculating influence coefficients and so on, has to be individually checked and verified. To aid this process it was decided to use a menu-driven structure for the lifting-surface program. Every stage is written as a separate process which only executes when chosen. The structure of the Host and Guest processes to achieve this were as follows:

```
HOST                              GUEST
SEQ                               SEQ
  ... initialise network            ... initialise network
  finito:=FALSE                     finito:=FALSE
  WHILE finito=FALSE                WHILE finito=FALSE
    SEQ                               SEQ
      ChooseMenuOption                  ReceiveMenuOption
      BroadcastOption                   ImplementOption
      ImplementOption               ... Terminate
  ... Terminate
```

This allows each individual process or stage to be independently developed, checked, verified, and its performance on varying sized arrays of transputers to be measured. Keyboard input is used to choose from the range of options displayed. The Host process then broadcasts this option to all the Guest processes. A CASE statement is used to select the correct option of both Host and Guest processes once the option choice message has been received.

Once the program is fully developed the menu structure can be dispensed with and the program rewritten as a sequential series of processes. Figure 1 is a flow chart of the overall lifting-surface panel program. The various processes will be described in subsequent sections.

The scaling of a lifting-surface problem of a given total number of panels onto the available number of transputers is of interest. The program was developed to allow the performance of the program and its component processes to be measured for both:

1) variable sized square arrays of transputers;

2) different total number of body panels across fixed sized square arrays of transputers.

The total number of panels which are geometrically distributed across an array is limited by the amount of external memory assigned to each transputer. To maximise the total number of panels available the memory requirements of the component processes were kept to a minimum. In some places this resulted in extra numeric processing but it was considered that the greater the number of panels per transputer the greater the use of the code. A common restraint on panel method performance is a limit on the number of panels which can be processed at once. This block size severely limits performance as a large amount of slow data transfer operations from storage to active memory are required for processing each block. By removing the need for this type of operation a considerable time saving is achieved at the small cost of extra numeric processing.

Occam2 TIMER channels allow the execution time of a process on a transputer to be measured in terms of an individual transputer's internal clock. Considerable amounts of detailed information can be obtained if timings are made on the individual guest processes. However, the inspection of the transputer clock involves a finite amount of time which, if too many inspections are made, can impair the code performance. It was decided that the performance of the component processes of the lifting-surface code would be measured only on the Host process. As each process starts and finishes by communication with the Host process the total execution time is easily obtained. Further detailed study of the performance of the component processes is of limited long-term interest and was therefore not considered important in this work.

## 4 Lifting Surface Definition

### 4.1 Geometric Data Distribution

The ease with which the geometry of a lifting-surface problem can be distributed across the array of transputers will determine the usefulness of the fluid dynamic code. As discussed in Ref. [1], it was decided in this work to use a scheme where each lifting-surface body is defined in terms of series of lines of points. A parametric cubic spline procedure is then used to sub-divide the body surface into the required number of surface panels. As only a small number of points are used to initially define each body it was decided that every guest process (transputer) would be sent the complete problem geometry. From the overall

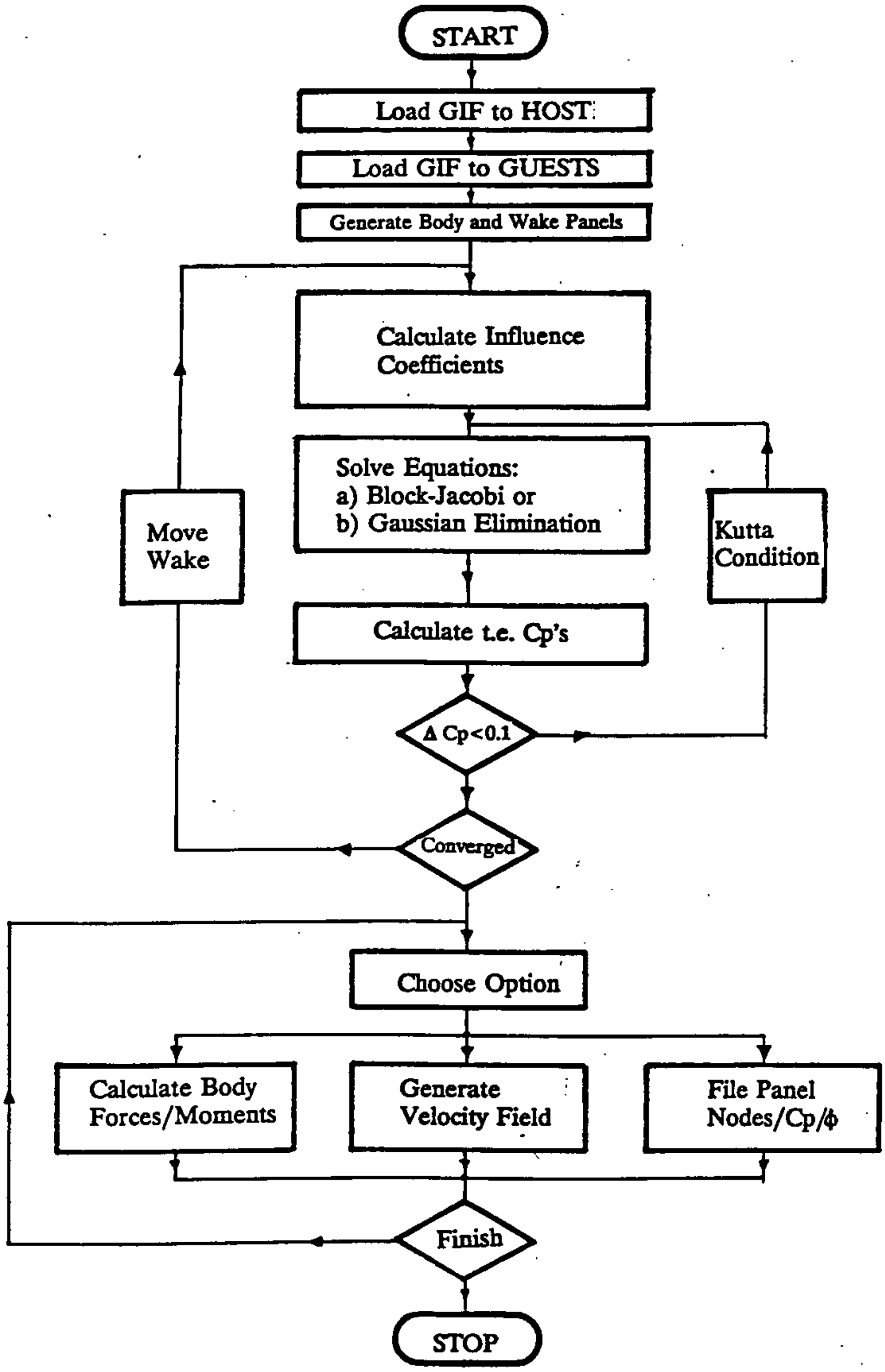


Figure 1 Flow chart of overall lifting surface algorithm

geometry definition, each guest process generates its allocation of surface panels. As the parametric cubic splines used on each process are based on identical information the panel boundary points between transputers will be same.

To ensure as even a load balance as possible and minimise delays each transputer is assigned, where possible, the same number of panels. In addition, as a considerable amount of numerical processing is required to calculate the wake strip influence coefficient, each transputer has the same number of wake strips. Delays may still occur as different schemes are used to calculate the panel-node influence coefficient in the near, mid and far field.

The method chosen to ensure an even load balance involved sub-dividing each body into streamwise panel strips with an equal number of streamwise strips assigned to each transputer. For example, this results in a two rudder problem as shown in Figure 2, where

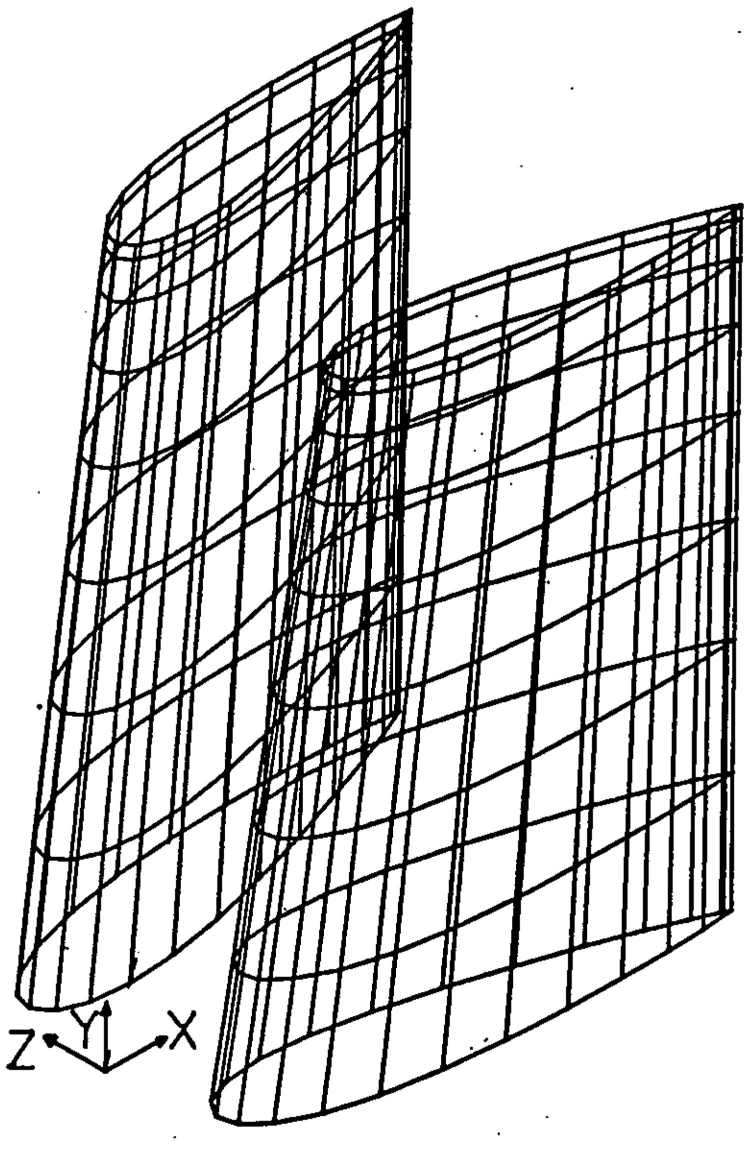


Figure 2 Panelling arrangement for two rudder arrangement

there are Nt streamwise panels by Ns spanwise panels for both rudders. Each transputer is assigned (Ns/T) strips of Nt panels for each rudder. This imposes a minimum limit of strips an individual body can be divided into, which is equal to the total number of transputers T (one strip per transputer). This is not considered a problem for moderate sized arrays of transputers ($T < 25$).

Each surface panel is assigned an unique absolute identity number to allow the dipole, source and wake influence coefficient matrices to be correctly assembled. A unique body identity number is used to define the correct relationship between individual panels on a body which is necessary for the calculation of surface potential derivatives. By relating the panel identity number to the individual transputer number in the array, the process of setting up and solving the coefficient matrices is made independent of a particular geometry. This allows multiple body problems to be easily solved.

In addition to the body panel data the velocity field and wake sheet data need to be sub-divided amongst the guest transputers. An identical method to that for the body is used for the wake sheet where the number of wake strips on each transputer are those corresponding to the body trailing-edges. As only a small amount of information is used to define a three-dimensional velocity field the complete field is defined on every guest process. The means by which memory storage requirements are minimised are described in the next section.

## 4.2 Data Store

In conventional Fortran or Pascal programs body geometry data, such as panel vertices [x,y,z], are stored in multi-dimensional variable arrays. The maximum number of elements in each dimension is specified at compile-time. This results in the need to recompile the program every time it is wished to alter, for example the balance between the maximum number of spanwise Ns and chordwise Nt panels. Also, for multiple body problems, each body either requires its own assigned array or to be assigned as a part of an overall array. This sort of programming problem results in cumbersome detail with inherent costs in programming and debugging time. A more sensible strategy is to write an explicit indexing procedure and store all the information as a single-dimension array of real numbers. The index procedure allows the required coordinate information to be directly accessed. The maximum size of the array can be related to the memory available on a transputer and a far more flexible approach to multiple body problems becomes possible.

The geometrical panel definition is used throughout the lifting-surface panel algorithm. Therefore, all the component processes have to be able to use the information. In Fortran such information is transferred between sub-routines using common-block type definitions. Similarly in Occam2 and Pascal the information can be accessed if the individual procedures are defined within the scope of the array definition. Again, the larger the amount of array information the more cumbersome the programming detail. Fortunately in Occam2 the whole problem can be circumvented by defining "live" storage. A storage array such as that for the panel vertices is written as a process running in parallel with the numeric algorithm. Information is accessed through communication along channels in and out of the memory store process. Considerable numeric processing can be carried out within the memory store process running in parallel, thereby helping to ensure satisfactory 'hiding' of inter-transputer communications across the harness.

The basic design of the "live" memory store developed for this work uses a one-dimensional array with a simple index procedure for access, the details of which are given in Appendix A. A single input channel with a tagged protocol is used to determine what operation the store carries out. The input statement is enclosed in a WHILE loop which terminates when a 'finish' tag input occurs. A representative store process is as follows:

```
PROC Store(CHAN OF pan In,Out)
  [MaxDim]REAL32 StoreArray:
  BOOL finito:
  SEQ
    finito:=FALSE
    WHILE finito=FALSE
       SEQ
         In ? CASE
           Startup
             ... initialise
           finish
             finito:=TRUE
           etc
             ...
```

The use of such a structure for memory storage has resulted in many programming advantages. The development time of individual component processes of the lifting-surface program is greatly reduced as many repetitive programming tasks can be included within the Store procedure. For example, determining a panel's vertices and centroid can be obtained for a given absolute panel identity number. Once the tagged process within the Store has received the panel identity number it picks out the four vertices from the array and uses them to calculate the panel centroid. The Store process then outputs the four vertices and centroid.

Another advantage is that data storage requirements for the component processes can be kept to a minimum. As the communication time for information is rapid, and simply involves the passage of the address in memory where the information is stored, information can be individually processed without the need to hold duplicate copies of data.

The Store structure allows simple means of checking and debugging, through the use of tag protocols, for inspecting the contents of the store array. For example, using the GetCentroid tag on the In channel the Host process can ask all the Guest processes to send all their individual panel coordinate information back to the Host for storage. The resultant panel geometry file is then inspected to check the panelling of the lifting surface.

The basic store structure allowed different variants to be developed all using the same tagged protocol but with different functions occurring within each store. A panel store, wake store, and velocity field stores were all produced. These will be discussed in the sections after the description of the format of the geometry input file.

## 4.3 Geometry Input File

A declared aim for the lifting-surface method was the ability to define a three-dimensional velocity inflow field and multiple lifting-surface geometries in a straightforward manner. A single fixed format input file was used. The layout of the geometry input file (GIF) is given in Appendix B.

The generation of the input file itself can be either carried out through the use of an ASCII file text editor or by writing of special programs to use information from, for example, CAD systems to generate the required format.

## 4.4 Body Surface Definition

Process PanelStore was written to store the relevant panel information on an individual transputer. All the guest processes are sent a complete copy of the geometry input file.

On receipt of the information, PanelStore uses the geometry information to define the panels it has been assigned. The vertex generation is carried out using the parametric cubic spline algorithm described in Ref. [1]. The relative spacing of the panel vertices is specified in the geometry input file. The following distributions are available and are illustrated in Figure 3 and the functions given in Appendix A.

The pivot, offset, scale, and angle transformations are carried out on the co-ordinates in PanelStore to obtain the final panel vertices in the overall cartesian co-ordinate system.

Once all the body surface panels have been defined across the array of transputers many different manipulations and inspections can be carried out. Only panel vertices are stored but, when needed, the centroid and panel unit normal vector are generated from the vertex information. The ordering of the panel vertices has to be consistent to ensure that all the body panel normals face outward from the body surface and into the exterior domain.

## 4.5 Wake Sheet Definition

A Wake Store process accepts the relevant information from the geometry input file. A fixed number of panels are distributed across the wake sheet in the same manner as the body surface panels. Once the wake sheet panels in the overall coordinate system have been generated, the location of the first (most upstream) panels are compared to their matching lifting surface trailing edge panels. If there is an offset the location of the whole wake sheet is translated so there is direct correspondence of the wakesheet start and the body trailing edge.

## 4.6 Velocity Field

The inflow velocity field is defined on a regularly spaced three-dimensional mesh of points. It represents the velocity field <u>in the absence</u> of the lifting-surface geometry to be tested. A velocity store on each transputer stores the whole velocity field after input from the geometry input file. In the store the velocity is scaled by the reference velocity to give the velocity in S.I. units (metres/second). The principle function of the store is to return,

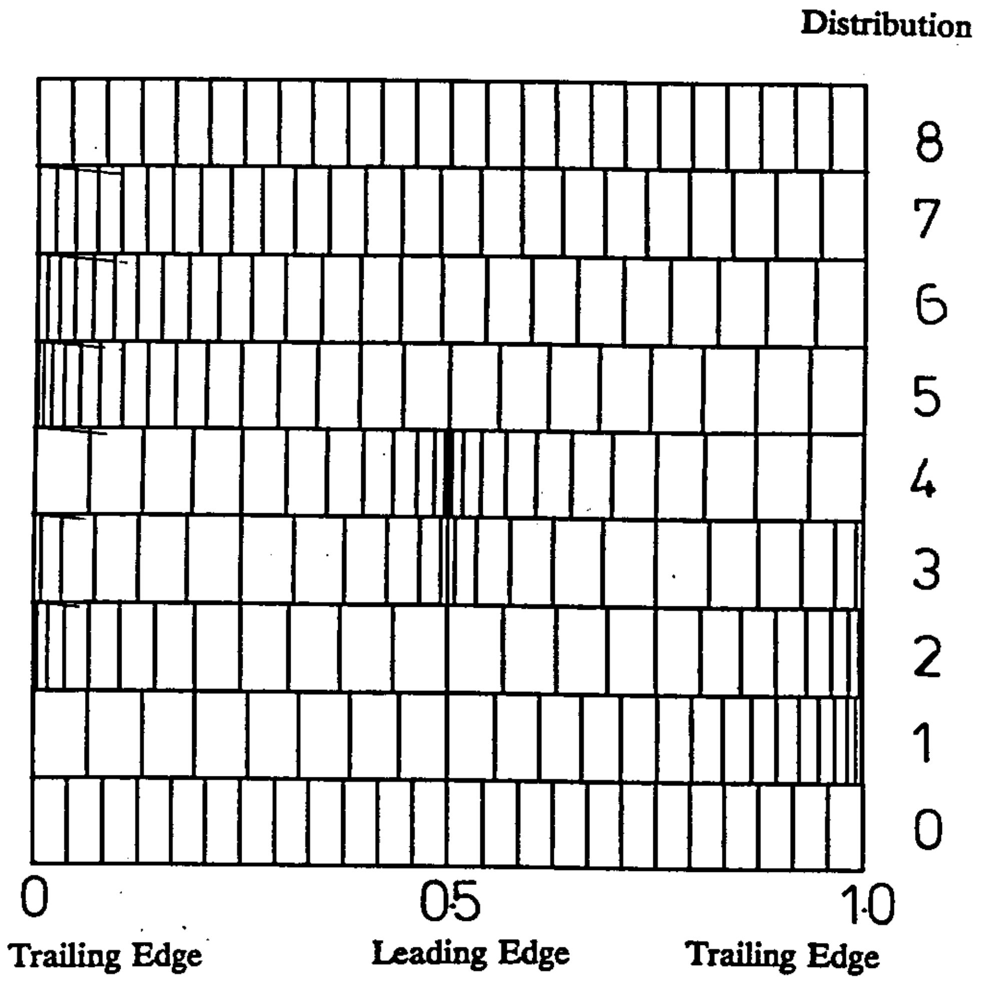


Figure 3 Possible panel size distributions

for a point anywhere within the domain of interest, the absolute inflow velocity at that point. A tri-directional linear interpolation is used to calculate the inflow velocity within the defined velocity mesh. External to the mesh the inflow velocity is set equal to the scale velocity in the freestream direction. Either a cylindrical or cubic three-dimensional grid can be used to define the velocity field. The choice is dependent on the application. For a circumferentially averaged inflow field less information is required to define the velocity field in cylindrical coordinates and this method is used for defining rudder-propeller interaction velocity fields.

## 4.7 Performance

The time for the geometry input file to be read, distributed to the array of transputers, and then the relevant store panels generated is a quick process. It is dependent on the number of bodies, number of sections for each body, number of wake sheets and the number of transputers. The lifting surface definition is a farm algorithm. The host process distributes the data packets and the individual guest processes reply on completion of the panel generation. The total time to distribute this information is proportional to the length of the geometry input file.

## 5 Calculation of Influence Coefficient

### 5.1 Introduction

The setting up of the lifting surface system of equations requires the calculation of the source and dipole influence coefficients $S_{ij}$, $D_{ij}$ between every panel (i=1 to N) and panel centroid (j=1 to N), and the wake strip influence coefficient $W_{kj}$ between every wake strip (k=1 to NWs) and panel centroid (j=1 to N). As each guest process stores a fraction (N/T) of the total number of panels considerable communication around the transputer network is required to allow the elements of the influence coefficient matrices $S_{ij}$, $D_{ij}$, $W_{kj}$ to be evaluated and stored.

The means by which information is transferred determines the efficiency of the setting-up process. The final results of the calculation are three matrices. The source and dipole matrices are of size N by N, where N is the total number of panels used to define the lifting surface. The wake influence matrix has dimensions NWs by N where NWs is the number of wake strips.

The boundary condition vector [U.n] at each panel centroid is known and hence each panel's source strength. So, rather than storing the elements of matrix $[S_{ij}]$ it is better to carry out the matrix-vector multiplication as part of the setting up process. This halves the required matrix memory storage.

To store the right hand side vector **RHS**, the dipole matrix $[D_{ij}]$ and wake strip matrix $[W_{kj}]$ three memory store variants were used. The dipole matrix is stored in a general square matrix routine which evenly distributes the matrix elements across the square array of transputers. This routine is used extensively in the solution of system of equations and is discussed in Section 6. Similarly, an equivalent general vector procedure is used for the vector **RHS** and a rectangular matrix for $[W_{kj}]$.

