INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 14, 95–108 (1992)

PARALLEL ALGORITHMS FOR PANEL METHODS

TIM DAVID AND GRAHAM BLYTH

Department of Mechanical Engineering, University of Leeds, Leeds, LS2 9JT U.K.

SUMMARY

A parallel algorithm for the solution of potential flow problems using the panel method of Hess and Smith and conjugate and bi-conjugate gradient techniques is presented. Analysis of the parallelism for the matrix solvers shows the algorithms to have scalable properties as the problem size grows indefinitely large. Speed-up and efficiency values are presented along with experimental and theoretical values for the optimum number of processors for maximum speed-up. It is envisaged that the parallel techniques presented here have applications using other boundary integral methods for solving engineering problems of a more complex nature.

KEY WORDS Parallel processing Panel methods Conjugate gradients Computational fluid dynamics

1. INTRODUCTION

For any design process incorporating a computational tool, interactiveness is an important criterion. In order to solve problems involving fluid flow around complex bodies within a reasonable time period, both the physical properties of the surrounding fluid require some form of approximation and the computational hardware must be fast in some sense. At present, currently available computational hardware is of insufficient power to provide interactiveness for the solution of the full Navier–Stokes equations around complex body shapes. In the initial stages of design, quick response times tend to be the important parameter, with the body shape changing frequently and in some cases radically. A possible way of providing this is firstly to assume that the fluid is both incompressible and irrotational and secondly to utilize the new and increasingly fast computational power of parallel processing. During the past few years there has been a rapid rise of parallel mechines both for shared memory architectures such as the Alliant FX series and for distributed memory such as the Meiko Computing Surface. This has meant that a need exists for fully scalable parallel software to complement these machines.

Considerable interest has been taken in parallel algorithms for the solution of large matrix systems and the reader is directed to the publication by Ortega.[1] In the majority of cases this interest has concentrated on iterative and direct schemes for sparse matrices. Boundary integral methods differ from the more common solution methods of finite element and finite difference in their production of fully populated matrices. The potential role in solving boundary element formulations for multiple-processor computers and vector machines is, as far as the authors are aware, only just being addressed.[2] Fiddes and Chalmers have, however, provided a demand-driven algorithm for the solution of the panel method implemented on a Meiko Computing Surface.[3] The solution technique utilizes a minimum path length topology with each processor tasked independently as it becomes idle. Jacobi and Gauss–Seidel solution methods have been implemented in parallel to solve the resulting influence matrix.

0271–2091/92/010095–14$07.00

Received October 1990
Revised March 1991

The following work attempts to set out a simple yet effective parallel algorithm, using transputer-based distributive memory hardware, for the fast solution of potential flow problems from non-lifting bodies to fully discretized complex aerodynamic shapes. In addition, the parallel solution algorithm has the possibility of being utilized in the more general area of boundary integral methods where dense matrix systems predominate.

'Potential' flows can represent a remarkably good approximation to the Navier–Stokes equations for a large class of problems. The classic work of Hess and Smith[4] has provided the framework for the solution of potential flows around complex body shapes. Now well known as panel methods, this solution technique has been applied in industry to great effect, especially in the area of aeronautics.[5,6] The panel method, being a linear inviscid approximation to the full Navier–Stokes equations, is ideally suited to a first-stage design tool for flow visualization around complex bodies.[7] In the parallel algorithm outlined below, for reasons of simplicity and for the purposes of explanation, we have chosen only to use the simple constant-source panel method.

In addition to the panel method, the parallel algorithm has the possibility of being easily augmented to include solutions to a variety of engineering problems utilizing the boundary integral technique[8] for constant piecewise elements. In these more complex cases a simple conjugate gradient method will probably not suffice, hence requiring the utilization of more powerful conjugate direction methods. However, as the present study shows, the transputer topology and the basic parallelism are also applicable to these methods without any significant changes to the algorithm. The integration of conjugate gradient methods (even without pre-conditioning) and the panel method has been shown to work previously;[9] however, the following is intended to give a more in-depth investigation of the possible advantages of using conjugate direction methods from a parallel perspective. All computations for this study were carried out on a Meiko Computing surface utilizing a maximum of 23 processors.

2. PANEL METHOD THEORY

Fluid equations

The governing equations for irrotational incompressible flow are relatively straightforward:[10]

mass

$$\nabla \cdot \mathbf{v} = 0, \tag{1}$$

momentum

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p, \tag{2}$$

where p is the pressure, ρ is density and $\mathbf{v}$ denotes the fluid velocity at any point in the domain. Expressing $\mathbf{v}$ as the sum of a free stream velocity $\mathbf{v}_\infty$ and an irrotational perturbation velocity $\mathbf{q}$, i.e.

$$\mathbf{v} = \mathbf{v}_\infty + \mathbf{q}, \tag{3}$$

then by implementing the zero-vorticity assumption for potential flow, the fluid may be modelled by the Laplace equation

$$\nabla^2 \phi = 0, \tag{4}$$

with ϕ the potential function. Hence for a given complex body shape the problem has been reduced to one of finding the appropriate perturbation potential ϕ or velocity $\mathbf{q}$ subject to boundary conditions at infinity and at the body surface $\mathbb{S}$. By assuming that the surface is modelled by a set of discrete panels, on which there exists a singularity distribution, and using the

zero normal velocity boundary condition, we may derive a set of linear algebraic equations and hence find a flow field exterior (or interior) to the complex body.

