Computational Linear Algebra Issues in the Galerkin Boundary Element Method

O.O. Ademoyero, M.C. Bartholomew-Biggs, A.J. Davies

Numerical Optimisation Centre, Mathematics Department
University of Hertfordshire

Abstract

This paper deals with the symmetric linear systems of equations arising in the Galerkin boundary element method. In particular we consider the merits of direct and iterative solvers and present some numerical results which illustrate the way that solution costs vary with the number of boundary elements and indicate the possible advantages of iterative techniques (such as conjugate gradients) over direct (Gaussian elimination type) approaches.

The first part of the paper is concerned with sequential implementations of the Galerkin boundary element method. We shall also present results from a parallel implementation, running on an *nCUBE* machine. We shall consider the speed-ups obtained and for this purpose it will be instructive to consider separately the three phases: (1) constructing the linear system; (2) solving the linear system; and (3) using the results to compute interior solutions.

Our results show clearly the benefits of parallel implementation, but they also demonstrate that these benefits may not be uniform across all aspects of the calculation.

This paper was presented at the conference IMSE2000, held in Banff, Alberta June 12-15, 2000.

1 Boundary integral equations

The two-dimensional mixed potential problem may be written in the form

$$\nabla^2 u = 0 \qquad \text{in} \quad D \tag{1}$$

subject to the boundary conditions

$$u = u_0 \ \text{ on } \ C_0 \quad \text{and} \quad q \equiv \frac{\partial u}{\partial n} = q_1 \ \text{ on } \ C_1 \tag{2}$$

where D is the region bounded by the closed curve, $C = C_0 + C_1$ as shown in Figure 1.

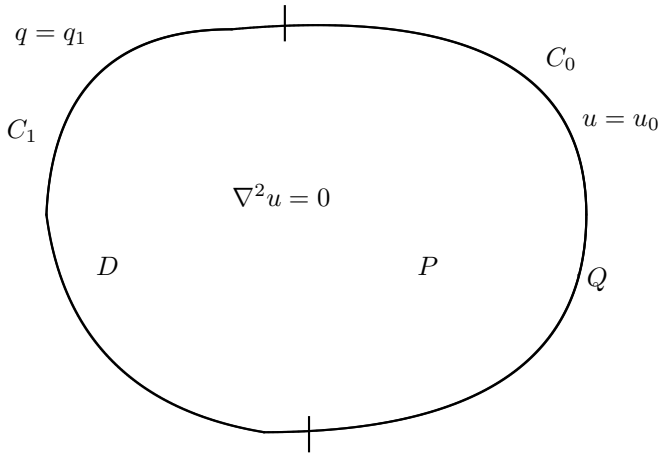

Figure 1: Potential problem in the region D.

For a point, P, in D we can write the potential at P in the form of a boundary integral [1], using the notation of Gray [2]:

$$\mathcal{P}(P) \equiv u(P) + \oint_C \left(u(Q) \frac{\partial G}{\partial n}(P,Q) - G(P,Q) q(Q) \right) dQ = 0 \tag{3}$$

where

$$G(P,Q) = -\frac{1}{2\pi} \ln |Q - P| = -\frac{1}{2\pi} \ln R$$

is the fundamental solution (Green's function) and $\mathbf{n} = \mathbf{n}(Q)$ is the unit outward normal on C at Q.

For properly-posed problems only one of u or q is known on C so that equation (3) is not directly of use as it stands. The usual approach [1] is to develop a boundary integral equation by considering the point P as a boundary point and to obtain the boundary integral equation by 'excluding' P with a small disc and taking the limit as the disc radius tends to zero to give

$$\alpha(P) u(P) + \oint_C \left(u(Q) \frac{\partial G}{\partial n}(P,Q) - G(P,Q) q(Q) \right) dQ = 0 \tag{4}$$

where $\alpha(P)$ is the so-called 'free-term' coefficient. Equation (4) is often called the potential boundary integral equation. Similarly the flux boundary integral equation is developed in the form [3]