At this stage it is worth noting that, if there are N panels distributed across a network of T transputers, each transputer has information for (N/T) panels. The dipole matrix is of size $N^2$ and each transputer stores a sub-matrix of size $(N^2/T)$. This indicates that as the dimension of the square array increases, for a fixed amount of memory per transputer, the maximum number of panels located on each transputer will decrease. This is due to the matrix store progressively occupying more of the transputers memory. To illustrate this Figure 4 plots the total number of panels distributed across the array of transputers against the individual store memory requirement for each transputer in the network. The lines plotted are for different sized transputer networks ranging from 1 to 121. The memory requirement (in bytes) for a panel, wake and matrix store is approximated by:

$$M = 4 \times \left[ 3 \times 1.5 \times \left[ \frac{N}{T} + 1 \right]^2 + 2 \times \left[ \frac{N^2}{T} \right] \right] \qquad [1]$$

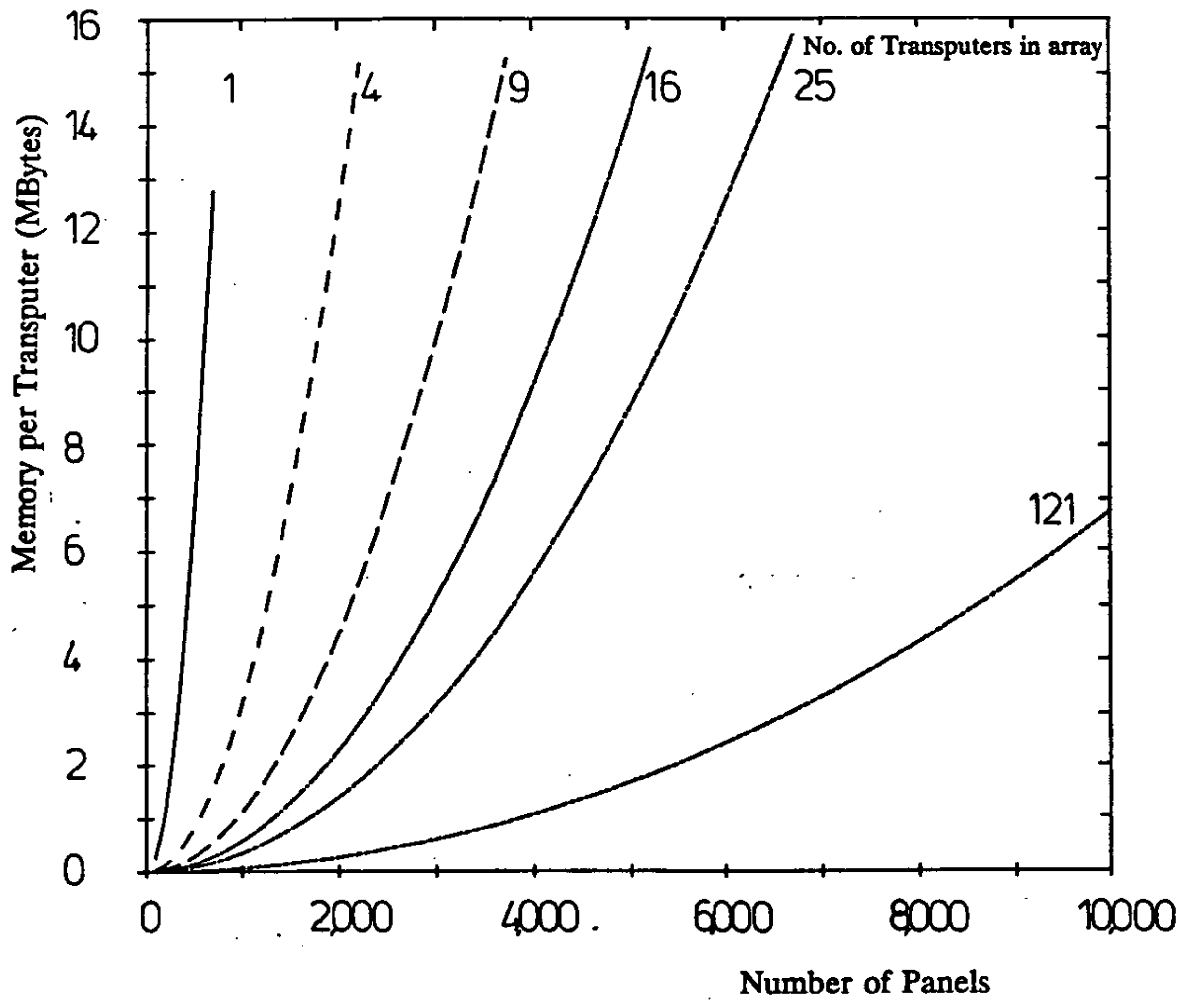


Figure 4 Individual transputer memory requirement for a given number of surface panels

where each real number requires 4 bytes and all the memory requirements but that for the panel and dipole store arrays are ignored. This provides a good estimate of the number of panels which can be solved on a given transputer array. The different lines represent various sized arrays of transputers. For instance the Ship Science Transputer System, of dimension N=2 and 1 Mbyte of memory per transputer, could solve a 400 panel problem. A machine with 8 Mbyte per transputer could solve a 10,000 panel problem if it were of dimension 11. A surface panel problem containing this number of panels is the order of the largest scale of lifting-surface problems currently solved on supercomputers. The Meiko Computing Surface available for use has 8MByte per transputer and with a sixteen transputer array could solve a 3800 panel problem.

The memory requirement for the dipole matrix emphasises the need to keep the memory demand of the lifting surface code to a minimum even at the expense of additional numeric calculations.

## 5.2 Strategy

The efficiency of the dipole influence matrix and **RHS** vector setting up process has to be as high as possible to ensure satisfactory operation of the lifting surface program on large dimension arrays solving large-scale panel problems. This means the strategy chosen has to keep the volume and amount of data communication to a minimum and ensure that every guest process always has information to process.

The even distribution of panels amongst the transputers helps to minimise communication. Each transputer has to calculate the self-influence of its own panels and their corresponding centroids. To calculate the influence of panels located on other transputers to its own panel centroids it is more sensible to communicate the panel centroid to the other transputers (three real numbers) rather than to receive the other transputers panel vertices (twelve real numbers).

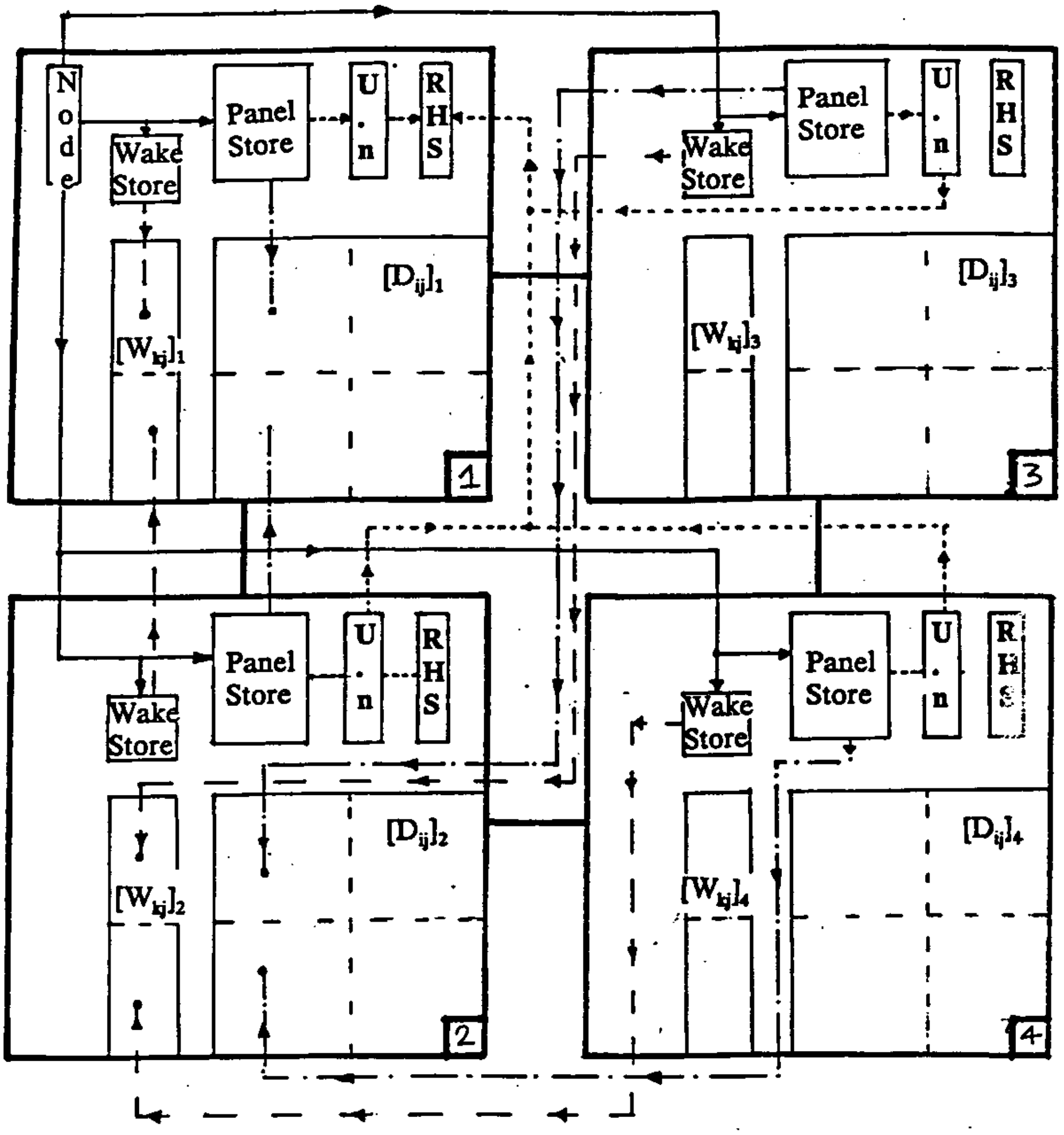
For a particular centroid it is necessary to calculate the source and dipole influence coefficients of every panel and of every wake strip. This is ideal for a parallel algorithm. Simultaneously, the setting up process is:
(1) calculating panel influence coefficients;
(2) calculating wake strip influence coefficients;
(3) receiving panel centroids;
(4) communicating the resultant matrix and vector elements to their final store destination.

The multiplicity of parallel tasks should ensure that each guest process is has minimal delays while waiting to carry out a numerical operation.

Each panel and corresponding centroid has a unique identity number related to the guest transputer on which it is stored. This allows an index address system to be used to ensure the result elements are returned to their correct destination.

As the communications harness performs better for larger messages, wherever possible the size of each message should be as close as possible to the maximum. Therefore, rather than a single centroid a packet of centroids is sent around the network. Each transputer on receiving the packet processes it with respect to its own panels and wake strip. For every panel located on a given transputer a packet of dipole and source influence coefficients are produced. The dipole packet is returned to the correct destination within the dipole matrix. The source packet elements are multiplied by the relevant source strength found from the boundary condition vector elements [**U.n**] and the result packet sent to the transputer on which part of the **RHS** vector is stored. Similarly, the packet of influence coefficients due to each wake strip is sent through to its destination in the WakePak Store.

Figure 5 illustrates the strategy for data flow through a network of four transputers for a packet of centroids originating from transputer 1. It should be noted that vectors are sub-divided between all transputers, each transputer having (N/T) elements and they are ordered in the same manner as the absolute panel identity numbers. The total amount of communication is related to the number of panels per transputer and the dimension of the transputer array. Increasing the dimension of a transputer array for a fixed total number of panels increases the amount of communication.



| | | | |
|---|---|---|---|
| Dipole | —·—·— | Source | - - - - - |
| Wake | — — — | Nodes | ——— |

Figure 5 Data flow schematic for Calculate Influence coefficient process on a four transputer array

## 5.3 Structure

A number of processes are required to allow all the tasks to be carried out in parallel. Data flows in and out of the setting-up process through the harness and onto the rest of the transputer network. Also, information is passed in and out of the various live-memory stores and between the various numeric processes. As an Occam2 channel is a one-way link joining two parallel processes, the in and out channels of the harness and stores can only be connected into one of the parallel processes. Multiplexing and de-multiplexing constructs are therefore used to ensure, for example, that the centroid packets arriving at a transputer are not confused with a packet of dipole influence coefficients. Tags attached to each message packet allow this routeing requirement to be achieved.

At the outer level the setting-up process CalculateIC runs in parallel with the various live-memory stores and Harness:

```
PRI PAR
  Harness()
  --
  PAR
     Matrix32()
     PanelStore()
     WakeStore()
     SurfVect()
     WakeIC()
     CalculateIC()
```

with two channels connecting all the stores and Harness with CalculateIC. PanelStore is connected by 4 channels to allow centroids to be generated and panel influence coefficients to be processed. Figure 6 is a block data flow diagram showing how the various sub-processes of calculateIC are connected to each other and the stores and harness. The different types of line are used to represent the various data packets. The CalculateIC process is sub-divided in the following manner within a WHILE loop:

```
SEQ
 finito:=FALSE
 WHILE finito=FALSE
  PAR
   ... GenerateCentroids
   ... etc,etc
   --
   ... finish
```

The correct termination of CalculateIC via the finish sub-process uses the fact that a fixed number of each of the types of data packets will be sent for a given total number of panels and array dimension. Cumulative tallies are kept of the data elements arriving to be stored and similarly the data packets generated. This allows each of the sub-processes to successfully terminate and hence allows the termination of the overall process.

The various processes which control the influence coefficient element packets act as a buffer for passing the information into the live-memory stores. This clears the receive

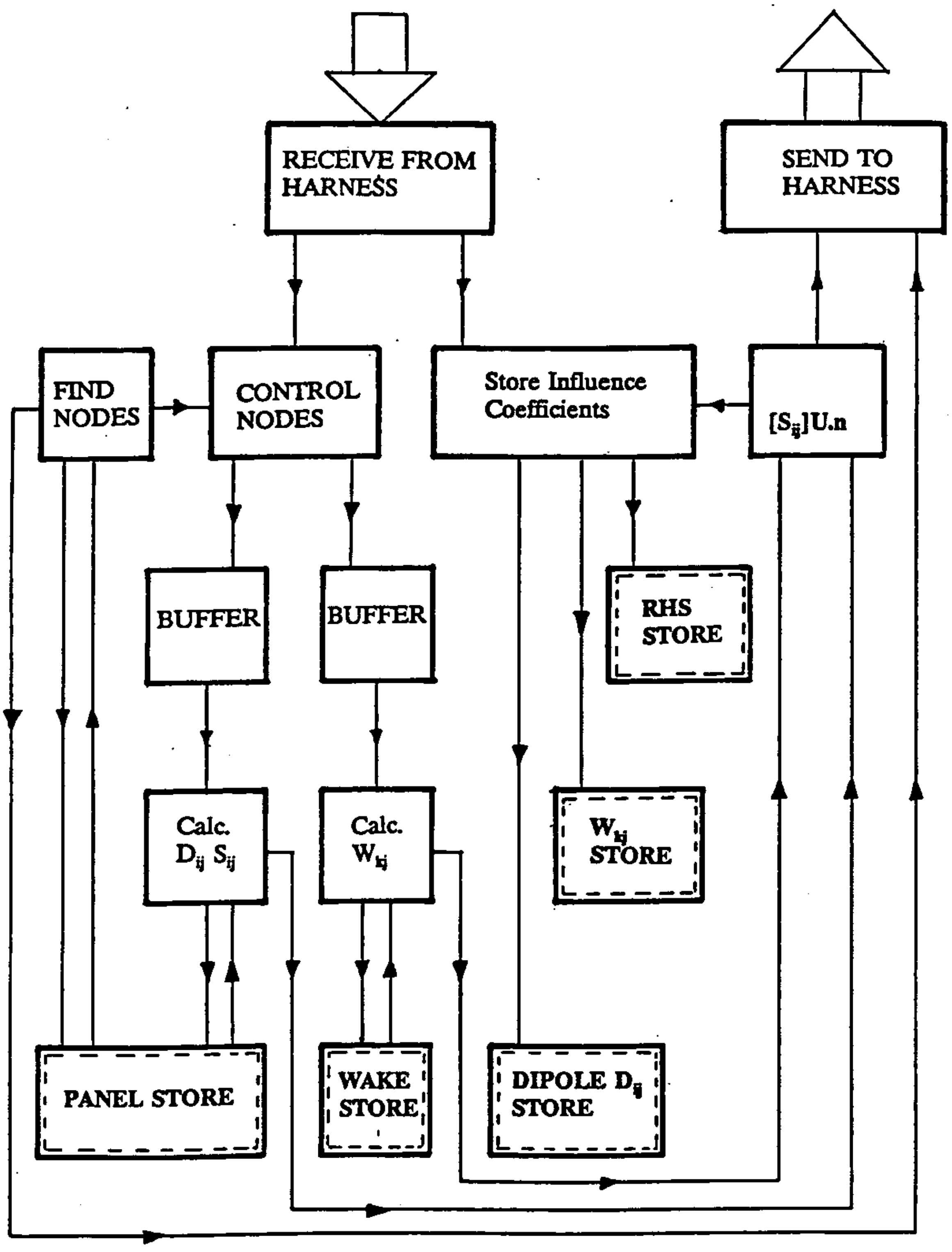


Figure 6 Internal structure of the calculate influence coefficient process

harness process to allow further input. The operation of the actual influence coefficient calculation processes are detailed in Sections 5.4 to 5.6.

## 5.4 Source and Dipole Influence Coefficients

The Newman panel influence coefficient procedure described in Chapter 4 calculates both the source and dipole influence coefficients. As input the procedure requires the test point and the panel vertices and geometric coefficients. To minimise data storage the panel geometric coefficients are recalculated every time a panel is used. By processing a package of centroids with each panel the number of times the geometric coefficients are recalculated is minimised.

For bodies with a plane of symmetry in the free-stream direction a reflection plane can be used to either halve the number of panels or double the panel density by using the same number of panels. An example of this is flow around a finite aspect-ratio wing at incidence where flow about the mid-span is symmetrical. The location of the reflection plane is specified in the geometry input file. The influence of panels on the reflected body half on the actual panel centroid is accounted for by reflecting the panel vertices about the reflection plane and using the image panel with the centroid in the newman panel routine. The symmetry condition means that a panel and its image will have the same value of surface potential and their influences can thus be summed.

An open-water propeller has rotational flow symmetry of order equal to the number of blades. Therefore, a multi-bladed propeller can be modelled by considering the flow around a single blade if the influence of all the image blade panels are calculated for the centroids of the panels on a single blade. Vector algebra allows all of the image panel vertices to be generated from the original panel vertices. The axis and origin of rotation are specified in the geometry input file. The reflection and rotational symmetry are also applied to the dipole panels defining the trailing wake sheet.

Within panel store on receipt of a packet of centroids for each panel the source and dipole influence coefficient are calculated for every centroid. If a reflection plane exists, using the reflected panel the additional image influence coefficients are added. Similarly if it is a rotationally symmetric body all the blade image panel influences are added. Two packets of source and dipole influence coefficients are generated for each panel. On calculating the sum of the actual, reflected and rotated influence panel for each centroid the two packets are sent out. Each centroid in a packet generates (2*N/T) influence coefficient packets.

## 5.5 Wake Strip Influence Coefficient

Each wake strip consists of a finite number of dipole panels all with the same strength. The total dipole influence coefficient at a particular centroid j is the sum of the influence coefficients of all the panels in the strip k, that is:

If required an expression for the additional influence of the wake sheet beyond its finite length to infinity $W_{\infty}$ can be included at this stage. In the same way as for the body

$$W_{kj} = \sum_{l=1}^{N} W_{lj} + W_{\infty\ kj} \qquad [2]$$

panels, reflected and rotated panel images are used to generate the final influence coefficient of each wake strip at a particular centroid. Again, each panel is used with all the centroids in a packet to save on unnecessary panel geometry recalculation.

## 5.6 Right-hand Side Vector

The Right-Hand-Side **RHS** vector is the matrix-vector product of the source influence coefficient matrix [Sij] and the surface boundary condition vector **U.n**. By calculating the RHS vector as part of CalculateIC it removes the need to store the complete source influence matrix. The saving in memory more than compensates for the additional programming complexity.

The solid surface boundary condition for a panel source strength $\sigma$ is:

$$\sigma = - U_i \cdot n \qquad [3]$$

where $\mathbf{U}_i$ is the inflow velocity at the panel centroid and **n** the panel unit surface normal. The inflow velocity is the sum of the specified velocity field $\mathbf{U}_s$ in the absence of the lifting-surface and any body rotational velocity. That is:

$$U_i = U_s + r \times \omega \qquad [4]$$

where **r** is the distance from the panel centroid to the origin of the axis of rotation and $\boldsymbol{\omega}$ the body rotation vector. The source strength at every panel is calculated during the lifting-surface definition process and stored in a vector store.