Surface discretization

Shape representation in the form of *B*-spline curves and surface forms can be adopted to represent complex bodies with sculptured form. The generation of the panels for a given object may be based on the creation of a planar polyhedral approximation of the object boundary in which the polyhedral facets can be used as flow panels.[7] Using this type of integration, the body surface is easily discretized and the required solution (for the simple source example) is the evaluation of the strengths of each source positioned at the centroid of each panel. For certain 2D, axisymmetric and 3D test cases the body panels can be easily obtained from a closed-form solution. Also, the recent work by Bloor and Wilson[11,12] is particularly apt for surface discretization.

Matrix evaluation

Let $\mathbf{q}_{ij}$ be the velocity induced at the control point of the ith panel by a unit source density on the jth panel. This induced velocity may be evaluated using the normal derivative of the perturbation potential and integrating over the panel surface $\mathbb{S}_j$. For 2D the integral can be evaluated analytically; for the axisymmetric and 3D geometries the integral is numerically determined using only the spatial and geometrical information of the panels i and j, which is readily available.[4] The normal velocity induced at the ith control point by the singularity distribution at the jth panel is thus

$$A_{ij} = \mathbf{n}_i \cdot \mathbf{q}_{ij}, \tag{5}$$

where $\mathbf{n}_i$ is the unit normal vector of the ith panel and A_{ij} is termed the influence coefficient. From the boundary condition of a prescribed normal velocity on the body surface at each panel i we obtain the followng set of simultaneous equations.

$$\sum_{j=1}^{N} A_{ij}\sigma_j = -\mathbf{V}_{i\infty} \cdot \mathbf{n}_i + F_i, \tag{6}$$

a linear matrix system

$$\mathbf{A}\boldsymbol{\sigma} = \mathbf{b}.$$

Here σ_j is the source density at the control point of panel j and F_i is a known transpiration velocity at the control point of panel i. Clearly, from the form of the integral defining the perturbation velocity, the matrix $\mathbf{A}$ is dense and the method chosen for solution is a conjugate gradient algorithm. Its iterative properties are dependent on the eigenvalues of the matrix. For the panel method it can be shown[13] that the eigenvalues of the perturbation matrix, λ_i, are such that $\lambda_i \in [0, 4\pi]$. Preconditioning of the matrix can be used to reduce the spectral radius of the matrix and enhance convergence properties. At this stage preconditioning is left for the subject of a further paper.

Once a solution has been obtained, the velocity at each control point on the solid surface is evaluated by

$$\mathbf{q}_i = \sum_{j=1}^{N} \mathbf{q}_{ij}\sigma_j + \mathbf{q}_{i\infty}. \tag{7}$$

Velocities may now be evaluated at positions within the fluid surrounding the solid surface by

using similar algorithms to that used in the influence matrix evaluation. The off-body velocity is evaluated with the use of the equation

$$\mathbf{V}_i = \sum_{j=1}^{N} \mathbf{V}_{ij}\sigma_j + \mathbf{V}_{i\infty}. \tag{8}$$

Here the matrix $\mathbf{V}_{ij}$ is evaluated by the same method as for $\mathbf{q}_{ij}$; however, in this case the panel centroid position i is substituted by the spatial position at which the off-body velocity is to be calculated.

3. CONJUGATE GRADIENT METHOD

The basic algorithm[14] is shown below. Defining the residual $\mathbf{r}$ as $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$, then with $\mathbf{x}_i$ as the ith iterate

$$\mathbf{r}(\mathbf{x}_i) = \mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i. \tag{9}$$

We choose an initial value for $\mathbf{x}$, $\mathbf{x}_0$, and set $\mathbf{r}_0 = \mathbf{p}_0$. Thus for $k = 0, 1, \ldots, N$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{10}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k, \tag{11}$$

$$\alpha_k = \frac{\langle \mathbf{r}_k^{\mathrm{T}}, \mathbf{r}_k \rangle}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A}\mathbf{p}_k}, \tag{12}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k, \tag{13}$$

$$\beta_k = \frac{\langle \mathbf{r}_{k+1}^{\mathrm{T}}, \mathbf{r}_{k+1} \rangle}{\langle \mathbf{r}_k^{\mathrm{T}}, \mathbf{r}_k \rangle}. \tag{14}$$

The iteration is stopped when $|\mathbf{r}_{k+1}| < \varepsilon$, with ε specified *a priori*.

4. PROCESSOR TOPOLOGY AND PARALLEL ALGORITHMS

For the current problem we have chosen a linear topology as shown in Figure 1. This network supports the parallelization of the matrix–vector product which is the 'workhorse' of the conjugate gradient algorithm. In addition it provides an environment for the parallel matrix evaluation as described below.

For the panel method the largest amount of work resides in the evaluation of the marix $\mathbf{A}$, whilst for the conjugate gradient algorithm it resides with the vector product with $\mathbf{p}_k$, $\mathbf{A}\mathbf{p}_k$.

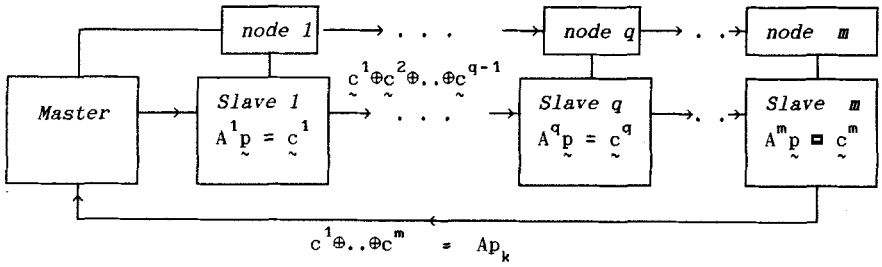

Figure 1. Processor topology and communication links between processors

Suppose we have a body consisting of N panels and a transputer topology of m processors such that m divides N. (In fact the current algorithm can take account of N mod $m \neq 0$ with the consequence of a minimal load imbalance; see later.) Since the matrix coefficients are evaluated simply on the basis of the geometric information of the panels, we may subdivide the matrix $\mathbf{A}$ into strips as shown below, each processor evaluating N^2/m coefficients.