To obtain the final matrix-vector product each influence coefficient element has to be multiplied by the corresponding panel source strength. Each element of the RHS vector is the sum of the product of source strength and influence coefficient for a particular centroid.

$$RHS_i = \sum_{j=1}^{N} S_{ij} \cdot \sigma_j \qquad [5]$$

## 5.7 Performance

For a given number of surface and wake panels on a fixed array of transputers the total amount of communication remains constant. Therefore, the least efficient operation of the CalculateIC process occurs when there is a minimum of numeric processing per panel to be carried out. In Figure 7, which plots the number of panels distributed across four transputers

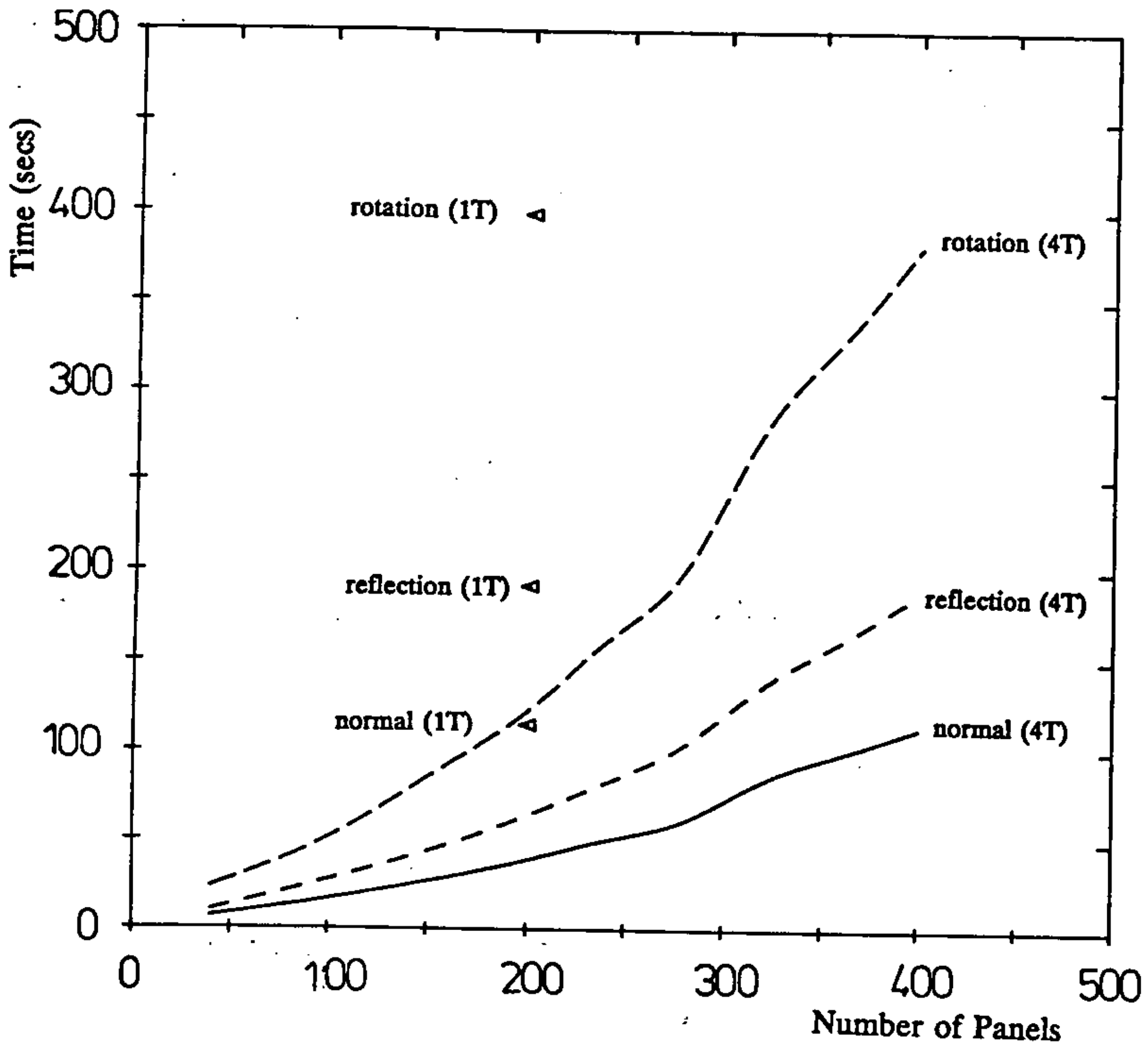


Figure 7 Performance of calculate influence coefficient on a four transputer array

against the time to calculate all the influence coefficients, lines are shown for three test cases of a rudder without reflection plane, a rudder with reflection plane and a four-bladed propeller. As the amount of numeric processing increases as the number of image panels increases the overall time increases. However, if the time to run a 200 panel problem on four transputers and one transputer is compared speed-ups of 2.96 ($\eta_c$=73.9%), 2.99 ($\eta_c$=74.8%) and 3.28 ($\eta_c$=82%) are obtained for the three cases which demonstrates the increased efficiency with additional numeric processing per panel. The relatively small increase in efficiency is due to the additional operations to reflect/rotate panels and generate their geometry coefficients.

Overall the calculate influence coefficient process exhibits a quadratic increase in processing time with the number of panels. The knee in the curve occurs when the panel centroid is sent as two packets rather than one which results in extra processing as each panel geometry data is generated twice as many times.

## 6 Solution of Simultaneous Linear Equations

### 6.1 Introduction

Efficient solution of the linear system of equations generated by the Calculate influence process is essential. The basic system to be solved is:

$$[\,M\,]\,\underline{\phi} = \underline{R} \qquad [6]$$

where vector **R** is known , matrix [M] is dense but diagonally dominant, and vector $\phi$ is unknown.

There are large numbers of techniques for solving linear systems of equations on sequential computers. Also, as it is a technique of general application to many numerical problems, parallel algorithms are being developed. For instance, Rua[4] investigated the similar problem of solving for eigenvectors on an array of transputers using a linear chain topology. An aim of this work has been to investigate the use of transputers for solving general C.F.D. type problems. Therefore, an investigation was carried out into methods for solving systems of linear equations when the matrix data is distributed across a square array of transputers.

Several methods both iterative and direct were implemented and performance comparisons have been made. To aid in the implementation a series of basic matrix-matrix and matrix-vector processes were developed.

### 6.2 Matrix Procedures

By developing processes which use a Matrix live-memory store to carry out basic matrix operations such as addition, subtraction and multiplication more complex processes can be rapidly written as combinations of the basic operations. The live-memory store aids in reducing development time by allowing easy access to individual elements, rows and columns. Also, initial values for the matrix such as the identity or zero matrix can be carried out within the store.

Matrix addition and subtraction operations are achieved by linking three parallel processes:

```
PAR
  Matrix32(Msize,Ain,Aout)
  --
  Matrix32(Msize,Bin,Bout)
  --
  MatrixAdd(Msize,Ain,Aout,Bin,Bout)
```

The input/output channels (*Ain,Aout,Bin,Bout*) of the two Matrix32 store processes are connected to the MatrixAdd process. This process, for every element in the sub-matrices of size Msize, obtains corresponding elements from [A] and [B] matrices, adds them, and then stores the value in Matrix [A]. No communication is required between guest transputers and all the host process need do is to start and then acknowledge the finish of all the guest processes. An identical technique is used for such things as matrix subtraction[7], equality[8], and scalar multiplication[9]:

$$[A] = [A] - [B] \tag{7}$$

$$[A] = [B] \tag{8}$$

$$[A] = k \times [B] \tag{9}$$

where k is a scalar constant. For all the above operations the guest processes are independent of each other and the code efficiency will be solely affected by the start and acknowledge finish communication across the whole network. Figure 8 shows the time to carry out a matrix add operation against a base of overall matrix dimension Msize for a four transputer array. The process is rapid and exhibits a quadratic behaviour as the number of operations increases as a square of the matrix dimension. Figure 8 also shows the time to carry out the other matrix operations described later.

Matrix multiplication and obtaining the transpose of a matrix:

$$[A] = [B]^T \tag{10}$$

$$[A] = [A][B] \tag{11}$$

requires the interaction of individual elements with many others and hence inter-guest process communication is needed. To achieve maximum efficiency a minimum amount of communication is required and no guest process should ever be waiting for information to process.

The transposing of a matrix involves the swopping of rows and columns. The process is schematically shown in Figure 9 with matrices [A] and [B] sub-divided across 4 transputers. For instance, row 10 of matrix [B] becomes column 10 of matrix [A]. In the example of the 4 transputer network, the diagonal transputers T1 and T3 simply take rows from [A] and put them in the corresponding column where as the off-diagonal transputers T2 and T4 swop rows and then store as columns. The parallel algorithm is written as:

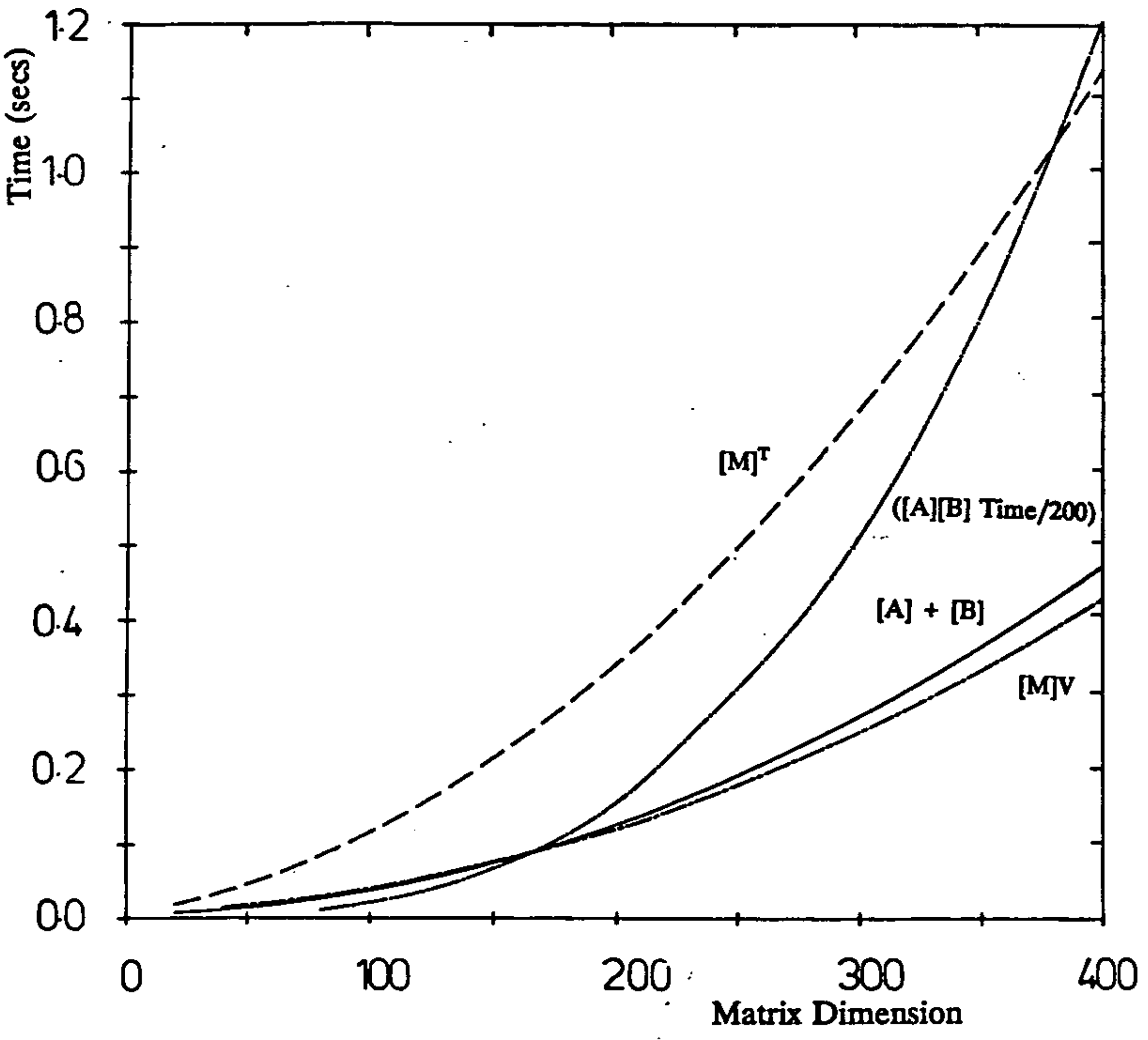


Figure 8 Time to carry out basic matrix operations on a four transputer array

```
PAR
  ... SendRow
  ... ReceiveColumn
  ... GetRows(Bin,Bout)
  ... StoreColumns(Ain,Aout)
```

where if the transputer is on the main diagonal each retrieved row is sent directly to the StoreColumn process otherwise it is passed across the harness through the SendRow process. At its destination it is received by ReceiveRow and then passed to StoreColumn. A matrix transpose simply involves communication. Figure 8 shows the time to carry out a transpose against a base of overall matrix size for a four transputer array, again a quadratic behaviour is seen.

Matrix-vector multiplication
requires the summation of the multiplication of each element of a matrix row with the corresponding element of the vector. That is:

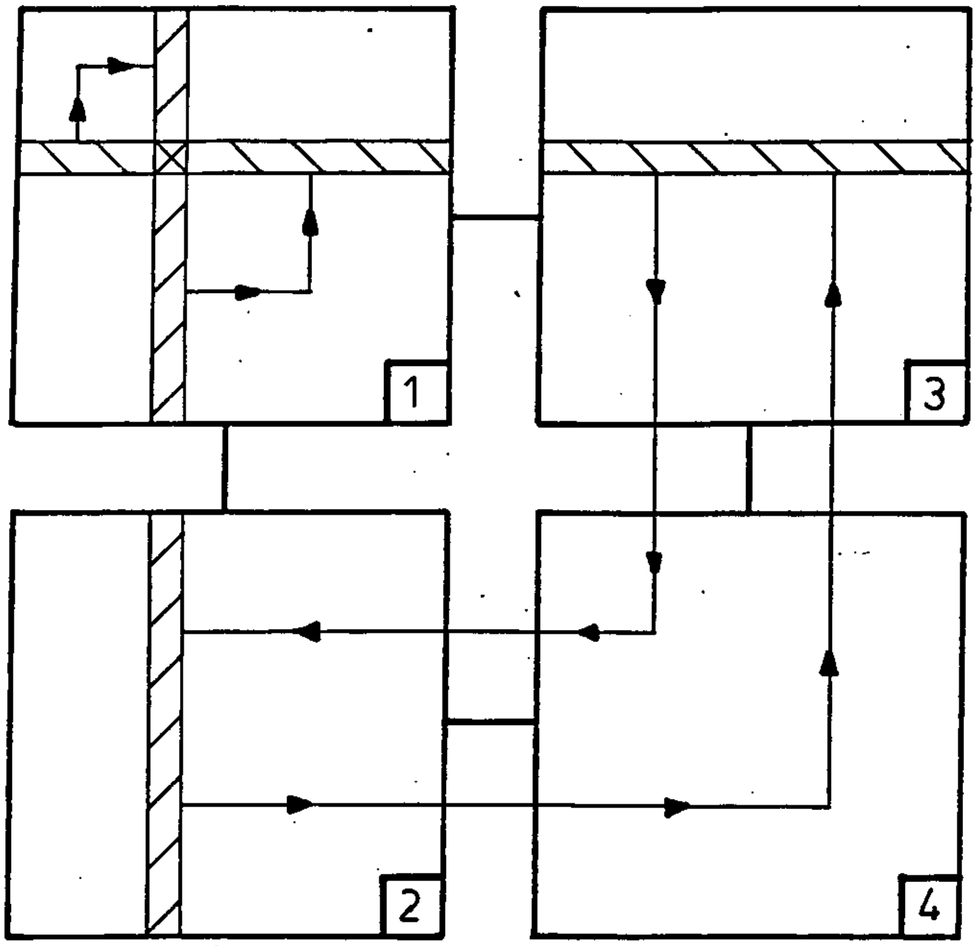


Figure 9 Schematic of matrix transposition on a four transputer array

$$Q_i = \sum_{j=1}^{Msize} M_{ji} \cdot V_j \qquad [13]$$

A vector is stored as a column through the transputer network, and as shown is schematically in Figure 10. For a 4 transputer network each guest process sends its part of the vector to all the transputers in its column in the array. This sub-row is summed with the corresponding elements in each of the rows and then the part answer vector sent back to its correct location. Each transputer carries out the same amount of communication which is proportional to the dimension of the transputer array. Figure 8 also gives the time to carry out matrix-vector multiplication for different sized matrices. It can be seen that the performance is similar to the matrix add operation; the number of operations being equivalent and the communication low.

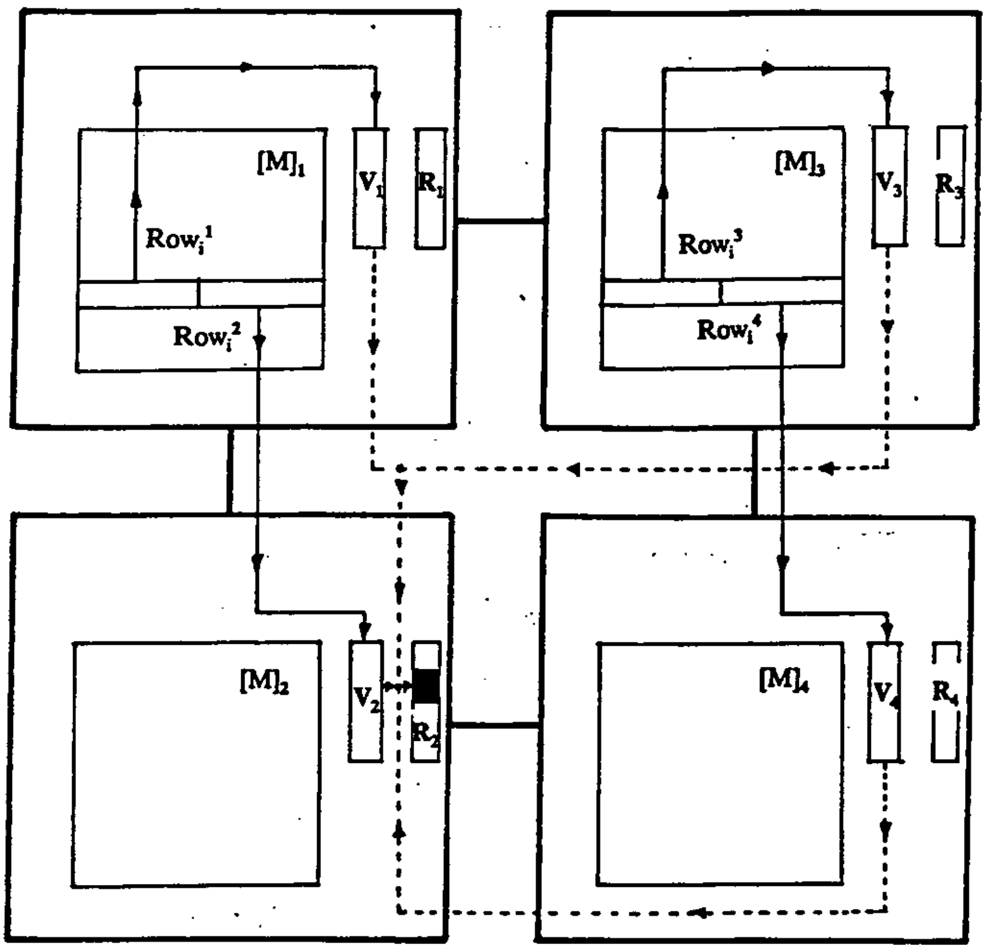


Figure 10 Schematic of matrix-vector multiplication on four transputers

Although matrix-matrix multiplication is not required as part of a solution procedure it is none the less an interesting test for a parallel computer. As every element in each matrix has to be multiplied together it requires a large amount of communication for a bare minimum of numeric processing. The approach chosen was to first transpose one matrix. Then every column in one matrix is passed horizontally through the array in a similar manner to that used in the matrix-vector multiplication and the result vectors are sent to their final destination and summed to give the result matrix. The time to carry out this operation, as also shown in Figure 8 (to a scale of time(secs)/200), is about 2/3rds of the time taken to transpose a matrix and then carry out N matrix-vector multiplications. The difference is due to the reduced amount of control commands from the Host transputer. Overall, matrix-matrix multiplication exhibits a cubic behaviour.

## 6.3 Iterative versus Direct methods

The choice of technique for solving a system of linear equations revolves around whether an iterative approach can converge more rapidly to solution than a direct method takes to explicitly solve the whole system. In general a dense full matrix, such as that generated by a lifting-surface, is more likely to favour an iterative approach. This is especially true because of the leading diagonal dominance of the system. This dominance arises from the self-influence of each dipole panel.

How the various algorithms are made to run in parallel and whether this influences the choice between iterative and direct methods needed to be investigated. A range of iterative methods were developed and their performance compared with a direct matrix inversion using Gaussian elimination.

6.4 Gaussian Elimination - Direct Inversion

The inverse of a matrix is defined such that the product of the matrix and its inverse gives the identity matrix

$$[\,I\,] = [\,M\,]^{-1}\,[\,M\,] \qquad [14]$$

If both sides of equation [6] are multiplied by $[M]^{-1}$ to give [15].

$$[\,M\,]^{-1}\,[\,M\,]\,\underline{\phi} = [\,M\,]^{-1}\,\underline{R} \qquad [15]$$

$\phi$ is obtained directly as a matrix-vector product shown in [16].

$$\underline{\phi} = [\,M\,]^{-1}\,\underline{R} \qquad [16]$$

A linear system of equations can be solved by first creating the lower diagonal form of [M] and by then progressive back-substitution to successively determine each element of $\phi$. However, this process has to be repeated every time the right hand side vector **R** is changed. This happens when the iterative Kutta condition is applied. Therefore, it was decided to concentrate on generating the complete inverse and the application of the resultant Kutta condition to vector **R** then requires only a single matrix-vector multiplication.

Gaussian elimination operates by successively carrying out row operations on both the original and an identity matrix to remove elements from the original. Comprehensive treatment of the method is given in textbooks, for example Kreysizg[3]. At first sight Gaussian elimination is not ideal for a parallel algorithm as the method works by progressively eliminating columns of elements.

The elements in a column are eliminated about a pivotal element. For a given pivot element $a_i$ of row j the following two-stage process is carried out, where r is a matrix element:

```
(1)     SEQ k = 1 FOR N
            $r_{jk} := r_{jk} / a_i$
```

and where $b_i$ is the element of a row in the same column as the pivot element $a_i$

```
(2)   SEQ j = 1 FOR N
        IF
          i=j
            SKIP
          TRUE
            SEQ k = 1 FOR N
              $r_{ik} = r_{ik} - ( b_i r_{jk} )$
```

The result of the process is that all the elements of the pivot column are set to 0 apart from the pivot element which is unity. Both stages are carried out on the identity matrix using element $a_i$ and $b_i$ from the original matrix. The column elimination process is repeated for all columns to generate the inverse matrix. The choice of the individual pivot element can influence the accuracy of solution especially for large matrices. Also, the number of element operations is of the order $(N^3)$ and so the time to solution increases rapidly as the matrix size increases.

The most accurate method of choosing the pivot element is so called maximum pivoting where at each stage the maximum (absolute size) element of the original matrix is chosen and used as a pivot. However, for diagonally dominant matrices, where the diagonals are of the same magnitude, the best choice is simply to use the main diagonal as the pivot elements. Main diagonal pivoting was used for the lifting-surface system but maximum pivoting was successfully implemented by adding a stage to obtain the maximum element on each guest pivot. This information was collected by the host process and the position of the overall maximum element broadcast back to the guests.

A sequential structure was used for the guest process as a series of N loops for pivoting on each column. For each loop the host process broadcasts the location of the pivot. Each guest then carries out steps dependent on where it is relative to the pivot. This is best illustrated in Figure 11 for a 16 transputer array. For a pivot on transputer T6 the steps that each transputer carries out are as follows:

| | |
|---|---|
| T6: | Divides Pivot row by Pivot element |
| | Sends Pivot column left & right |
| | Sends Pivot row up & down |
| T2,T10,T14: | Receive Pivot Column |
| | Divide row by Pivot Element |
| | Send Pivot Row up and down |
| | Carry out row operations |
| T5,T7,T8: | Send Pivot Column left and right |
| | Receive Pivot row |
| | Send Pivot row |
| | Carry out Row Operations |
| T1,T3,T9,T11, | Receive Pivot Row |
| T12,T13,T15,T16: | Receive Pivot Column |
| | Carry out Row Operations |

As rapidly as possible each guest process passes out information required by other processes. This minimises the delay while processes await information. The process is only shown for the original matrix but the pivot rows and columns are also passed for the eventual inverse matrix.

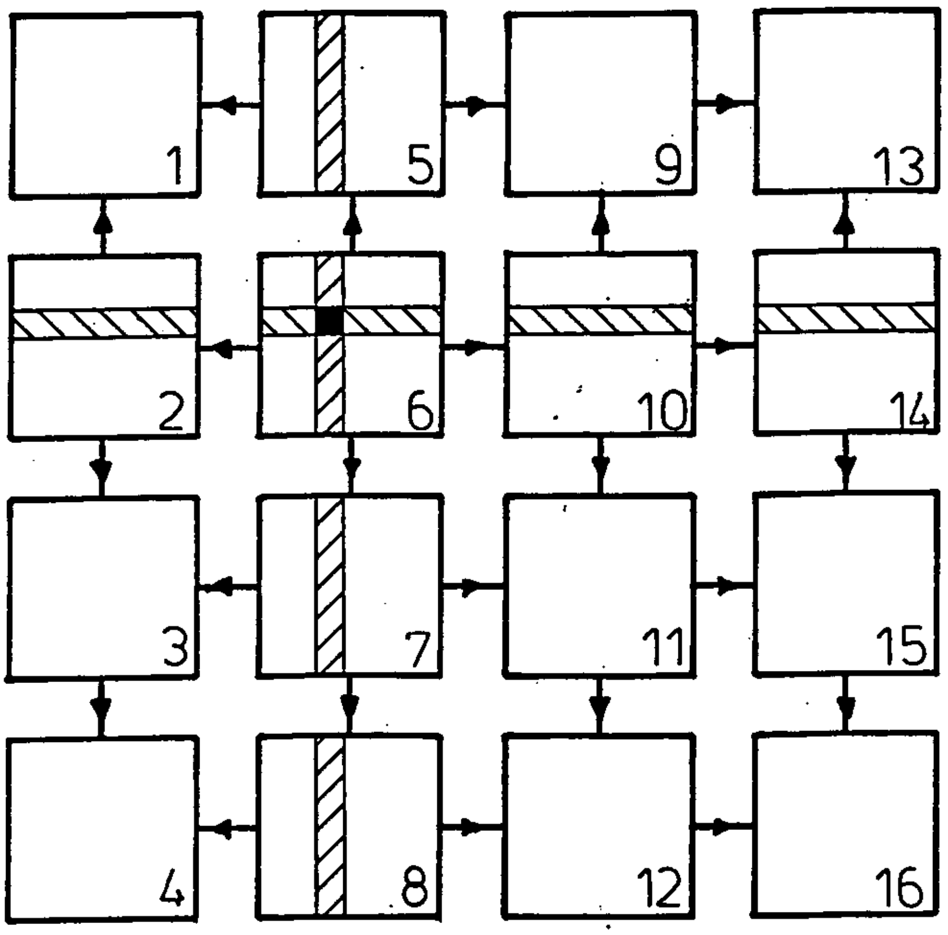


Figure 11 Schematic of Gaussian elimination on a sixteen transputer array

Two matrix stores are used for the inversion. At the completion of the process the original matrix is the identity matrix and the other is the required inverse of the original.

6.5 Jacobi-Iterative Scheme

An iterative scheme uses an initial guess for the unknown vector $\phi$ to generate a better approximation to the solution. The process is repeated until the solution has converged to a given level of accuracy. The simplest possible method for diagonally dominant matrices is to use the main diagonal elements to calculate the correction. This process is known as the Jacobi method.

The Jacobi correction $\Delta\phi$to the original estimate of vector $\phi$ can be written as:

$$R^* - [\,M\,]\,\phi^{k-1} \qquad [17]$$

$$\Delta\phi_{jj} = \frac{1}{M_{jj}}\left( R_j - R^*_j \right) \qquad [18]$$

where $M_{jj}$ is the jth leading diagonal element. The kth approximation is then becomes:

$$\underline{\phi}^k = \underline{\Delta\phi}^k + \underline{\phi}^{k-1} \qquad [19]$$

If each guest process is sent the relevant part of the matrix main diagonal, then the Jacobi scheme can be implemented using standard matrix procedures:-

```
SEQ
  Matrix_Vector_Multiply (M,$\phi$,R')
  Vector_Subtract(R,R', Ans)
  Vector_Divide(Ans, Mjj, $\Delta\phi$)
  Vector_Add($\Delta\phi$, $\phi$)
```

A convergence check is carried out on correction vector $\Delta\phi$. The maximum absolute value is sent to the host process which either allows the guest process to proceed or, if all the maxima are below a cut-off criteria, finishes the calculation. The cut-off criteria used was that the absolute biggest change in an element is less than $1\times10^{-5}$.

The matrix-vector multiply process is the only one which requires communication during each iteration. As a result each iteration is fast. However, as only one element is used in the updating process a large number of iterations are required. An advantage of the scheme is that all the processes are working all the time in generating the next iteration. The Gauss-Seidel scheme uses the same technique for updating, only each element update is carried out sequentially, based on using the latest values for each element. The interdependence for element updating means that a parallel single-element Gauss-Seidel method is inefficient.

For the Jacobi scheme the initial approximation is usually taken as the zero vector. However, for the lifting-surface method with the iterative Kutta condition it is better to use the final solution for the next series of iterations as most values of potential only change by a small amount.

## 6.6 Single-Block Iterative Scheme

The drawback of the Jacobi scheme of using only one element in the updating process suggests the use of a block of information from the original matrix. A block size equal to the number of panels on each guest transputer as illustrated in Figure 12 can be easily accommodated.

The first stage is to transfer the leading diagonal matrices to the guest process where their self-influence panels are stored. These sub-matrices are then directly inverted using Gaussian elimination. The iterative process can then commence. It is identical to the Jacobi

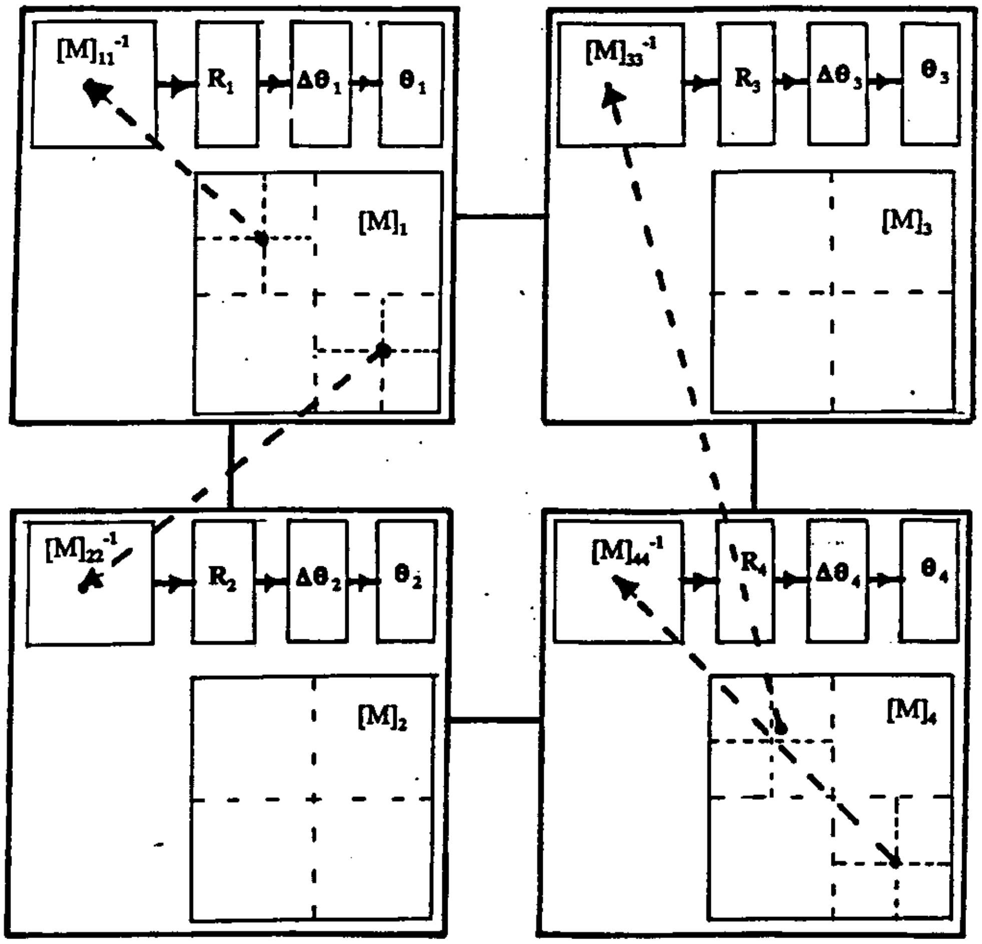


Figure 12 Schematic of Single block iterative scheme on four transputers

except that instead of the Vector Divide stage the sub-matrix inverse is used to multiply the vector difference **R-R'**. That is:

$$\underline{\Delta\phi}^k_T = [\,M\,]_{TT}^{-1}\left(\underline{R}_T - \left([\,M\,]\underline{\phi}^{k-1}\right)_T\right) \qquad [20]$$

where T is the transputer number and $[M]_{TT}^{-1}$ the sub-matrix inverse. The implementation of the scheme is identical in all other respects to the Jacobi. A drawback to this scheme is that if the number of panels per transputer gets too large the inversion of the sub-matrix will take a long time. As in the case of the Jacobi scheme for each iteration the only communication is during the matrix-vector multiply stage. Also, once the sub-matrices have been passed to their destination, the direct inversion is an independent process.

## 6.7 Multi-Block Jacobi Scheme

To restrict the maximum block size the dimension of the sub-matrices was made variable between 1 (Jacobi - single element) and that for 1 block per transputer (single-block). This is also shown schematically in Figure 12. The implementation is the same as for the single-block scheme except that each guest process has a number of blocks sent to it. Each block is then inverted and used in the iteration process to update its portion of $\Delta\phi$ vector. The ability to change the block size allows the iterative procedure to be tuned to obtain minimum iteration time for a given lifting-surface problem.

## 6.8 Comparative Performance

Figure 13 shows the time in seconds to solve a system of linear equations against the size of the system. Plots are shown for the four schemes previously described and also the time to set up the influence coefficient matrix. The data given is for a four-transputer system.

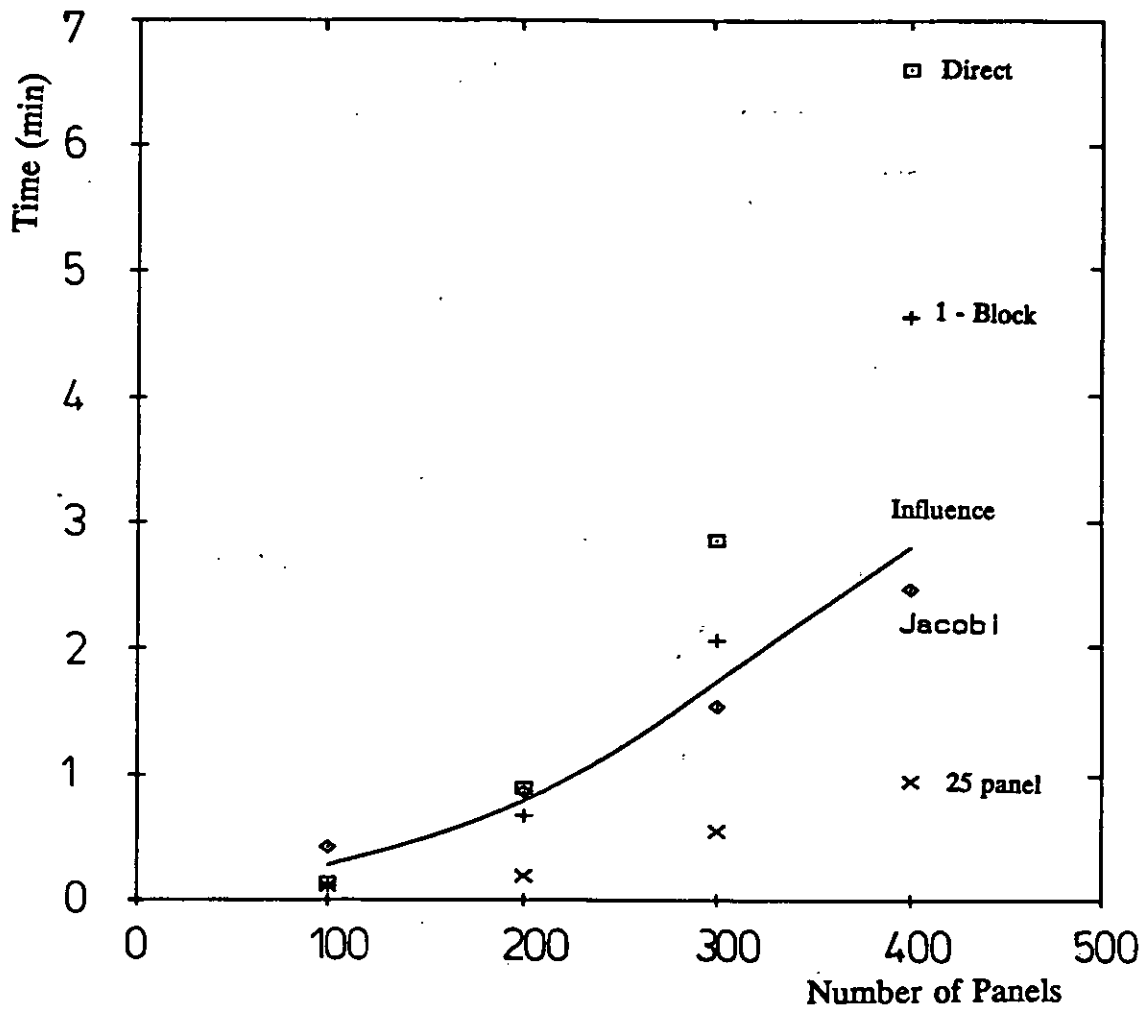


Figure 13 Performance of matrix solution procedures on a four transputer array

The $O(N^3)$ behaviour of the direct inversion scheme can be clearly seen. Also, exhibiting a similar trend is the single-block Jacobi scheme as the individual block size increases. The single element Jacobi performs much better than the direct scheme for matrix dimensions greater than 220. This is to be expected as the scheme converges in O(N) iterations and the time for each iteration has a fairly constant relationship to the total problem size. For the particular problem in question the Jacobi scheme takes about the same time as the influence coefficient process. The multi-block scheme used a constant block size of 25 panels and for the 400 panel problem was 7.2 times faster than the direct method and 2.7 times faster than the Jacobi method. It is worth noting that for this geometry the multi-block scheme is significantly faster than the influence coefficient set up process.

The reason for the choice of block size of 25 panels for the multi-block can be seen in Figure 14. This plots convergence time against block size for the 400 panel problem. A pronounced minimum is found for a block size of 25. This corresponded to the number of panels in each chordwise strip for the problem which was held constant for producing Figure 5.13. This is not entirely unexpected as the panel potential strengths will generally be dominated by other panels in the same chordwise strip and so only using the influence of these panels in updating should give the quickest convergence. Clark[5] also found this was the best block size for his accelerated convergence scheme. This scheme was used by Kerwin & Lee[6]. However, at present it has not been investigated as it contains substantial sequential processing components and requires extra memory storage.

## 7 Kutta Condition

The iterative Kutta condition was implemented in a straight forward manner. For the first iteration the Morino condition that $\Delta\phi$ was equal to the difference in potential at the trailing edge was used. After the potential vector $\phi$ had been solved either by a direct or iterative method, the pressure loading $\Delta Cp$ and the rate of change of pressure loading with wake sheet strength $d(\Delta Cp)/d(\Delta\phi)$ were found for all the wake strips on each guest process. These values give a corrected wake strength $\Delta\phi'$ for each wake strip. Multiplying by the wake strip influence matrix $[W_{jk}]$ gives a correction vector to be applied to the right hand side vector:

$$\underline{R}' = \underline{R} + [W_{jk}]\,\underline{\Delta\phi}' \qquad [21]$$

A vector is used to store the wake strength $\Delta\phi$ as :

$$\underline{\Delta\phi} = \left(\underline{\phi}_u - \underline{\phi}_l\right)^k \qquad [22]$$

where $\phi_u$ and $\phi_l$ are the trailing-edge panels potentials. The process is repeated until $\Delta Cp$ has disappeared within a given limit. The total wake strength correction is:

$$\underline{\Delta\phi} = \sum_{k=1}^{m} \underline{\Delta\phi}'_k \qquad [23]$$

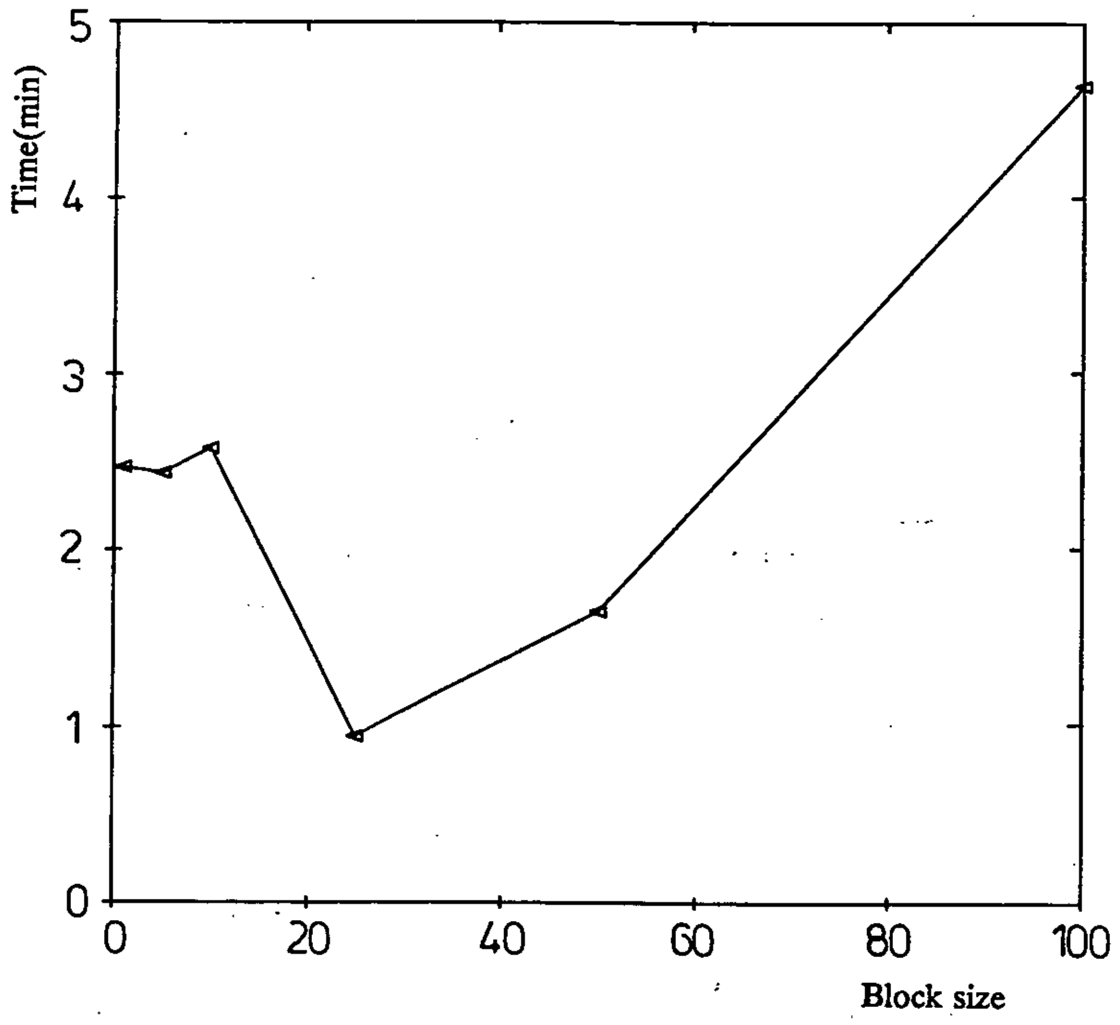


Figure 14 Variation in convergence time with block size for 400 panel problem

where m is the number of iterations to convergence.

The calculation of vector **R'** is rapid and performance figures are not given.

For the direct method of solution each Kutta condition iteration requires only a matrix-vector multiplication.

$$\underline{\phi}^{k} = [\, M \,]^{-1} \, \underline{R}'^{k} \qquad [24]$$

Whereas, for the iterative solution methods the number of iterations to convergence generally halves after each Kutta condition iteration. As noted previously, the initial approximation is taken to be the previous final solution for all but k=0.