$$\left[\begin{array}{l}
\begin{array}{llll} a_{1,1} & a_{1,2} & \cdots\cdots & a_{1,N} \\ \vdots & & & \\ a_{k,1} & a_{k,2} & \cdots\cdots & a_{k,N} \end{array} \\
\begin{array}{llll} a_{k+1,1} & a_{k+1,2} & \cdots\cdots & a_{k+1,N} \\ \vdots & & & \\ a_{2k,1} & a_{2k,2} & \cdots\cdots & a_{2k,N} \end{array} \\
\vdots \\
\begin{array}{lll} a_{(q-1)k+1,1} & \cdots\cdots & a_{(q-1)k+1,N} \\ \vdots & & \\ a_{qk,1} & \cdots\cdots & a_{qk,N} \end{array} \\
\vdots \\
\begin{array}{lll} a_{(m-1)k+1,1} & \cdots\cdots & a_{(m-1)k+1,N} \\ \vdots & & \\ a_{N,1} & \cdots\cdots & a_{N,N} \end{array}
\end{array}\right] = \mathbf{A}. \tag{15}$$

Here $N/m = k$.

Initially all the nodes receive, by broadcast link, the geometric data of all panels. Each processor then evaluates coefficients corresponding to its position in the matrix. The initial vector $\mathbf{p}_0$ is broadcast to all processors and the conjugate gradient iteration algorithm is then initiated.

Consider the qth strip of the matrix and its product with the conjugate vector $\mathbf{p}$. For a submatrix denoted by $\mathbf{A}^q$ we have that

$$\mathbf{A}^q\mathbf{p} = \left[\begin{array}{lll} a_{(q-1)k+1,1} & \cdots\cdots & a_{(q-1)k+1,N} \\ \vdots & & \\ a_{qk,1} & \cdots\cdots & a_{qk,N} \end{array}\right] \left[\begin{array}{l} p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_N \end{array}\right] = \mathbf{c}^q. \tag{16}$$

Using the submatrix–vector product as shown above, processor q has therefore stored the vector $\mathbf{c}^q$. It receives from the $(q-1)$th processor the accumulated vector $\mathbf{c}^1 \oplus \mathbf{c}^2 \oplus \ldots \oplus \mathbf{c}^{q-1}$.

Collating this into the correct order, the qth processor transmits to the $(q+1)$th processor the larger vector $\mathbf{c}^1 \oplus \mathbf{c}^2 \oplus \ldots \oplus \mathbf{c}^q$ as shown in Figure 1. The operator $\oplus$ is defined as

$$\mathbf{c}^i \in \mathbb{R}^i, \qquad \mathbf{c}^j \in \mathbb{R}^j,$$

then

$$\mathbf{c}^i \oplus \mathbf{c}^j \in \mathbb{R}^{i+j}, \tag{17}$$

so that

$$\mathbf{c}^i \oplus \mathbf{c}^j = [c_1^i, c_2^i, \ldots, c_i^i, c_1^j, \ldots, c_j^j]. \tag{18}$$

We note here that the qth process only transmits the minimum amount of information to the $(q+1)$th node, since the vector length of $\mathbf{c}^q$ stored at the qth node is $qk < n$.

The master process now receives the full vector $\mathbf{Ap}$ from the end node m and evaluates the three linked triads and the two constants α_k and β_k from equations (1)–(5) to find $\mathbf{r}_{k+1}$, $\mathbf{p}_{k+1}$ and $\mathbf{x}_{k+1}$. Then $\mathbf{p}_{k+1}$ is broadcast back to the m nodes and the process is repeated. A flow diagram of this algorithm is given in Figure 2.

The evaluation of the influence conditions for the panel method clearly do not produce symmetric linear systems, and although conjugate gradient methods have been used previously to find solutions to potential flow problems over complex body shapes,[9] convergence certainly cannot be guaranteed. Indeed, the assumption of the algorithm being a direct method is based