**8 Calculation of Velocity Field**

In modelling the interaction of a ship rudder and propeller a requirement is the development of velocity field both up and downstream from a lifting surface. Differentiation of the expression for the Newman dipole and source panel influence coefficients gives the disturbance velocity at a point in space due to a unit strength panel. Once the values for the individual dipole panel strengths $\phi$ and wake sheet $\Delta\phi$ have been solved the total disturbance velocity at a point due to the whole lifting-surface can be found.

The total velocity $U_T$ at a point is the vector sum of the disturbance velocity $U_d$ and the local inflow velocity $U_i$.

$$\underline{U}_T = \underline{U}_i + \underline{U}_d \qquad [25]$$

The velocity field within a volume of space can be found by determining the total velocity on a regular mesh of points within the volume of space. This information can be used for various post-processing tasks. For example, many three-dimensional flow visualisation techniques require a regular mesh of data points. Also, the interaction velocity field can be developed from such an arrangement.

A farm algorithm controlled by the Host process is ideal for carrying out the velocity field generation process. The user specifies the x, y, and z limits of the volume of space and the required number of points Nx, Ny, and Nz in each direction within that space. A packet of points is sent to all guest processes. Each guest calculates, for every point in the packet, the disturbance velocity field due to its body surface and wake sheet panels. Also, it finds the inflow velocity field. The two packets of velocity information are returned to the Host process. The total velocity at a point is the sum of all the guest disturbance velocity field plus the inflow velocity at that point:

$$\underline{U}_T = \sum_{i=1}^{T} \underline{U}_d + \underline{U}_i \qquad [26]$$

where T is the number of guest transputers. Packets are sent from the Host until the velocity has been found for all points defined by the user.

As it is a farm algorithm the performance of the velocity field process is proportional to the number of panels and, the number of points, and inversely proportional to the total number of guest transputers.

## 9 Adaptive Wake

The method described in Ref. [1] was straight-forward to implement using a parallel algorithm similar to that for calculating the influence coefficients. Each wake strip is divided into a near and far region. In the near region the panels are aligned with the local flow direction. For the far wake, the panels follow the direction of the most downstream free wake panel. The first stage of the algorithm requires the generation of the centroids of the free wake panels. The total velocity is then calculated at each centroid. Each packet of centroids is broadcast around the transputer array and the result packets returned to the originating Guest transputer. For the rotational case, the velocity is obtained in cylindrical coordinates, ($V_a$, $V_r$, and $\omega$). The velocity is stored for each panel node by averaging the values of velocity at the surrounding four panel centroids. This summing process requires an exchange of information between transputers as identical nodes are held on different transputers. Once the information has been exchanged the remaining two processes are independent. For each node the change in position is obtained using the magnitude of the total velocity and length between stream-wise nodes to determine the timestep $\Delta t$. The increments in node position are then applied, using the regression relationships of Ref. [1] , to set the new node position. For the rotational case the increments are applied in the cylindrical coordinate system and then transformed back into cartesian system for storage in the wake store.

The performance of the wake adaption process is the same as for the velocity field generation.

## 10 Performance of Implicit Algorithms on an Array of Transputers

### 10.1 Introduction

Implicit algorithms require communication and the amount of communication traffic depends on the number of transputers used to solve a given problem. The estimation of the performance of an implicit algorithm on different sized arrays of transputers requires expressions for the time to carry out the communication transfer as well as for the numeric calculation. For an explicit algorithm, as represented by the Euler solver described in Ref. [3], the ratio of calculation to communication is constant for a given number of nodes per transputer and hence performance can be scaled from timing measurements made on a small array of transputers. However, for an implicit algorithm, how the communication time scales with array size has to be known for an estimation process based on the performance of a small array of transputers.

As the performance of the Harness across an array of transputers is known an estimate can be made of the total communication time for any given array of transputers. In general, there will be one or more transputers in the network whose messages (to and from) have to travel over the number of links equal to the dimension of the array. This transputer will have the maximum data transmission time and determine the code efficiency.

This section firstly details the code efficiencies of the various processes which make up the lifting-surface panel method. An estimation procedure is developed using the code efficiency measurements and the performance of the communications harness so that the performance of the processes can be scaled for different sized arrays of transputers. Finally, the overall performance of the lifting-surface panel method is assessed and estimates made for its scaled performance.

### 10.2 Code Efficiency

For an implicit algorithm, code efficiency scales with the size of the transputer array. Figure 15 plots code efficiency against the total number of panels. These are code efficiencies for a 4 transputer array with a diameter of 2 links as compared to the performance obtained for an identical number of panels on a single guest transputer. The memory size of 1MByte per transputer on the Ship Science Transputer System only allowed performance on up to 200 panels to be assessed. Lines are shown for influence coefficient setting up, solution of linear system of equations(block Jacobi and Gaussian elimination), and velocity field calculation. Also shown are lines for the two fundamental processes: matrix transposition and matrix-vector multiplication. For all the processes $\eta_c$ generally increases with the number of panels. The velocity field calculation performance is constant as would be expected for a farm algorithm. The efficiency of 91% reflects the overheads associated with generating the test nodes on the Host transputer. Therefore, the efficiency of the wake adaption process would be expected to be higher.

The efficiency of the CalculateIC process is only 75% and reflects the inherent complexity of the process. It also suggests that potential improvements could be expected from further development of this process.

The Gaussian elimination process, for an implicit algorithm, shows a rapid increase in efficiency as the number of panels increase as does the Matrix-Vector and Block-Jacobi processes to a 70%-80% efficiency. Even the Transpose operation has a 50% efficiency for a 200 panel problem.

Overall, even for a relatively small panel problem (200) the code efficiency of the component processes of the lifting-surface panel analysis are all in the range of 70% to 90%. This efficiency would be expected to increase for larger sized panel problems.

## 10.3 Performance scaling

For a given problem with N body panels, NW wake panels, and NI images (reflection: NI=1, 4-bladed propeller: NI=3) the total number of calls to calculate the dipole/source influence of a panel at a given node will be:

$$N_{ic} = ( NI + 1 ) \, . \, N \, . \, ( N + NW ) \qquad [27]$$

Each transputer in an array of T transputers carries out ($N_{ic}/T$) of these calls. The efficiency of the influence coefficient process on T transputers is $\eta_{cic}$. The time to solve the resulting system of equations and to carry out the Kutta condition iteration can be assumed proportional to $N^2/T$. These assumptions gives the total time t to solve an N panel problem on T transputers as:

$$t = \left[ \frac{N_{ic}}{\eta_{cic} \, T} \right] t_{ic} + k \left[ \frac{N^2}{\eta_{cbj} \, T} \right] \qquad [28]$$

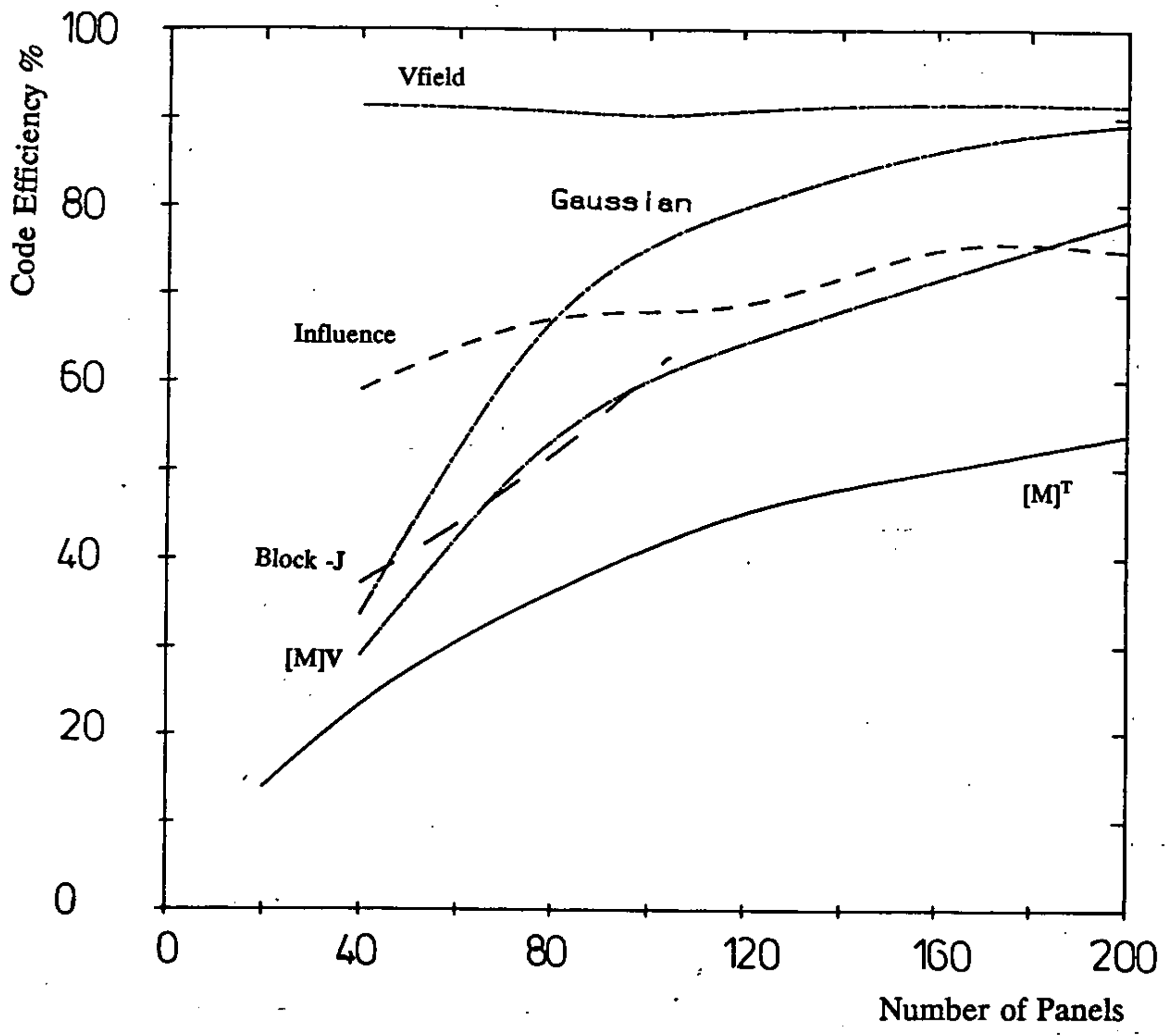


Figure 15 Code-efficiency of lifting-surface component processes on a four transputer array

where $t_{ic}$ is the average time to calculate the source/dipole influence coefficient using the Newman panel procedure (for a rudder problem an average of 18-20 ticks of the transputer clock). The block-Jacobi calculation has a fairly low communication requirement and its efficiency ($\eta_{cbj}$) will only be weakly degraded by increasing the transputer array size. However, the influence coefficient efficiency $\eta_{cic}$ will reduce as a function of $\sqrt{T}$, the mean message distance being $\sqrt{T}$-1 links. However $\eta_{cic}$ is likely to still increase with increasing total number of panels which will generally increase with the transputer array size. This increase will offset some of the degradation due to increased communication. If the Harness performance Equation[3.14] is used with $L=\sqrt{T}-1$, and b (no. of bytes) proportional to $(N/\sqrt{T})$ then the total communication time $t_c$ for $(N/\sqrt{T})$ messages becomes (expressed in transputer clock ticks for different transputer array sizes):

The above expressions indicate that, for constant N, $t_c$ reduces as T increases. The above expressions [28] and [29] show for a 400 panel problem an almost constant overall $\eta_c$ with

$$t_c = 825\ N + 2.2\ N^2 \qquad T = 4$$
$$t_c = 225\ N + 0.3\ N^2 \qquad T = 9 \qquad [29]$$
$$t_c = 91\ N + 0.08\ N^2 \qquad T = 16$$

increasing array size.

## 10.4 Lifting-Surface Panel method

The Interaction Velocity Field method for solving rudder-propeller interaction is described in Ref. [7]. It is an iterative process: for each iteration the flow is solved individually for a rudder and propeller geometry. The performance of PALISUPAN is problem specific. It is dependent on the total number of panels, panel distribution, wake shape, imposed inflow velocity field, presence of a reflection plane or axi-symmetric images, variables such as rudder incidence or propeller r.p.m. and so on. As an illustration the overall performance is given for a typical rudder geometry (400 panel Rudder No. 2 at -0.4$^{o}$ incidence with representative propeller velocity field) and propeller geometry (400 panel 4 bladed propeller and hub).

Figure 16 shows the total time (in minutes) for one complete rudder-propeller interaction cycle for three advance ratio. Each total time is broken down into its four components. The propeller calculation (both solution and velocity field) dominates the overall time. The increase in time with reducing advance ratio indicates the increased three-dimensional rudder/propeller inflows in the more accelerated propeller flow (low J).

It is worth noting the fact that although surface panel methods are considered to only have a moderate computational requirement, this work has demonstrated that there are large possible savings in time through the use of parallel processing. In this case, if the velocity field calculations are assumed to work at 90% efficiency, the propeller at 80% and the rudder calculation at 75% the overall process efficiency is 83%. This is very good performance for what should be a worst case test for parallel processing.

The cumulative tally of calls to the Newman panel process is 7.36 million which gives an average time per call of 9 clock ticks on a 4 transputer array. The time to carry out the J=0.51 iteration cycle estimated for a single transputer (with enough memory) would be 226 minutes, measured on a 4 transputer array as 68 minutes, estimated for a 9 transputer array at 30 minutes and estimated for a 16 transputer array as 17 minutes.

## 10.5 Summary

The high code efficiencies (70% to 90%) obtained for the PALISUPAN component processes indicate that:

1) The Harness communications process performs well for the large communications requirements of implicit algorithms.

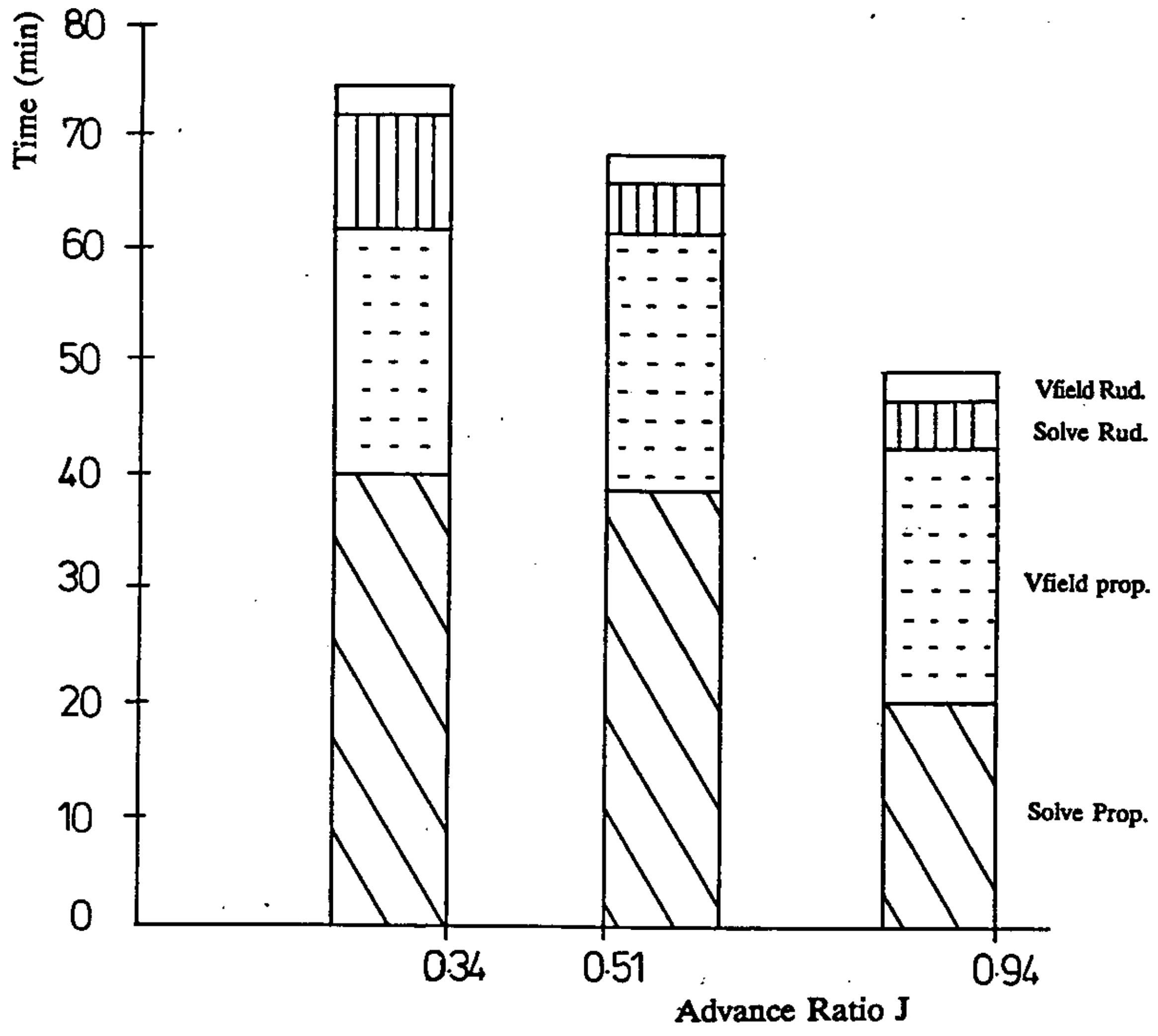


Figure 16 Comparison of time to carry out a single rudder-propeller iteration cycle for different advance ratio

2) The use of geometric data distribution and geometric algorithms allows implicit methods to perform as well as explicit algorithms and to benefit from the use of parallel processors.

From the analysis of the calculation influence coefficient performance and that of Harness it is expected that there will be little or no degradation with increasing size of transputer array as long as there are at least 25 panels located on each transputer. The expressions given should allow reasonable estimates of the performance of these algorithms to be made for different sized arrays of transputers.

The overall performance of PALISUPAN in solving rudder-propeller interaction illustrates the benefits of parallel processing. On a 16 transputer array the time per complete iteration cycle would be of the order of 20 minutes, a reasonable timescale for the investigation of detailed design information.

A large scale panel problem such as the analysis of a complete aircraft with flaps with the order of 10,000 panels could be solved using 121 transputers with 8MBytes per transputer. The estimated time to solve such a system, with the order of 200 million calls to Newman panel, would be of the order of 60 minutes. This performance comparable is with that of existing super-computers for a fraction of investment and running costs.

The transputer used in this work is now a 5 year old design. The rapid developments in similar devices will result in even better performance and greater cost saving in the future.

## 11 Conclusion

A complex lifting-surface panel algorithm has been successfully implemented to run across variably sized square arrays of transputers. The ease with which the parallel algorithm has been developed has highlighted the use of the geometric approach for both the algorithm and data distribution. Development time was also reduced through the separation of the CFD algorithm from the Harness.

Significant memory saving and reduced programming time has resulted from the development of a live-memory datastore process for storing large arrays of information.

A large number of standard algorithms has been implemented in the lifting-surface program and the measurement of their performance can be used to provide information on how well other computational fluid algorithms would perform.

## References

[1] Turnock, S.R., "Lifting surface panel method for modelling ship rudders and propellers". Ship Science Report No. 50, University of Southampton. 1992.

[2] Allwright, J.,"Parallel algorithms for combinatorial optimization on transputer arrays", Ph.D Thesis, Department of Electronics and Computer Science, University of Southampton, 1990.

[3] Turnock, S.R., "Parallel Implementation of an Explicit Finite Volume Euler Solver on an Array of Transputers ",ICAS Congress 1990, Proceedings Vol 2.,pp. 1557-1568.

[4] Rua, R., "Parallel computation of eigenvalues and eigenvectors using occam and transputers", Ph.D. Thesis, University of Southampton 90--076060, 1990.

[5] Clark, R.W., "A new iterative matrix solution procedure for three-dimensional panel methods", Proceedings of AIAA 23rd Aerospace Science Meeting, Reno, Nevada, Jan. 1985.

[6] Kerwin, J.E., & Lee, C-S, "Prediction of steady and un-steady marine propeller performance by numerical lifting surface theory", SNAME Transactions, Vol. 86, 1978, pp. 218-253.

## APPENDIX A INDEX PROCEDURE FOR PANEL STORE

The co-ordinate data in the Panel store and other live-memory store derivatives is stored in the manner, best illustrated using a series of nested SEQ loops, where index is a variable indicating the position from the zeroth element of a one-dimensional array of REAL32 numbers, and Panel number is the absolute number defining a particular set of four ordered co-ordinates.

```
TotNoPanperT := N / T     -- where N is the total number of panels and T the total number
                          -- of Guest transputers

Mstart := (TotNoPanPerT * ( PNo - 1 ) ) -- where PNo is the absolute guest transputer -
                                        -- number between 1 and T
Body[0]:= 0

Pstart[0]:= Mstart

SEQ i = 0 FOR (NB + NLB)  --  Number of bodies NB, and number of lifting bodies NLB

    Body[i+1] := 3 * (Nt[i] * Ns [i]) + Body[i]
    Pstart[i+1] := Pstart[i] + ( (Nt[i]-1) * (Ns[i]-1) )

    SEQ j = 0 FOR Nt[i] -- Number of nodes in t direction for body i

        SEQ k = 0 FOR Ns[i] -- Number of nodes in s direction for body i

            X coordinate stored at index:= Body[i] + 3 * ( (j * Ns[i]) + k )
            Y coordinate stored at index:= Body[i] + 3 * ( (j * Ns[i]) + k )+1
            Z coordinate stored at index:= Body[i] + 3 * ( (j * Ns[i]) + k )+1
            PanelNo: = Pstart[i] + ( ( j * Ns[i] ) + k )
```

To find the four vertices of a given absolute PanelNo the following process is carried out:

```
    RelativePanelNo := PanelNo - Mstart

    BodNo := 0
    WHILE RelativePanelNo < Pstart[BodNo+1]
        BodNo := BodNo + 1

    RelativeBodyPanelNo := RelativePanelNo - Pstart[BodNo]

    j value (t direction) := RelativeBodyPanelNo / ( Ns[BodNo] - 1 )
    k value (s direction) := RelativeBodyPanelNo - ( j * (Ns[BodNo] - 1))

    Panel Node P1 is j,k
```

Panel Node **P2** is j+1,k
Panel Node **P3** is j+1,k+1
Panel Node **P4** is j,k+1

The outward facing panel normal **n** is defined as:

$$\mathbf{n} = (\mathbf{P3} - \mathbf{P1}) \times (\mathbf{P4} - \mathbf{P2})/(\,|(\mathbf{P3} - \mathbf{P1})|\;\;|(\mathbf{P4} - \mathbf{P2})|\,)$$

## APPENDIX B GEOMETRY INPUT FILE

A standard ASCII format text file is used.