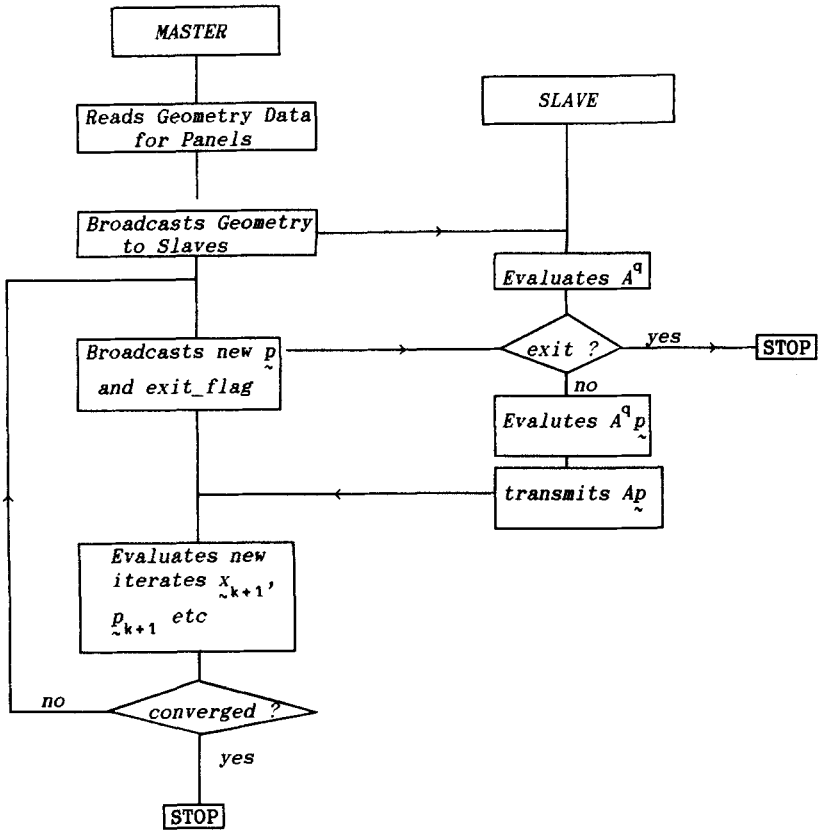

Figure 2. Flow diagram for parallel algorithm of conjugate gradient method

primarily on the matrix **A** being symmetric positive definite; however, that being the case, the above topology and algorithm may still be used for the biconjugate gradient method[15] which has been proposed for non-symmetric systems. The algorithm is given below: $\mathbf{r}_0 = \mathbf{p}_0$ as before and $\hat{\mathbf{r}}_0 = \hat{\mathbf{p}}_0$, with $\hat{\mathbf{r}}_0$ an arbitrarily chosen vector normally evaluated as $\hat{\mathbf{r}}_0 = \mathbf{I}\mathbf{r}_0$;

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{19a}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k, \tag{19b}$$

$$\hat{\mathbf{r}}_{k+1} = \hat{\mathbf{r}}_k - \alpha_k \mathbf{A}^{\mathrm{T}} \hat{\mathbf{p}}_k, \tag{19c}$$

$$\alpha_k = \frac{\langle \hat{\mathbf{r}}_k^{\mathrm{T}}, \mathbf{r}_k \rangle}{\hat{\mathbf{p}}_k^{\mathrm{T}} \mathbf{A}^{\mathrm{T}} \mathbf{p}_k}, \tag{19d}$$

$$\mathbf{p}_{k+1} = \hat{\mathbf{r}}_{k+1} + \beta_k \mathbf{p}_k, \tag{19e}$$

$$\hat{\mathbf{p}}_{k+1} = \hat{\mathbf{r}}_{k+1} + \beta_k \hat{\mathbf{p}}_k, \tag{19f}$$

$$\beta_k = \frac{\langle \hat{\mathbf{r}}_{k+1}^{\mathrm{T}}, \mathbf{r}_{k+1} \rangle}{\langle \hat{\mathbf{r}}_k^{\mathrm{T}}, \mathbf{r}_k \rangle}. \tag{19g}$$

All that is required in this case is the additional evaluation of the matrix–vector product $\mathbf{A}^{\mathrm{T}}\hat{\mathbf{p}}$ with $\hat{\mathbf{p}}$, a conjugate search vector, and the additional calculation of the vector triad to form the next search vector iterate $\hat{\mathbf{p}}_{k+1}$ from $\hat{\mathbf{p}}_k$. By utilizing the information of the submatrix $\mathbf{A}^q$ that the qth processor has available, the full vector $\mathbf{A}^{\mathrm{T}}\hat{\mathbf{p}}$ may be subdivided amongst the processors by storing the partially summed vector $(\mathbf{A}^q)^{\mathrm{T}}\hat{\mathbf{p}}^q$ as shown below:

$$(\mathbf{A}^q)^{\mathrm{T}} \delta\hat{\mathbf{p}} = \begin{bmatrix} a_{(q-1)k+1,1} & \cdots & a_{qk,1} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ q_{(q-1)k+1,N} & \cdots & a_{qk,N} \end{bmatrix} \begin{bmatrix} \hat{p}_{(q-1)k+1} \\ \vdots \\ \vdots \\ \hat{p}_{qk} \end{bmatrix} = \delta\hat{\mathbf{c}}^q, \tag{20}$$

with $\delta\hat{\mathbf{c}}^q \in \mathbb{R}^N$, so that $\mathbf{A}^{\mathrm{T}}\hat{\mathbf{p}}$ is given by

$$\mathbf{A}^{\mathrm{T}}\hat{\mathbf{p}} = \sum_{q=1}^{m} \delta\hat{\mathbf{c}}^q. \tag{21}$$

The same communications process is used by the biconjugate algorithm as that used for the conjugate gradient method. However, the vector transmitted to the qth processor from the $(q-1)$th is of length N and, once received, the qth processor simply adds on the partial sum $\delta\hat{\mathbf{c}}^q$ so that the end processor can transmit the full vector $\mathbf{A}^{\mathrm{T}}\hat{\mathbf{p}}$ back to the master process.

Once a solution has been obtained, off-body velocities may be evaluated. This calculation can also be implemented in parallel by dividing the fluid domain $\mathscr{D}$ into subdomains such that

$$\mathscr{D} = \mathscr{D}_1 \cup \mathscr{D}_2 \cup \ldots \mathscr{D}_q \cup \ldots \mathscr{D}_m,$$

where these subdomains may be either exterior or interior to the body surface depending on the problem being solved. Since each processor has stored the entire geometry data of the solid body, no communication between processors is required and the velocities for each subdomain may be calculated in parallel. This type of evaluation may be particularly useful for field integral methods in subsonic regions such as that described by Sinclair.[16]

5. TIME COMPLEXITY ANALYSIS

Here we follow the analysis originally done by Gropp and Smith.[17] It can be seen from the above that the evaluation of the matrix of influence coefficients is implemented in perfect parallelism, since no interprocessor communication is required. Initially we ignore the matrix coefficient evaluation for the purposes of analysing the scalability of the algorithm. With the topology chosen and the conjugate gradient algorithm as a solution procedure we assume that the time taken for a conjugate gradient iteration is split into two disjoint parts:

(i) evaluation of the submatrix–vector product $\mathbf{A}^q\mathbf{p}_k$, the linked triads and the associated inner products
(ii) the communication time for broadcasts and pulsed accumulation of the vector $\mathbf{Ap}$, both between node and slave and between processors.

For the communication time we assume that there exists a start-up time γ and a data transfer rate α; these parameters are non-dimensionalized using the floating point speed of the processor. To model the communication times, we define four different types of communication: type 1, that of communication between the master process and node 1; type 2, between node $(q-1)$ and node q; type 3, between node q and slave q; type 4, between slave $(q-1)$ and slave q. Subscripts 1–4 will be used to denote the corresponding start-up and transfer rates γ_i and α_i. For the presented linear topology, types 1 and 3 are between *processes*, types 2 and 4 are between *processors*. It is expected that 'on-board' communication (types 1 and 3) will be faster than interprocessor communication (types 2 and 4).

The iteration is initiated by the master process transmitting the search vector $\mathbf{p}_k$ to node 1. Node 1 in turn transmits this information to the slave 1 process before passing the data to node 2. In general, node q passes the data to the slave q process before transmitting to the next processor node. After each slave process has received the vector $\mathbf{p}_k$, it evaluates the submatrix–vector product $\mathbf{A}^q\mathbf{p}_k$ and transmits the accumulated vector $\mathbf{c}^1 \oplus \ldots \oplus \mathbf{c}^q$ to the slave $(q+1)$ process. Finally, the slave m process receives the $\mathbf{c}^1 \oplus \ldots \oplus \mathbf{c}^{m-1}$ accumulated vector and transmits the full vector $\mathbf{Ap}_k$ to the master process. In order to model the iteration time as a function of the problem size N and the number of processors m, we assume that the work done by each processor in evaluating $\mathbf{A}^q\mathbf{p}_k$ is equal and there exists perfect load balancing. Since each slave process receives the search vector $\mathbf{p}_k$ in numerical order, slave process j will therefore have evaluated $\mathbf{A}^j\mathbf{p}_k$ before process $j+1$. Each slave process is thus ready to receive the accumulated vector from the left and the rate-limiting step becomes the transmission of the accumulated vector $\mathbf{c}^1 \oplus \ldots \oplus \mathbf{c}^{m-1}$ to slave m and its subsequent passing of the full vector back to the master process. The evaluation of the general submatrix–vector product $\mathbf{A}^q\mathbf{p}_k$ can effectively be ignored for all the slave processes apart from the end process in the modelling of the time per iteration, since this is done in parallel owing to the order in receiving the search vector $\mathbf{p}_k$ and the assumption of perfect load balancing.

For a problem of N panels and m processors we have that the time for a single conjugate gradient iteration is

$$\tau(m, N) = 3\gamma_1 + N\alpha_1 + (m-1)(\gamma_2 + N\alpha_2) + m(\gamma_3 + N\alpha_3) + 2(\gamma_4 + N\alpha_4) + \frac{AN^2 + BN}{m} + CN + D. \tag{22}$$

In the above model we have used non-dimensionlized constants A, B, C and D to take account of loop start-up and counter times for the work done in each slave process. These constants were evaluated using small independent timing programmes and non-diminisionalized using the

floating point speed of the processor. Also, we have taken the worst case in communicaton, i.e. sending a full vector to the end slave process, but the true length of the actual vector transmitted will vary little from that modelled. In addition, the master computations are evaluated whilst the slave processes are in idle time, since they require the new conjugate direction search vector $\mathbf{p}_k$ before continuing their next iteration. In order to model this, we have effectively added the master and slave work together. As can be seen, as N gets larger, the communication costs are a smaller fraction of the overall work per processor. However, for a constant problem size, with both N and m increasing in size such that $N^2/m = \text{constant}$, the communication time and time taken to evaluate the linked triads will be an important factor.

We may now use Gropp and Smith's ideas further to analyse the algorithm, in particular as N becomes indefinitely large. Since the time function $\tau(m, N)$ includes an inverse term in m, then for a specific problem size there will be an optimal value for m above which the efficiency will begin to decrease. The maximum speed-up is achieved when

$$\frac{\mathrm{d}\tau}{\mathrm{d}m} = \gamma_2 + N\alpha_2 + \gamma_3 + N\alpha_3 - \frac{AN^2 + BN}{m^2} = 0, \tag{23}$$

and from this the optimal value of m is

$$m_{\text{opt}} = \sqrt{\left(\frac{AN^2 + BN}{\gamma_2 + \gamma_3 + N(\alpha_2 + \alpha_3)}\right)}. \tag{24}$$

As N gets progressively larger, m_{opt} tends to the value

$$m_{\text{opt}} \to \sqrt{\left(\frac{AN}{\alpha_2 + \alpha_3}\right)}. \tag{25}$$

By using the optimal value (equation (24)), we may evaluate the maximum speed-up S_m obtainable for a specific problem size, i.e. N^2/m. Initially we write the speed-up as

$$S_m(m, N) = \frac{\tau(1, N)}{\tau(m, N)} \tag{26}$$

and the optimal speed-up therefore becomes

$$S_{m_{\text{opt}}}(N) = \frac{AN^2 + N\psi_3 + \phi_3}{\phi_2 + N\psi_2 + 2\sqrt{[(AN^2 + BN)(\phi_1 + N\psi_1)]}}, \tag{27}$$

where

$$\begin{aligned} \phi_1 &= \gamma_2 + \gamma_3, & \psi_1 &= \alpha_2 + \alpha_3, \\ \phi_2 &= 3\gamma_1 + \gamma_2 + D, & \psi_2 &= \alpha_1 - \alpha_2 + C, \\ \phi_3 &= 4\gamma_1 + \gamma + D, & \psi_3 &= 2\alpha_1 + \alpha_3 + C + B. \end{aligned}$$