The first line defines the number of non-lifting bodies, lifting bodies, and number of free panels. Then, if a plane of symmetry is to be used or axisymmetric body the necessary axes, origins and rates of rotation are defined.

The velocity field is specified as a uniform distribution of points Nx, Ny, and Nz in the overall cartesian or cylindrical axes system directions. At each point a scaled velocity vector is given relative to a defined uniform inflow velocity.

The definition of the non-lifting and lifting surface bodies is carried out in order. For each body the pivot, offset, scale, and angle of its own body centred axis system is given as described in chapter four. Each body is defined by a variable number of sections, and each section consists of a variable number of points. Also given are the number of spanwise and chordwise panels for the body. Two parameters define the type of panel size distribution on the body. Example distributions allow clustering of panels at body leading and trailing edges using various forms of sinusoidal function.

For the lifting surface bodies additional information is given to describe the trailing edge wake sheet in a manner similar to that of the body itself. An example lifting surface body definition, with comments, is given below for Rudder No. 2. Additional comments denoted **(i)** are given at end.

```
  0(NB)    1(NLB)    0(NFP)           No. of Bodies,  No. of Lifting Bodies,  Dummy
  0.0   0.0   0.0   Origin of reflection plane  x , y, z
  0.0   0.0   1.0   Unit normal to reflection plane  nx, ny, nz
  0.0   0.0   1.0   dummy nx,ny,nz
  0.0   0.0   0.0   Origin of axis of rotation  x , y, z
  0.0   0.0   0.0   Axis of rotation, magnitude is rate of rotation in rad/sec
  10.0                      scale velocity m/s
   1       1   1    velocity cube no of points in x,y,z direction
   0.0     0.0  0.0         origin of velocity cube  Vxo,Vyo,Vzo
   0.0     0.0  0.0  increments in Vdx, Vdy ,Vdz
(i) 1.0    0.0   0.0        non-dimensional velocity vx,vy,vz, data for (NVx.NVy.NVz)
sets
(ii)  26   17   3   -1 Nt, Ns, NS, Image
```

**(iii)** 25 10 No of wake panels, No of free panels { this line is not used for a body}

```
(iv) 4   0  1  0   t distribution, s distribution, t close, s open
  30.0   0.0   0.0 Pivot Vector, x, y, z
  0.0   0.0   0.0  Offset Vector, x, y, z
  0.00667   0.0667  1.0   Scale Vector, x, y, z
  0.0   0.0   -0.4 Ordered rotation: angle about x, about y, about z axis ( all in degrees)
(v) 35                 NPS No. of points in each section
  100.0  0.0   0.0      start at trailing edge  x, y, z
  95.0   0.134  0.0
  90.0   0.241  0.0
  80.0   0.437  0.0
  ....   .....  ...
  5.0   0.592  0.0
  2.5   0.436  0.0
  1.25  0.316  0.0
  0.0   0.0    0.0 around leading edge
  1.25  -0.316  0.0
  2.5   -0.436  0.0
  5.0   -0.592  0.0
  7.5   -0.7    0.0
  ...   ....   ....
 95.0  -0.134   0.0
 100.0  0.0     0.0       and finish at trailing edge x,y,z
 35                 repeat for next two sections
 100.0  0.0    0.5 start of middle section
 95.0   0.134  0.5
 ....   .....  ....
 90.0  -0.241   0.5
 95.0  -0.134   0.5
 100.0  0.0     0.5
 35
 100.0  0.0    1.0 start of top section
 95.0   0.134  1.0
 ....   .....  ....
 70.0  -0.611   1.0
 80.0  -0.437   1.0
 90.0  -0.241   1.0
 95.0  -0.134   1.0
 100.0  0.0     1.0
(vi) 4                 no of points in each of three wake sections
  100.0 0.0     0.0
  125.0 0.0     0.0
  150.0 0.0     0.0
  250.0 0.0     0.0
  4
  100.0 0.0     0.5
  125.0 0.0     0.5
```

```
150.0 0.0    0.5
250.0 0.0    0.5
4
100.0 0.0    1.0
125.0 0.0    1.0
150.0 0.0    1.0
250.0 0.0    1.0
```

(i) Velocity field information is ordered in the manner shown below

```
SEQ i=0 For NVx
  SEQ j= 0 For NVy
    SEQ k=0 For NVz
        $Vx_{i,j,k}$ , $Vy_{i,j,k}$ , $Vz_{i,j,k}$
```

and the co-ordinates (using the same indices) as: x:= Vxo + (i*Vdx)
y:= Vyo + (j*Vdy)
z:= Vzo + (k*Vdz)

A cylindrical velocity field is designated by the z component of the index increment having a value greater than 10000. The origin of the axis is taken to be the defined origin, and the unit vector in the axis direction **A** is defined thus: Ax := Vdz - 10000
Ay := $Vy_{0,0,0}$
Az := $Vz_{0,0,0}$
Vdx is the increment in the axial direction, and Vdy the increment in radius dr. NVz is always equal to one, and Vy corresponds to the radial velocity Vr (positive outward) and Vz the velocity on the circumferential direction Vw. The direction of Vw is determined by the cross-product **A x (0,0,1)**.

(ii) Nt = No of nodes in t (usually chordwise direction), number of panels is Nt-1,
Ns = No nodes in s (usually spanwise direction), if the total number of transputers is greater than 1 has to be a multiple of the total number of transputers +1
NS = No of sections of data used to define body, limit on TOTAL number of sections to define a body or multiple bodies is 30, a minimum of 3 sections per body.
Image = 0 { no reflection plane} , = -1 { reflection plane located at defined origin and with defined outward facing normal }, = +N { no of degrees of rotational symmetric bodies, 1 is just defined body, >1 is 1 body + N-1 images at a spacing of $360^{o}$/N degrees, origin and rotational axis of symmetry are given at start of file }.

(iii) First integer is total number of nodes to be used to panellise wake sheet strip, of which there will be (Ns-1), second integer is the number of wake panels which are allowed to move in wake adaption process.

(iv) First integer is the type of panel size distribution in the t direction and the second integer the distribution in the s direction. At present the allowed mapping function to give the distribution are:

| | |
|---|---|
| 0 | Equal sized panels of Δ |
| 1 | Sinusoid clustering as parameter tends to unity |

| | |
|---|---|
| 2 | Sinusoid clustering as parameter tends to zero and unity |
| 3 | Sinusoid clustering as parameter tends to zero, half and unity |
| 4 | Sinusoid clustering as parameter tends to half |
| 5 | Sinusoid clustering as parameter tends to zero |
| 6 | Modified sinusoid clustering as parameter tends to zero with minimum size of $0.25\Delta$ |
| 7 | Modified sinusoid clustering as parameter tends to zero with minimum size of $0.5\Delta$ |

The third integer is set to 1 if the t direction is a closed shape e.g. aerofoil, or 0 if open. The fourth integer states whether the s distribution is open or closed.

**(v)** The number points defining each of the NS sections is variable between 3 and 100.

**(vi)** The number of wake sections is equal to the number of lifting body sections, with points defined from the trailing edge direction.

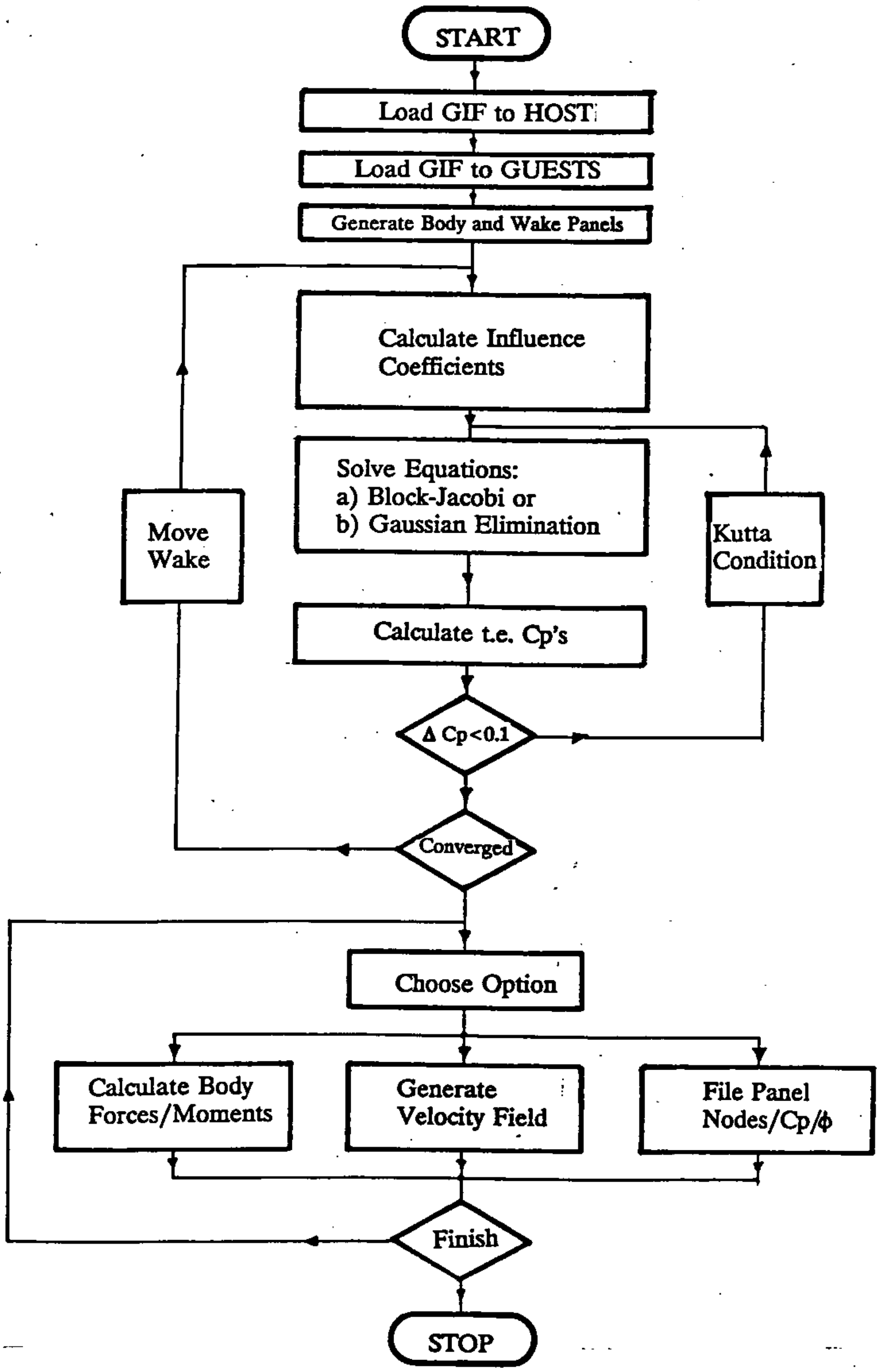


Figure 1 Flow chart of overall lifting surface algorithm

geometry definition, each guest process generates its allocation of surface panels. As the parametric cubic splines used on each process are based on identical information the panel boundary points between transputers will be same.

To ensure as even a load balance as possible and minimise delays each transputer is assigned, where possible, the same number of panels. In addition, as a considerable amount of numerical processing is required to calculate the wake strip influence coefficient, each transputer has the same number of wake strips. Delays may still occur as different schemes are used to calculate the panel-node influence coefficient in the near, mid and far field.

The method chosen to ensure an even load balance involved sub-dividing each body into streamwise panel strips with an equal number of streamwise strips assigned to each transputer. For example, this results in a two rudder problem as shown in Figure 2, where

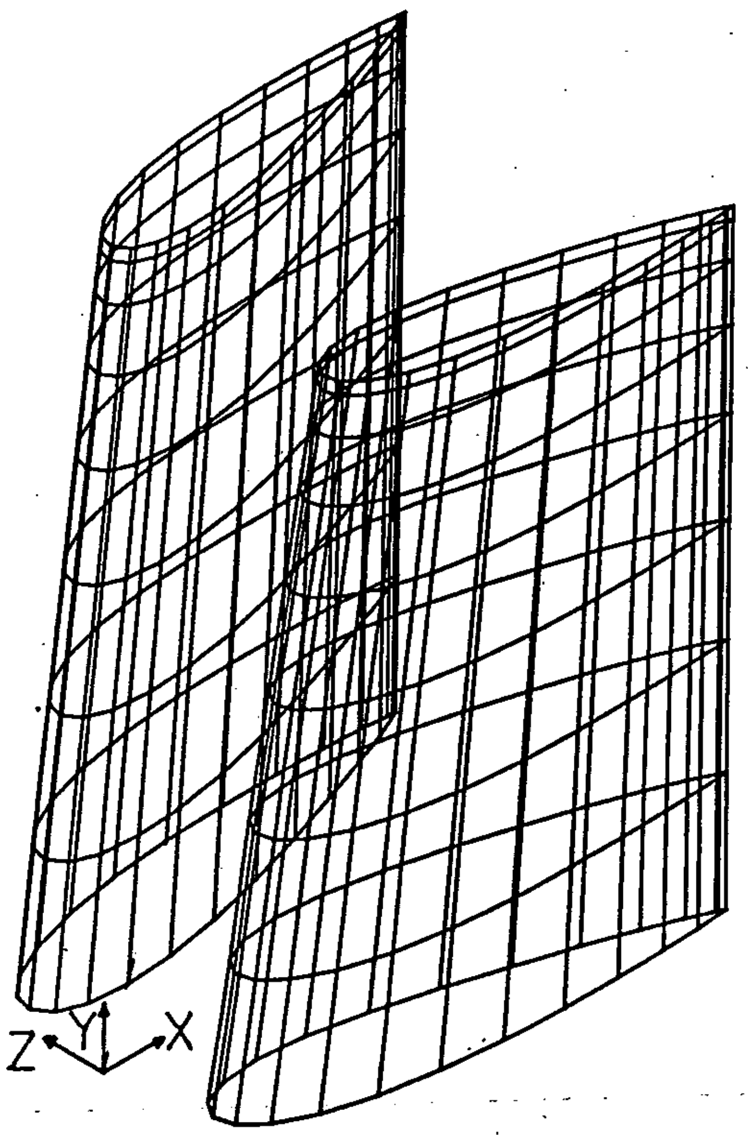


Figure 2 Panelling arrangement for two rudder arrangement

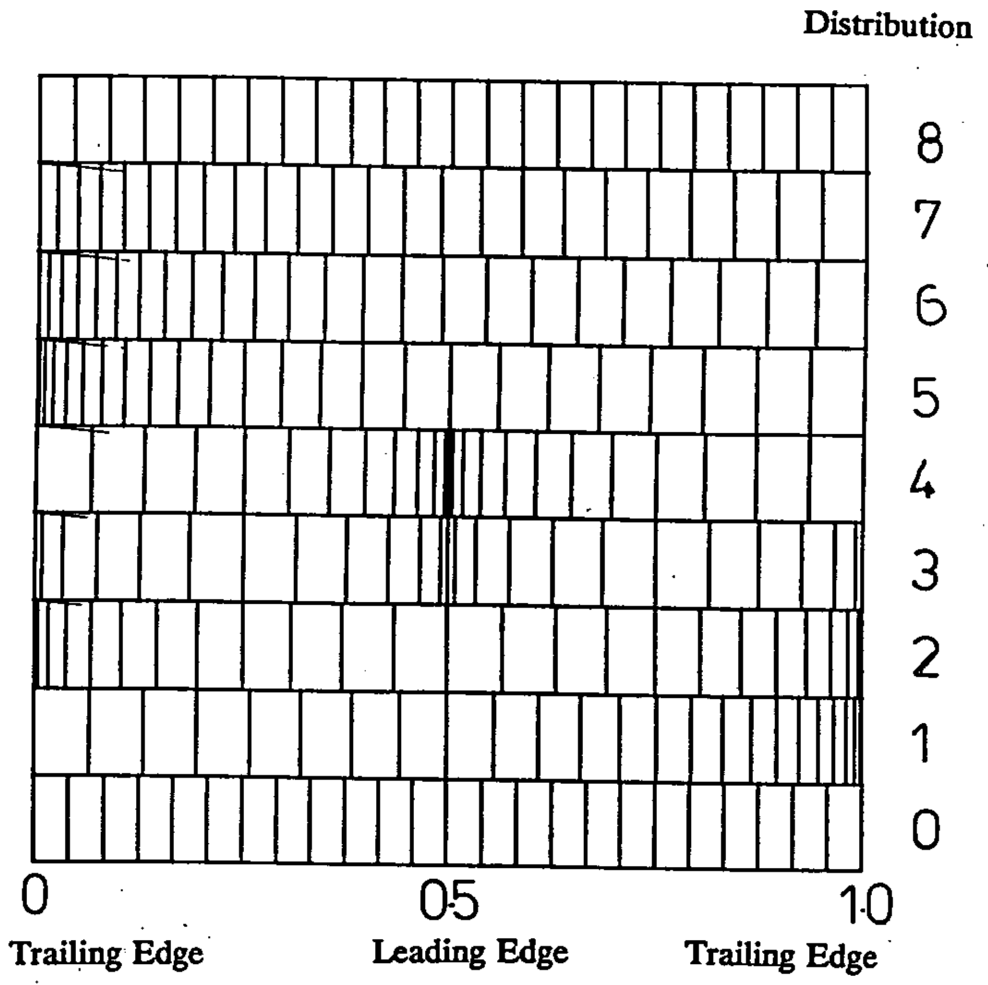


Figure 3 Possible panel size distributions

for a point anywhere within the domain of interest, the absolute inflow velocity at that point. A tri-directional linear interpolation is used to calculate the inflow velocity within the defined velocity mesh. External to the mesh the inflow velocity is set equal to the scale velocity in the freestream direction. Either a cylindrical or cubic three-dimensional grid can be used to define the velocity field. The choice is dependent on the application. For a circumferentially averaged inflow field less information is required to define the velocity field in cylindrical coordinates and this method is used for defining rudder-propeller interaction velocity fields.

### 4.7 Performance

The time for the geometry input file to be read, distributed to the array of transputers, and then the relevant store panels generated is a quick process. It is dependent on the number of bodies, number of sections for each body, number of wake sheets and the number of transputers. The lifting surface definition is a farm algorithm. The host process distributes the data packets and the individual guest processes reply on completion of the panel

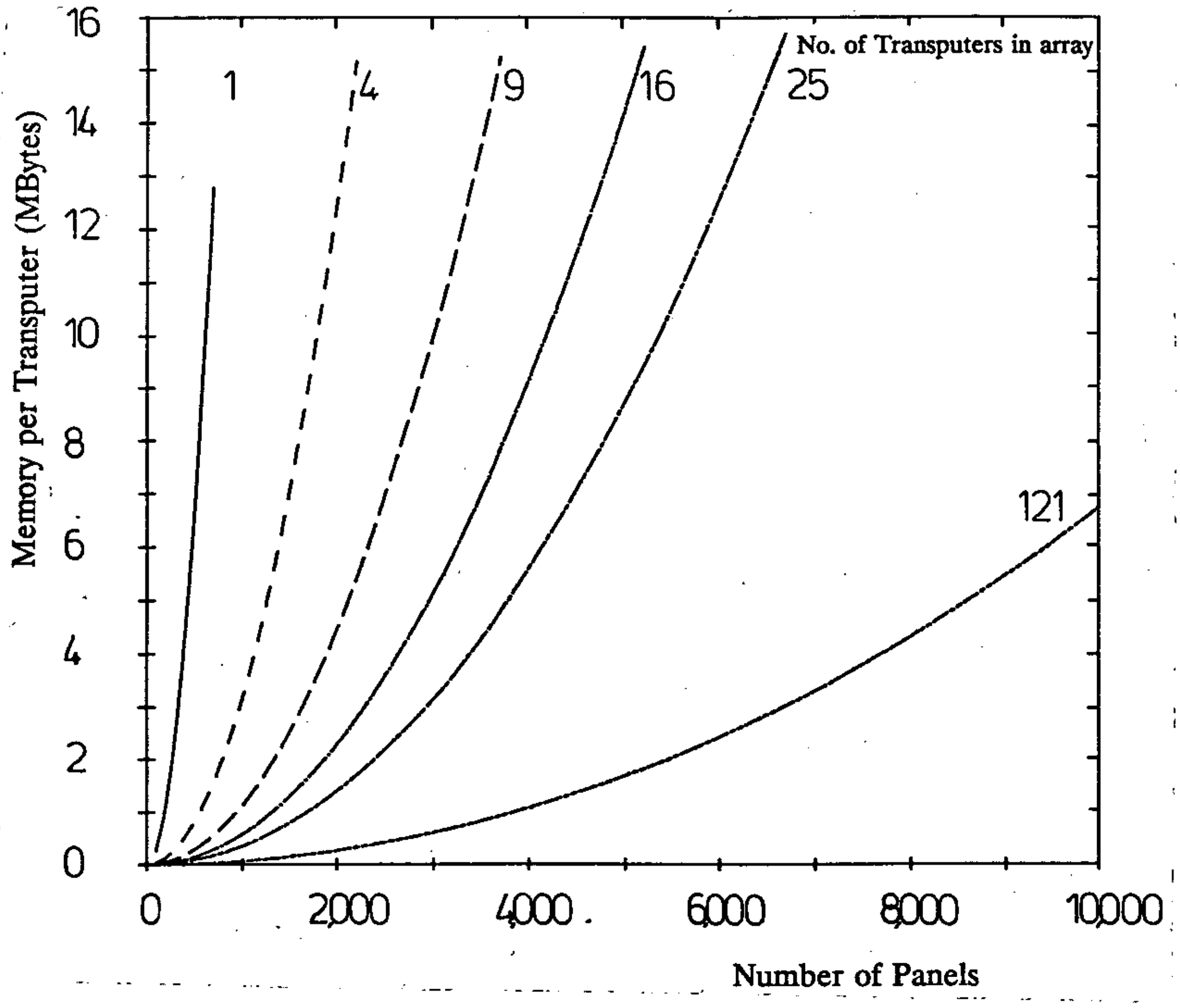


Figure 4 Individual transputer memory requirement for a given number of surface panels

where each real number requires 4 bytes and all the memory requirements but that for the panel and dipole store arrays are ignored. This provides a good estimate of the number of panels which can be solved on a given transputer array. The different lines represent various sized arrays of transputers. For instance the Ship Science Transputer System, of dimension N=2 and 1 Mbyte of memory per transputer, could solve a 400 panel problem. A machine with 8 Mbyte per transputer could solve a 10,000 panel problem if it were of dimension 11. A surface panel problem containing this number of panels is the order of the largest scale of lifting-surface problems currently solved on supercomputers. The Meiko Computing Surface available for use has 8MByte per transputer and with a sixteen transputer array could solve a 3800 panel problem.