It is easy to show that for N increasing without bound,

$$\lim_{N \to \infty} S_{m_{\text{opt}}}(N) = \frac{1}{2}\left(\frac{AN}{\psi_1}\right)^{1/2} = \tfrac{1}{2} \lim_{N \to \infty} (m_{\text{opt}}) \tag{28}$$

to leading order, showing that the speed-up has no upper bound for the problem size growing indefinitely large. The ratio of slave work to communication time between slaves provides the coefficient for speed-up, indicating, as is intuitive, that faster communication times will significantly enhance the solution times. Clearly, memory restraints on each transputer will provide the physical restriction to this limit.

The above model assumes that m divides N. If this is not the case, the algorithm tries to balance the load such that each processor has in excess no more than N elements (i.e. a single row of the matrix) to work on compared to any other. Suppose N mod $m=q$. Then the first q processors work on $N/m+1$ rows and the $m-q$ processors work on the remaining N/m rows. Tests have shown that this type of load imbalance does not significantly degrade speed-up.

For the bi-conjugate gradient method we may write a similar equation relating the time taken for a single iteration step as a function of problem size N and number of processors m:

$$\tau(m, N)=3\gamma_1+2N\alpha_1+(m-1)(\gamma_2+2N\alpha_2)+m(\gamma_3+2N\alpha_3)+(\gamma_4+2N\alpha_4) + \frac{AN^2+BN}{m}+EN+F+\frac{AN^2}{m}+GN. \tag{29}$$

For large enough N the communication time is twice that for the conjugate gradient algorithm and the only difference lies in the work done per processor plus some additional vector additions by the master process. This yields the optimum number of processors for the algorithm as a function of N as

$$m_{\text{opt}}=\sqrt{\left(\frac{2AN^2+BN}{\phi_1+2\psi_1 N}\right)}. \tag{30}$$

This value is identical to that evaluated for the conjugate gradient algorithm as N grows indefinitely large.

6. RESULTS AND DISCUSSION

Simple timing tests were carried out to evaluate the communication constants γ_i and α_i. These are shown below in their appropriate non-dimensionalized form:

$$\gamma_1=\gamma_2=\gamma_3=\gamma_4=10, \qquad \alpha_1=\alpha_3=0{\cdot}16, \qquad \alpha_2=\alpha_4=1{\cdot}9.$$

As a preliminary test case an axisymmetric body immersed in a cross-flow of specified angle β was chosen. The test body was a sphere and the onset flow U_∞ has unit magnitude with $\beta=\pi/4$. Ellipsoids were also used but timings for the solution were almost identical to those of the sphere and are not shown here.

Speed-up S_m and efficiency E are defined as

$$S_m=\frac{\text{execution time for a single processor}}{\text{execution time for } m \text{ processors}}, \tag{31}$$

$$E=\frac{S_m}{m}. \tag{32}$$

In the above we have deviated from the more usual definition of S_m (i.e. using the best serial algorithm) by utilizing the time taken for a single processor running both master and slave processes. This is due to the differences in operating system when running a serial programme to that for a parallel programme. If the more usual definition of S_m is used, then efficiency values become *greater than unity* and hence do not provide a representative statement of parallel efficiency.

Table I shows the speed-up and efficiency as a function of the number of processors for the axisymmetric sphere composed of a number of panels, namely 100, solved using the conjugate gradient method. The total number of iterations required for a solution with $\varepsilon=10^{-3}$ was three using an initial value of the source strengths of zero.

Table I

m (number of processors)	Constant N		Constant N^2/m	
	S_m	E	S_m	E
3	2·86	0·95	2·96	0·99
4	3·83	0·96	3·96	0·99
5	4·73	0·95	4·92	0·98
6	5·50	0·92	5·88	0·98
7	6·15	0·88	6·85	0·98
8	6·96	0·87	7·70	0·96
9	7·44	0·83	8·63	0·96
10	8·50	0·85	9·61	0·96
11	8·59	0·78	10·41	0·95

To place the results given in Table I in context, the times taken for a single transputer running both master and slave processes for 100 and 250 panels were 21·6 and 133·8 s respectively. The efficiency figures are encouragingly high and this is due to the large time spent in evaluating the influence matrix **A** compared to that of the conjugate gradient iterations. A performance analysis of the code showed that for the above axisymmetric case only 3·5% of the total computation time was spent in iterations. For problems involving larger numbers of iterations this figure will clearly increase, but the dominant part of the CPU time will almost always be spent in matrix evaluations. Table I also shows the speed-up and efficiency values for a constant grain size, i.e. $N^2/m = 10^4$. Here again the efficiencies maintain high values, principally owing to the large proportion of the time taken for solution in evaluating the influence matrix.

Tests were done to provide experimental evidence for values of m_{opt}. For each problem size (constant N and constant number of iterations) timings were measured for increasing numbers of processors, thus enabling a minimum time to be evaluated and hence an optimum number of processors for the iteration stage of the algorithm. Table II shows the comparison between the theoretical value for m_{opt} and the optimum value found from computations using the axisymmetric sphere test case. The results show excellent agreement with theory.

Although only a small percentage of the total computation time, single-iteration time results were obtained for various values of N and m using the linear topology. These showed that for high numbers of processors (22) the time for each iteration cycle varied between 0·2 s (500 panels) and 2·9 s (3000 panels).