The memory requirement for the dipole matrix emphasises the need to keep the memory demand of the lifting surface code to a minimum even at the expense of additional numeric calculations.

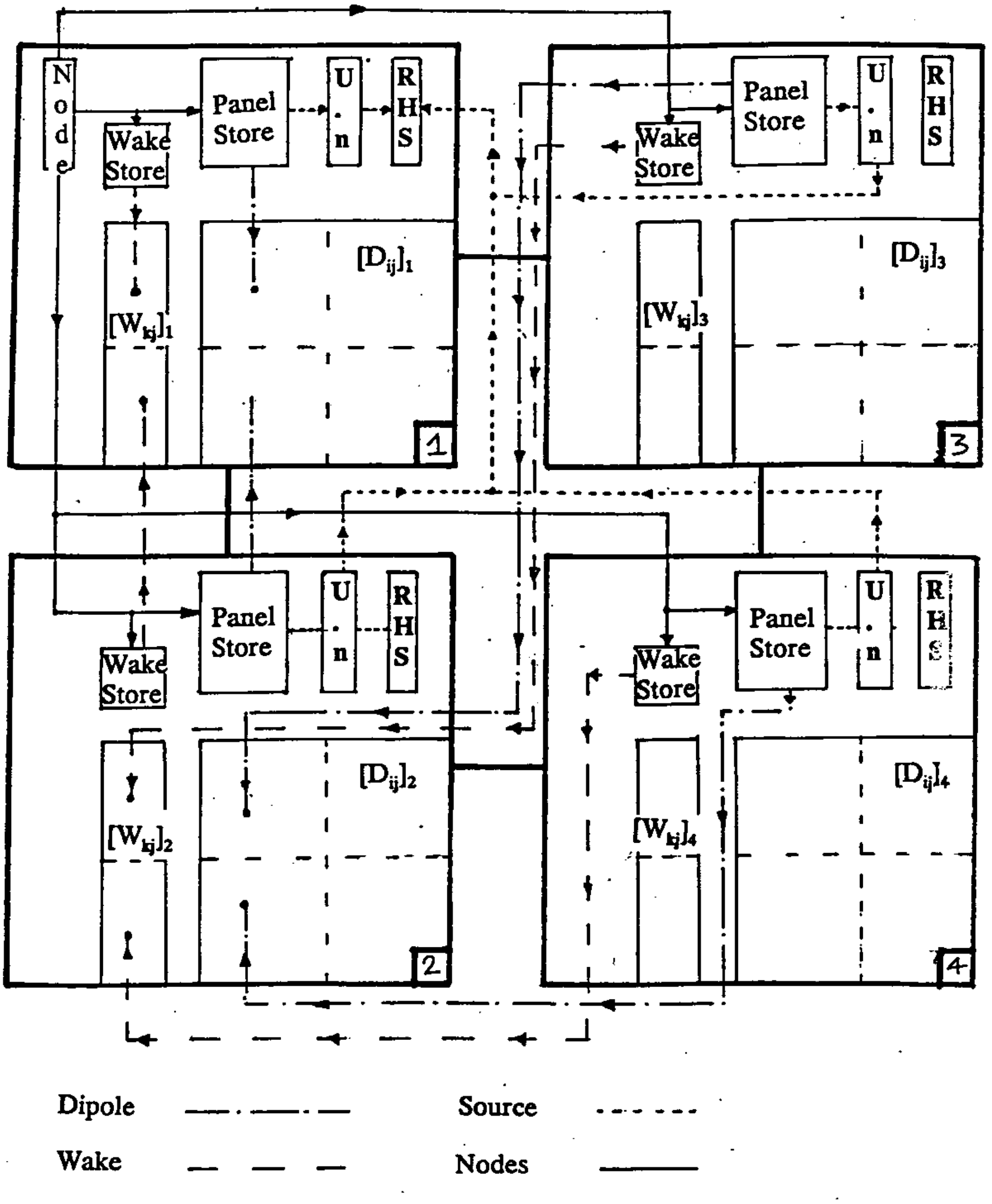


Figure 5 Data flow schematic for Calculate Influence coefficient process on a four transputer array

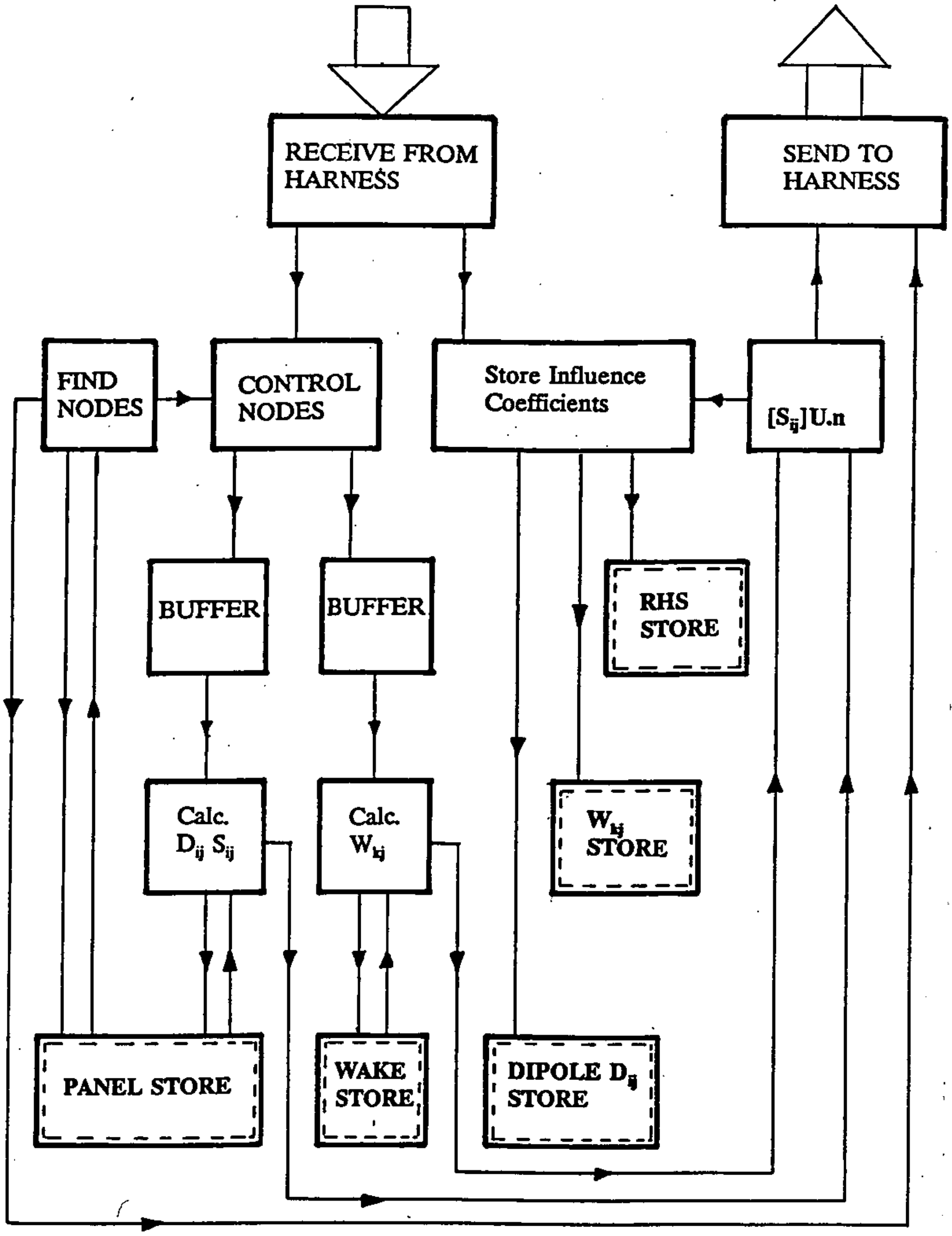


Figure 6 Internal structure of the calculate influence coefficient process

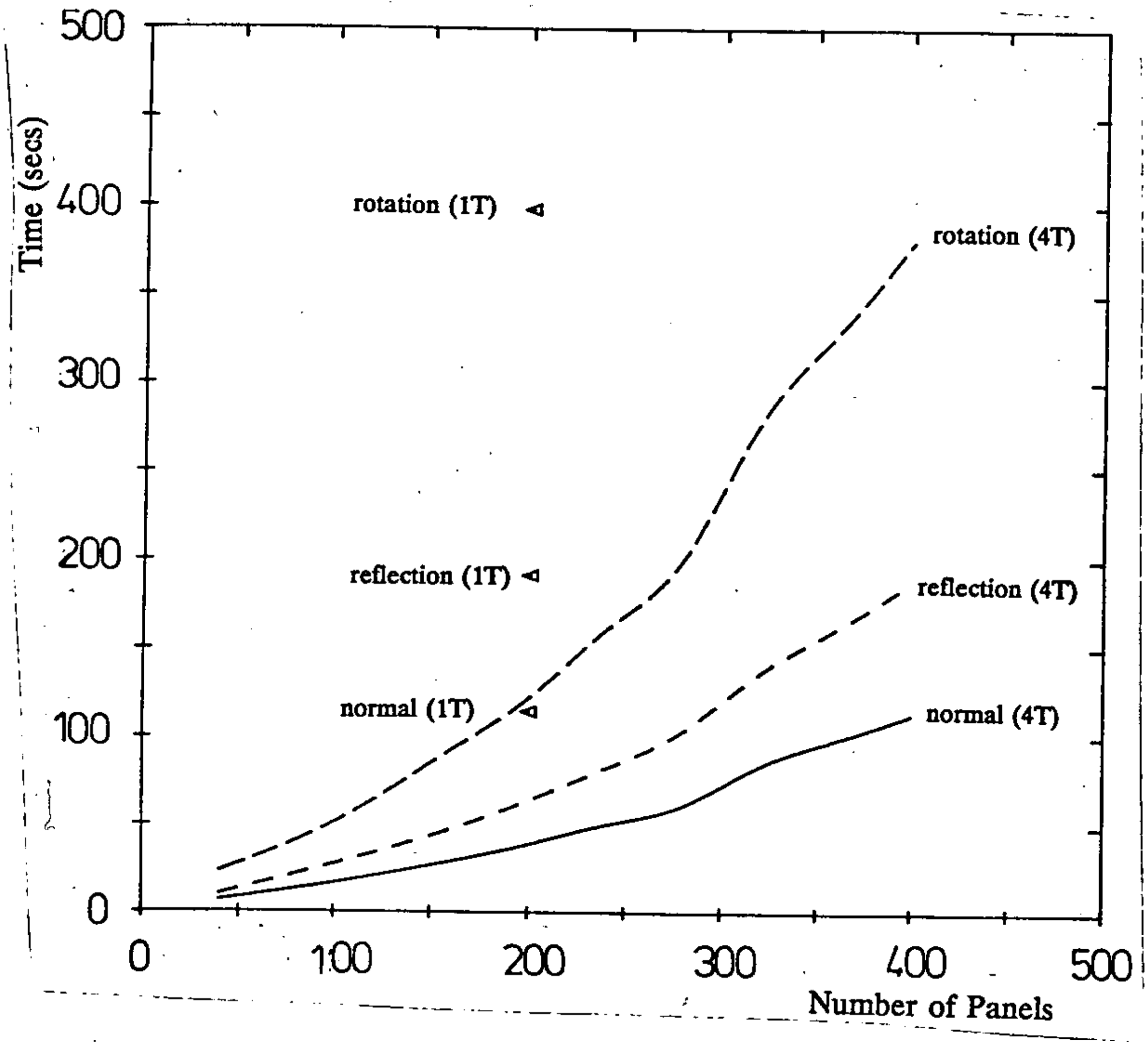


Figure 7 Performance of calculate influence coefficient on a four transputer array

against the time to calculate all the influence coefficients, lines are shown for three test cases of a rudder without reflection plane, a rudder with reflection plane and a four-bladed propeller. As the amount of numeric processing increases as the number of image panels increases the overall time increases. However, if the time to run a 200 panel problem on four transputers and one transputer is compared speed-ups of 2.96 ($\eta_c$=73.9%), 2.99 ($\eta_c$=74.8%) and 3.28 ($\eta_c$=82%) are obtained for the three cases which demonstrates the increased efficiency with additional numeric processing per panel. The relatively small increase in efficiency is due to the additional operations to reflect/rotate panels and generate their geometry coefficients.

Overall the calculate influence coefficient process exhibits a quadratic increase in processing time with the number of panels. The knee in the curve occurs when the panel centroid is sent as two packets rather than one which results in extra processing as each panel geometry data is generated twice as many times.

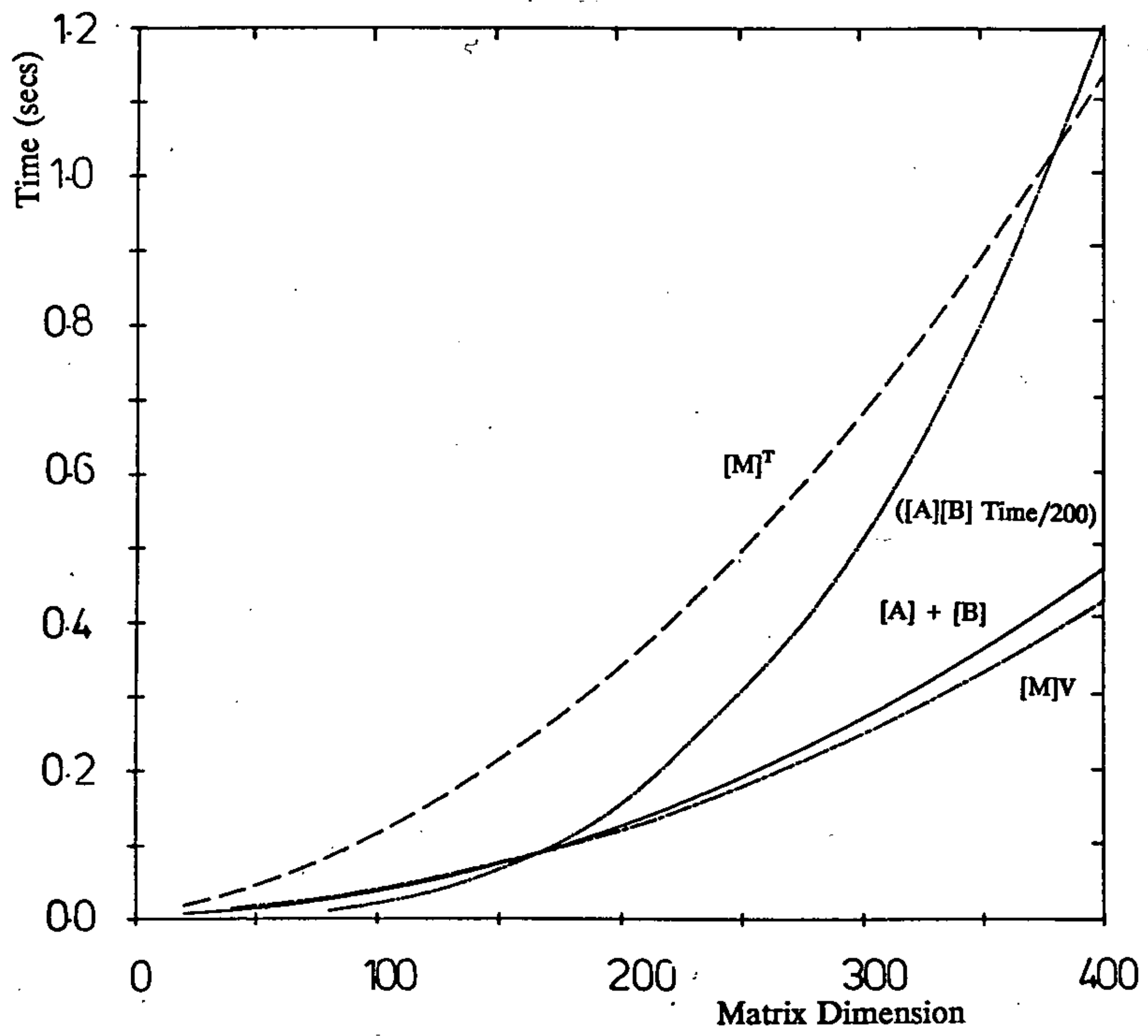


Figure 8 Time to carry out basic matrix operations on a four transputer array

```
PAR
  ... SendRow
  ... ReceiveColumn
  ... GetRows(Bin,Bout)
  ... StoreColumns(Ain,Aout)
```

where if the transputer is on the main diagonal each retrieved row is sent directly to the StoreColumn process otherwise it is passed across the harness through the SendRow process. At its destination it is received by ReceiveRow and then passed to StoreColumn. A matrix transpose simply involves communication. Figure 8 shows the time to carry out a transpose against a base of overall matrix size for a four transputer array, again a quadratic behaviour is seen.

Matrix-vector multiplication
requires the summation of the multiplication of each element of a matrix row with the corresponding element of the vector. That is:

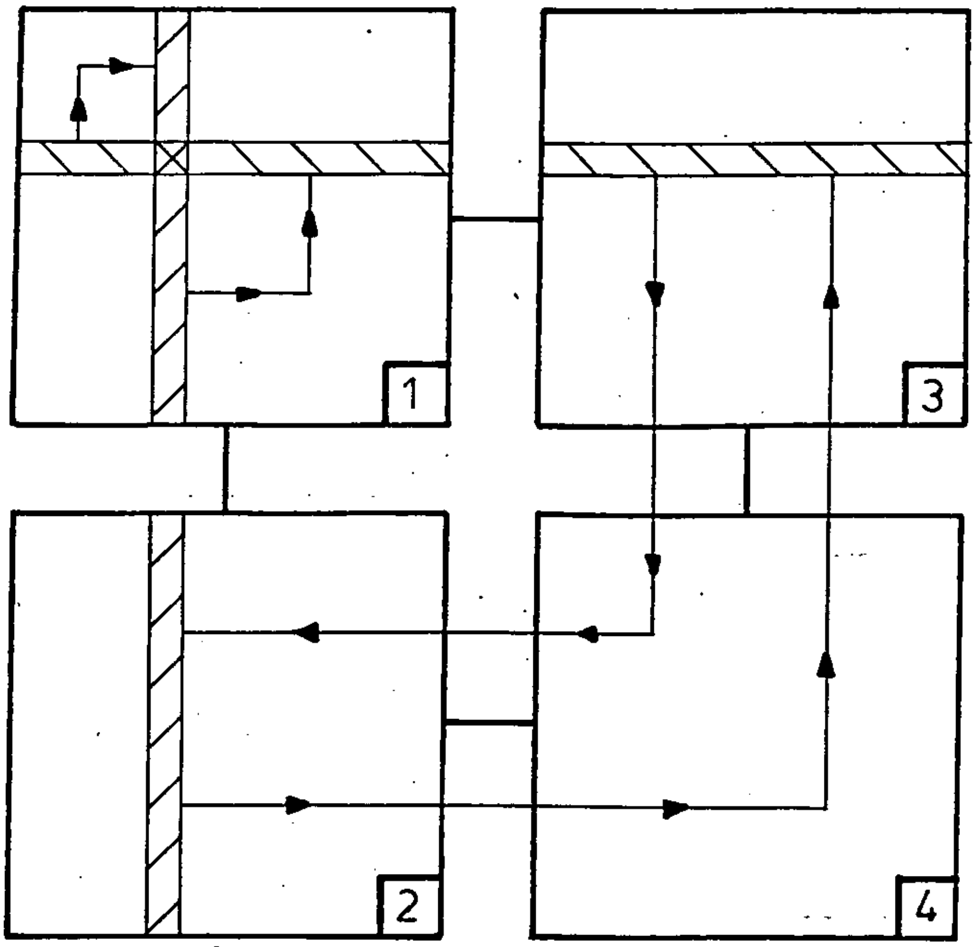


Figure 9 Schematic of matrix transposition on a four transputer array

$$Q_i = \sum_{j=1}^{Msize} M_{ji}.V_j \qquad [13]$$

A vector is stored as a column through the transputer network, and as shown is schematically in Figure 10. For a 4 transputer network each guest process sends its part of the vector to all the transputers in its column in the array. This sub-row is summed with the corresponding elements in each of the rows and then the part answer vector sent back to its correct location. Each transputer carries out the same amount of communication which is proportional to the dimension of the transputer array. Figure 8 also gives the time to carry out matrix-vector multiplication for different sized matrices. It can be seen that the performance is similar to the matrix add operation; the number of operations being equivalent and the communication low.