It seems interesting to note the apparent robustness of the conjugate gradient algorithm used with the panel method. As mentioned in Section 1, non-preconditioned CG metods have been successfully employed on quite complex structures, in the case of Reference 9 a full helicopter body. The method presented in this paper has also been used with great success on several problems, notably that of the simulation of the flow of bone marrow surrounding an aspiration needle immersed in trabecular bone.[18] This involved a difficult rectangular shape to be simulated in addition to non-zero normal velocity boundary conditions. Figure 3 shows a 2D example using multiple bodies. The onset flow is provided by a uniform stream from left to right with a single vortex situated at the quarter-chord point of the aerofoil. The number of panels required to simulate the flow was approximately 400 and no preconditioning was required. That being said, preconditioning of the matrix will be needed in the future to ensure convergence for the large variety of industrial problems encountered.

Table II

N (number of panels)	m_{opt}(theory)	m_{opt}(exp)
100	10·1	10
200	14·6	13
300	18·0	17
400	21·9	20
500	23·4	$\geqslant 22$
600	25.7	—
700	27·8	—
800	29·7	—
900	31·6	—
1000	33·3	—

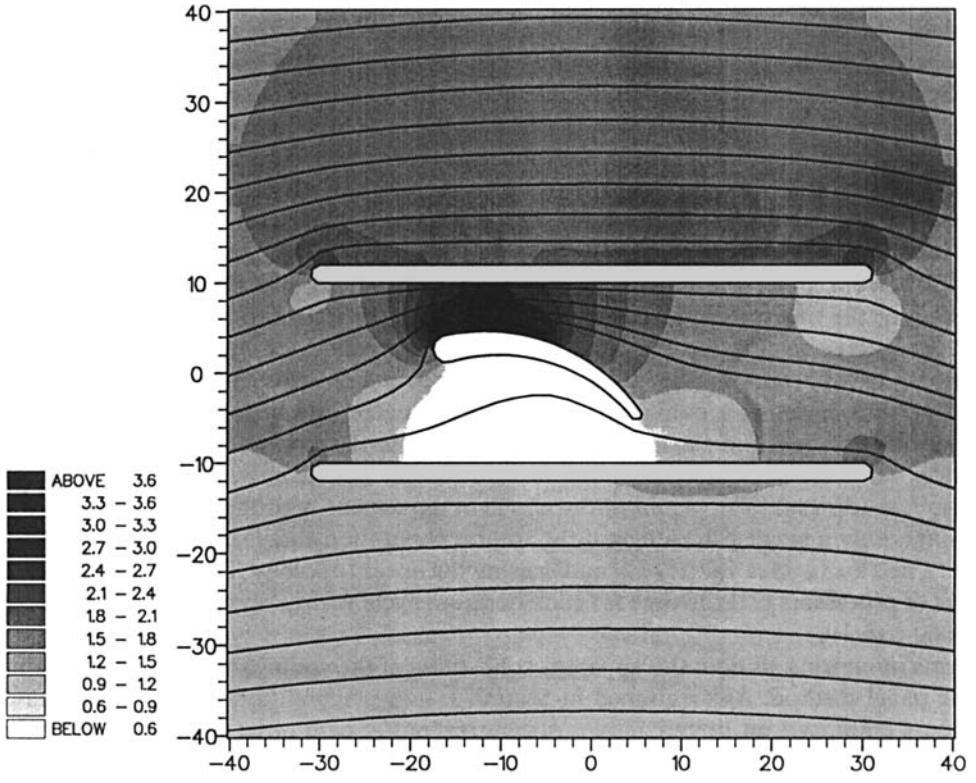

Figure 3. Flow around aerofoil in a duct with constant-velocity contours and streamlines

In order to show that the algorithm could provide reasonable 'turnaround' times for large values of N, tests were carried out on flows past an ellipsoid of revolution with a major/minor axis ratio of 8 : 1. Table III shows values of the times (in seconds) taken for the total calculation, τ_t, the symmetric flow matrix, τ_1, the cross-flow matrix, τ_2, and a single CG iteration, τ_i, for values of N up to 3000. In this case the number of processors was held constant at 22.

Table III

N	τ_t	τ_1	τ_2	τ_i
1000	156	45	46	0·54
1500	261	103	104	0·98
2000	419	181	198	1·5
3000	929	414	447	2·9

In comparing these results with other panel method implementations such as that of Fiddes and Chalmers[3] care must be taken in that for the axisymmetric test case

(i) two independent $N \times N$ matrices are evaluated per solution
(ii) numerical integration is used to evaluate all influence coefficients, apart from the diagonals, whereas simple analytic element evaluations are used for both 2D and 3D cases.

In addition, allowance must be made for the different number of processors used.

For the axisymmetric and two-dimensional geometries the number of panels needed for a reasonably good simulation is small enough for the on-board transputer memory to hold all the coefficients of the influence marix $\{a_{ij}\}$. In the fully 3D case a significantly larger number of panels are needed. However, only a small percentage of these coefficients need be stored; the rest may be recalculated at each iteration stage using simple multipole expansion approximations.[4] Assuming a small number of iterations (compared to N) are required for solution, the increased computation will have the effect of increasing the value of A in the complexity analysis. New vector processors such as the Intel i860 in conjunction with transputers provide a considerable increase in Mflop rates, especially when used with vectorizing compilers.[19] Hence the increased work per node would not seem to be a significant problem. By utilizing the 'freed-up' memory, a substantial increase in the problem size may be attempted. In addition, the processors used in the present method had only 4 Mbytes on-board memory. This is not an upper limit for this type of hardware (48 Mbytes are available at present) and the next generation of transputers will probably have the capability for memory sizes an order-of-magnitude higher than this.

7. CONCLUSIONS

The present study has shown that a simple linear topology of distributed memory processors can provide an environment for fast and highly efficient parallel algorithms using conjugate direction methods. Significant speed-ups can be achieved by parallelizing the matrix evaluation such that each processor evaluates only a submatrix. This is achieved by the fact that the matrix coefficients are independent of each other and rely only on the boundary geometry. The resulting matrices are dense and large (order N^2), so that by subdividing the matrix, significant increases in problem size may be attempted. This type of simple parallelization may also be used for other boundary element formulations and is not restricted to panel methods.

Similar advantages for parallelism become apparent when considering the computation of vector and scalar variables at external (internal) points within the computational domain. Here the domain is divided into subdomains, with each processor responsible for a particular subdomain. Task-farming algorithms may be used for this type of problem; however, this seems inappropriate at present with such a simple topology, since idle processors would require information from the master process and thus the passing of that information through all node processes to the left of that processor. Calculations for the evaluation of off-body velocities and

pressure values may be done in parallel, with a minimum of communication required to collate the data for transmission to the master process.

Domain integrals may be evaluated in parallel in the same manner and this has particular use for problems involving subsonic potential flows,[6] where field integrals are required as part of an iterative process, and for the study of convective problems in laminar flow.[20]

The parallel algorithms for the conjugate and biconjugate gradient methods have been shown to be scalable insofar as maximum speed-up is attained for specific values of the number of processors used, m_{opt}, increasing as $N^{1/2}$, with N the problem size. Additionally, the speed-up also increases as $N^{1/2}$. Results have shown that for a constant grain size N^2/m, efficiencies of the order of 95%–99% are attainable. This is due to the large computational time taken in the matrix evaluation; however, it seems that for an increasing proportion of time taken in the conjugate matrix solver, efficiencies will still be high if the optimum value of processors is made available.

For most problems solved using panel methods the value of N is large (order 10^3) and in these circumstances the number of processors available will rarely exceed $N^{1/2}$, so that the maximum speed-up may be achieved by simply evenly distributing the matrix evaluation work across the linear array of processors.

ACKNOWLEDGEMENT

Our thanks are due to Robert Lewis, an undergraduate in the Department of Mechanical Enginering, Universicy of Leeds for his work in producing the results of Figure 3.

REFERENCES

1. J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, New York, 1988.
2. P. Melli and C. A. Brebbia, *Supercomputing in Engineering Structures*, Springer, New York, 1989.
3. S. Fiddes and A. G. Chalmers, 'Parallel processing for panel methods', *Parallel Processing for Fluid Flow, Methods and Systems*, The Royal Institute, London, 27 June 1990, pp. 85–94.
4. J. L. Hees and A. M. O. Smith, 'Calculation of potential flow about arbitrary bodies', *Prog. Aeronaut. Sci.*, **8**, 1–138 (1967).
5. J. A. H. Petrie, 'Development of an efficient and versatile panel method for aerodynamic problems', *Ph.D. Thesis*, University of Leeds, March 1979.
6. P. M. Sinclair, 'A field integral method for the calculation of transonic flow over complex aerodynamic shapes', *Ph.D. Thesis*, University of Leeds, August 1987.
7. T. David, P. H. Gaskell and A. Saia, 'Integrating sculptured surface design with the panel method for flow visualisation', in D. C. Handscomb (ed.), *Mathematics of Surfaces III*, Clarendon, Oxford, 1989, pp. 301–315.
8. C. A. Brebbia, *The Boundary Element Method for Engineers*. Pentech, London, 1978.
9. J. Ryan, T. H. Lê and Y. Morchoisne, 'Panel code solvers', *Notes Numer. Fluid Mech.*, **20**, 335–342 (1988).
10. G. K. Batchelor, *An Introduction to Fluid Mechanics*, Cambridge University Press, Cambridge 1967.
11. M. I. G. Bloor and M. J. Wilson, 'Blend design as a boundary value problem', in W. Straßer and H. P. Seidel (eds), *Geometrical Modelling (Theory and Practice)*, Springer Berlin, 1989, pp. 221–234.
12. M. I. G. Bloor and M. J. Wilson, 'Using partial differential equations to generate free form surfaces', *CAD*, **22**, 202–212 (1990).
13. O. D. Kellogg, *Foundations of Potential Theory*, Dover, New York, 1929.
14. W. D. Joubert and T. A. Manteuffel, 'Iterative methods for non-symmetric linear systems', in D. R. Kincaid and L. J. Hayes (eds), *Iterative Methods for Large Linear Systems*, Academic, New York, 1990.
15. R. Fletcher, 'Conjugate gradient methods for indefinite systems', in *Lecture Notes in Mathematics, Vol. 506*, 1975, pp. 73–89.
16. P. M. Sinclair, *Aeronautical Journal of Royal Aeronautical Society Paper No. 1394*, June/July 1986.
17. W. D. Gropp and E. B. Smith, 'Computational fluid dynamics on parallel processors', *Comput. Fluids*, **18**, 289–304 (1990).
18. J. Rijcken, 'A computational model of the flow of bone marrow in the vicinity of a needle', *Final Year Report*, Department of Mechanical Engineering, University of Leeds, April 1990.
19. N. Miller, 'The wave equation, case study of a parallel i860 application', *Parallelogram*, (32) 14–16 (1990).
20. P. Skerget and C. Brebbia, 'The solution of convective problems in laminar flow', in *Boundary Elements, Proc. 5th Int. Conf. on Boundary Elements*, Springer, New York, 1983, pp. 251–274.