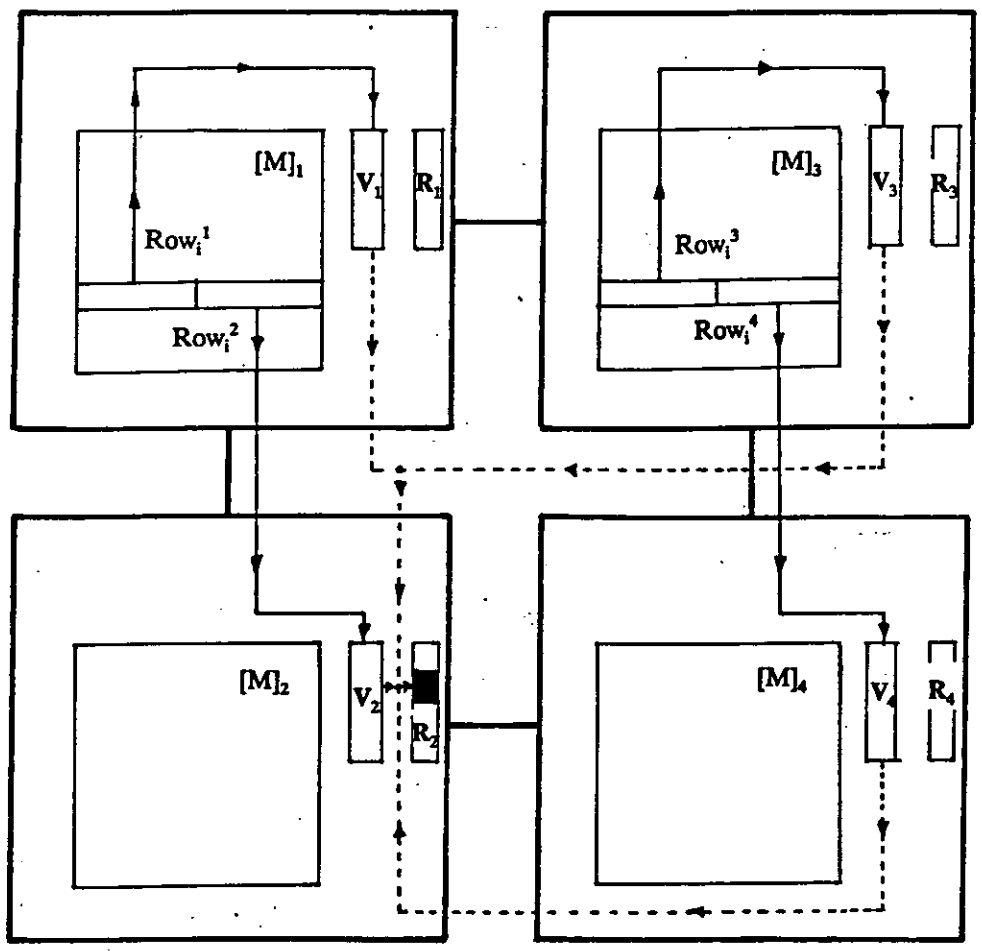


Figure 10 Schematic of matrix-vector multiplication on four transputers

Although matrix-matrix multiplication is not required as part of a solution procedure it is none the less an interesting test for a parallel computer. As every element in each matrix has to be multiplied together it requires a large amount of communication for a bare minimum of numeric processing. The approach chosen was to first transpose one matrix. Then every column in one matrix is passed horizontally through the array in a similar manner to that used in the matrix-vector multiplication and the result vectors are sent to their final destination and summed to give the result matrix. The time to carry out this operation, as also shown in Figure 8 (to a scale of time(secs)/200), is about 2/3rds of the time taken to transpose a matrix and then carry out N matrix-vector multiplications. The difference is due to the reduced amount of control commands from the Host transputer. Overall, matrix-matrix multiplication exhibits a cubic behaviour.

## 6.3 Iterative versus Direct methods

The choice of technique for solving a system of linear equations revolves around whether an iterative approach can converge more rapidly to solution than a direct method takes to explicitly solve the whole system. In general a dense full matrix, such as that generated by a lifting-surface, is more likely to favour an iterative approach. This is

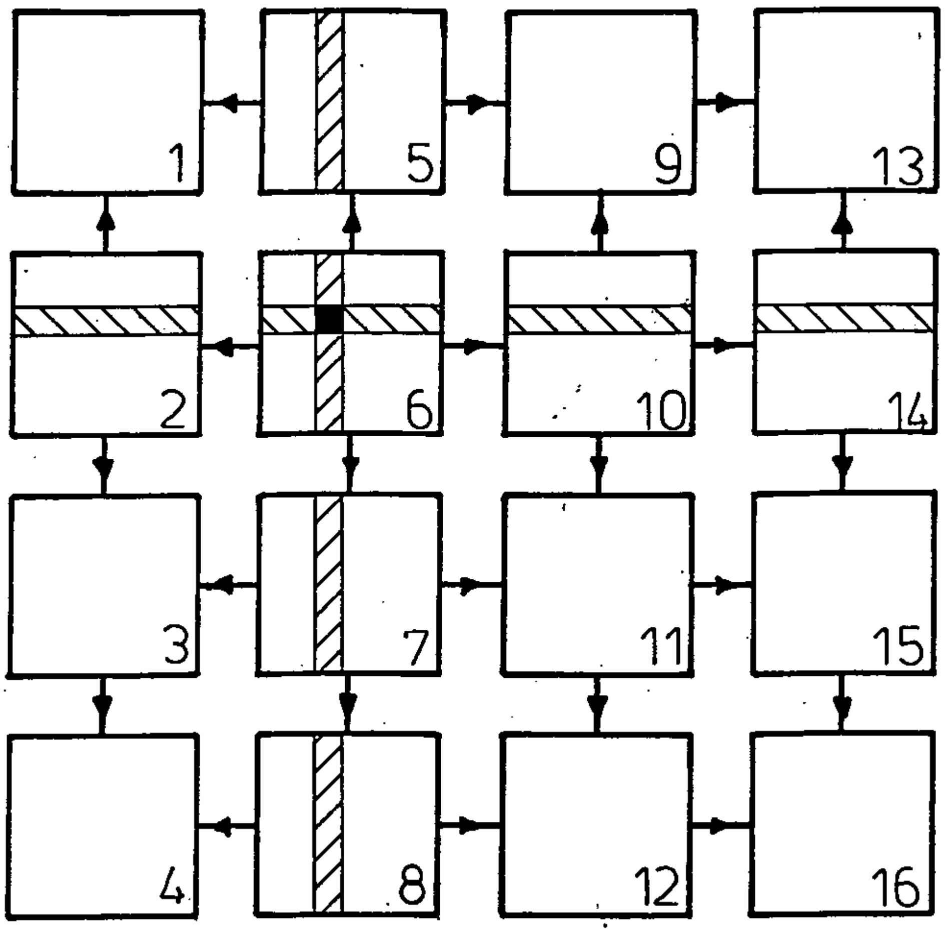


Figure 11 Schematic of Gaussian elimination on a sixteen transputer array

Two matrix stores are used for the inversion. At the completion of the process the original matrix is the identity matrix and the other is the required inverse of the original.

## 6.5 Jacobi-Iterative Scheme

An iterative scheme uses an initial guess for the unknown vector $\phi$ to generate a better approximation to the solution. The process is repeated until the solution has converged to a given level of accuracy. The simplest possible method for diagonally dominant matrices is to use the main diagonal elements to calculate the correction. This process is known as the Jacobi method.

The Jacobi correction $\Delta\phi$to the original estimate of vector $\phi$ can be written as:

$$R^{*} - [\, M \,]\, \phi^{k-1} \qquad [17]$$

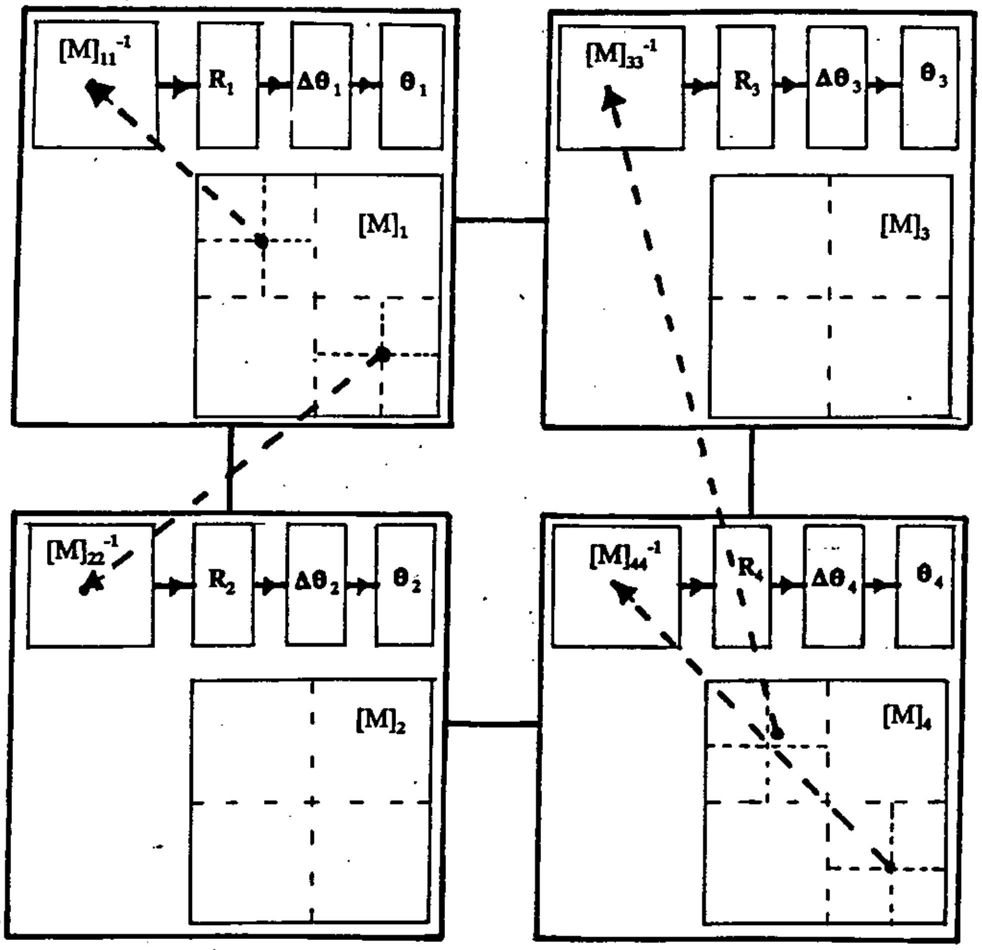


Figure 12 Schematic of Single block iterative scheme on four transputers

except that instead of the Vector Divide stage the sub-matrix inverse is used to multiply the vector difference **R**-**R**'. That is:

$$\underline{\Delta\phi}^{k}_{T} = [\,M\,]_{TT}^{-1}\left(\underline{R}_{T} - \left([\,M\,]\underline{\phi}^{k-1}\right)_{T}\right) \qquad [20]$$

where T is the transputer number and $[M]_{TT}^{-1}$ the sub-matrix inverse. The implementation of the scheme is identical in all other respects to the Jacobi. A drawback to this scheme is that if the number of panels per transputer gets too large the inversion of the sub-matrix will take a long time. As in the case of the Jacobi scheme for each iteration the only communication is during the matrix-vector multiply stage. Also, once the sub-matrices have been passed to their destination, the direct inversion is an independent process.

## 6.7 Multi-Block Jacobi Scheme

To restrict the maximum block size the dimension of the sub-matrices was made variable between 1 (Jacobi - single element) and that for 1 block per transputer (single-block). This is also shown schematically in Figure 12. The implementation is the same as for the single-block scheme except that each guest process has a number of blocks sent to it. Each block is then inverted and used in the iteration process to update its portion of $\Delta\phi$ vector. The ability to change the block size allows the iterative procedure to be tuned to obtain minimum iteration time for a given lifting-surface problem.

## 6.8 Comparative Performance

Figure 13 shows the time in seconds to solve a system of linear equations against the size of the system. Plots are shown for the four schemes previously described and also the time to set up the influence coefficient matrix. The data given is for a four-transputer system.

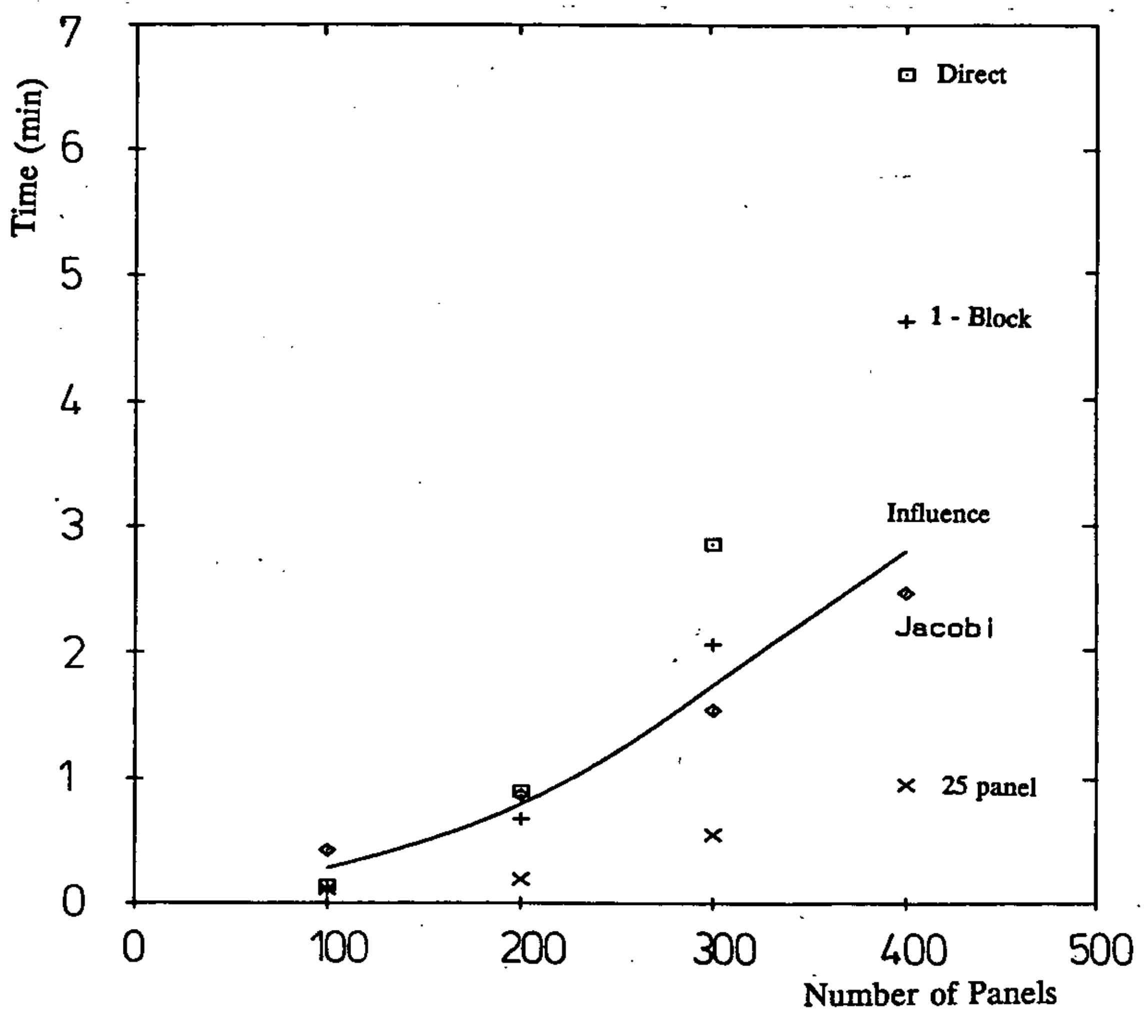


Figure 13 Performance of matrix solution procedures on a four transputer array

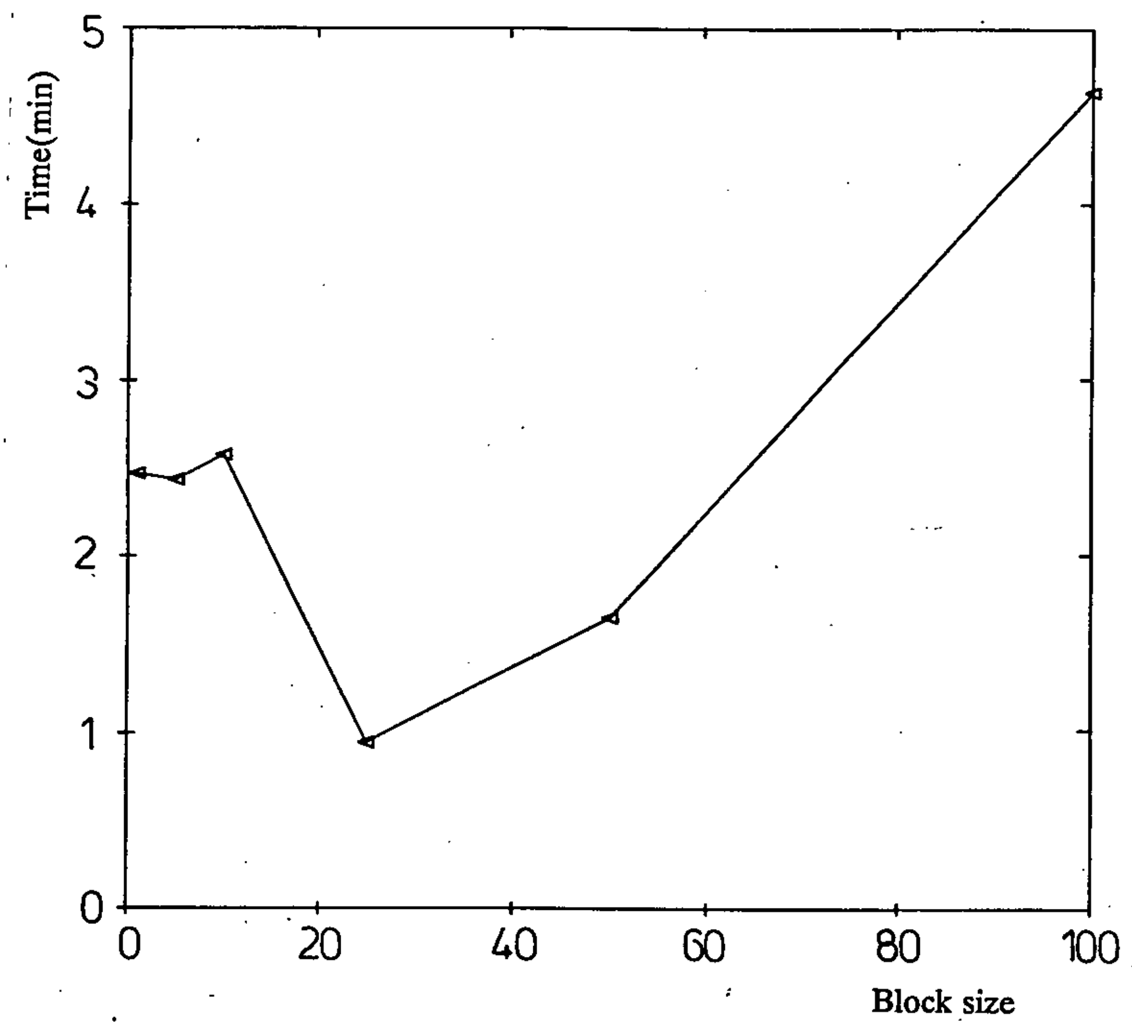


Figure 14 Variation in convergence time with block size for 400 panel problem

where m is the number of iterations to convergence.

The calculation of vector **R'** is rapid and performance figures are not given.

For the direct method of solution each Kutta condition iteration requires only a matrix-vector multiplication.

$$\underline{\phi}^{k} = [\ M\ ]^{-1}\ \underline{R}'^{k} \qquad [24]$$

Whereas, for the iterative solution methods the number of iterations to convergence generally halves after each Kutta condition iteration. As noted previously, the initial approximation is taken to be the previous final solution for all but k=0.

## 8 Calculation of Velocity Field

In modelling the interaction of a ship rudder and propeller a requirement is the development of velocity field both up and downstream from a lifting surface. Differentiation of the expression for the Newman dipole and source panel influence coefficients gives the disturbance velocity at a point in space due to a unit strength panel. Once the values for the individual dipole panel strengths $\phi$ and wake sheet $\Delta\phi$ have been solved the total disturbance velocity at a point due to the whole lifting-surface can be found.

The total velocity $U_T$ at a point is the vector sum of the disturbance velocity $U_d$ and the local inflow velocity $U_i$.

$$\underline{U}_T = \underline{U}_i + \underline{U}_d \qquad [25]$$

The velocity field within a volume of space can be found by determining the total velocity on a regular mesh of points within the volume of space. This information can be used for various post-processing tasks. For example, many three-dimensional flow visualisation techniques require a regular mesh of data points. Also, the interaction velocity field can be developed from such an arrangement.

A farm algorithm controlled by the Host process is ideal for carrying out the velocity field generation process. The user specifies the x, y, and z limits of the volume of space and the required number of points Nx, Ny, and Nz in each direction within that space. A packet of points is sent to all guest processes. Each guest calculates, for every point in the packet, the disturbance velocity field due to its body surface and wake sheet panels. Also, it finds the inflow velocity field. The two packets of velocity information are returned to the Host process. The total velocity at a point is the sum of all the guest disturbance velocity field plus the inflow velocity at that point:

$$\underline{U}_T = \sum_{i=1}^{T} \underline{U}_d + \underline{U}_i \qquad [26]$$

where T is the number of guest transputers. Packets are sent from the Host until the velocity has been found for all points defined by the user.

As it is a farm algorithm the performance of the velocity field process is proportional to the number of panels and, the number of points, and inversely proportional to the total number of guest transputers.

## 9 Adaptive Wake

The method described in Ref. [1] was straight-forward to implement using a parallel algorithm similar to that for calculating the influence coefficients. Each wake strip is divided into a near and far region. In the near region the panels are aligned with the local flow direction. For the far wake, the panels follow the direction of the most downstream free wake panel. The first stage of the algorithm requires the generation of the centroids of the free wake panels. The total velocity is then calculated at each centroid. Each packet of centroids is broadcast around the transputer array and the result packets returned to the originating Guest transputer. For the rotational case, the velocity is obtained in cylindrical coordinates, ($V_a$, $V_r$, and $\omega$). The velocity is stored for each panel node by averaging the values of velocity at the surrounding four panel centroids. This summing process requires