$$\beta(P) q(P) + \oint_C \left(u(Q) \frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q) q(Q) \right) dQ = 0 \tag{5}$$

where $\mathbf{N} = \mathbf{N}(P)$ is the unit outward normal on C at P. These equations are well-established and have been studied for quite some time. However, there are considerable worries about the existence

of the integrals. Equation (4) involves a weakly singular part, due to G, and a strongly singular part, due to $\frac{\partial G}{\partial n}$. Equation (5) includes a strongly singular part, due to $\frac{\partial G}{\partial N}$, and a hypersingular part, due to $\frac{\partial^2 G}{\partial N \partial n}$. In fact the strongly singular parts are handled in the Cauchy principal value sense and the hypersingular part is usually handled in terms of a Hadamard finite-part integral in which it is assumed that divergent terms from neighbouring regions cancel. This causes particular difficulties when collocation is used to develop the system equations. Gray [2], suggests an alternative approach via the Galerkin method. We develop our argument in just the same way. We consider the potential and flux integrals for points P in D as follows:

$$\mathcal{P}(P) \equiv u(P) + \oint_C \left(u(Q) \frac{\partial G}{\partial n}(P,Q) - G(P,Q) q(Q) \right) dQ = 0 \tag{6}$$

$$\mathcal{F}(P) \equiv q(P) + \oint_C \left(u(Q) \frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q) q(Q) \right) dQ = 0 \tag{7}$$

where equation (7) is obtained by direct differentiation of equation (6) using the fact that we can reverse the order of integration and differentiation since the integrals in (6) are well-behaved. We now consider limiting values as the point P approaches the curve C.

2 Galerkin formulation

C_N

C

Figure 2: Boundary approximated by a polygonal curve.

We use the usual boundary element approximation in which the curve C is approximated by a piecewise curve, C_N. In our case C_N is taken as a polygon, as in Figure 2 and the boundary values of u and q are approximated by

$$\tilde{u}(Q) = \sum_{j=1}^{N} w_j(Q) u_j \quad \text{and} \quad \tilde{q}(Q) = \sum_{j=1}^{N} w_j(Q) q_j \tag{2. 1}$$

where $\{w_j(Q) : j = 1, 2, \cdots, N\}$ is a set of linearly independent basis functions. We shall consider linear elements in which the $w_j(Q)$ are the usual 'hat' functions. The boundary element formulation of equations (6) and (7) takes the form

$$\tilde{\mathcal{P}}_{C_N}(P) \equiv u(P) + \oint_{C_N} \left(\tilde{u}(Q) \frac{\partial G}{\partial n}(P,Q) - G(P,Q) \tilde{q}(Q) \right) dQ = 0 \tag{2. 2}$$

$$\tilde{\mathcal{F}}_{C_N}(P) \equiv q(P) + \oint_{C_N} \left(\tilde{u}(Q) \frac{\partial^2 G}{\partial N \partial n}(P,Q) - \frac{\partial G}{\partial N}(P,Q) \tilde{q}(Q) \right) dQ = 0 \tag{2. 3}$$

The limiting process proposed by Gray then sets equations (2. 2) and (2. 3) in the form

$$\lim_{\epsilon \longrightarrow 0} \tilde{\mathcal{P}}_{C_N}(P_\epsilon) = 0 \quad \text{and} \quad \lim_{\epsilon \longrightarrow 0} \tilde{\mathcal{F}}_{C_N}(P_\epsilon) = 0 \tag{2. 4}$$

where, as $\epsilon \rightarrow 0$, $P_\epsilon \rightarrow P_0$ on C_N. Finally, then, the Galerkin formulation is taken as

$$\oint_{C_N} w_k(P_0) \lim_{\epsilon \longrightarrow 0} \tilde{\mathcal{P}}_{C_N}(P_\epsilon) dP_0 \quad = \quad 0 \tag{2. 5}$$

$$\oint_{C_N} w_l(P_0) \lim_{\epsilon \longrightarrow 0} \tilde{\mathcal{F}}_{C_N}(P_\epsilon) dP_0 \quad = \quad 0 \tag{2. 6}$$

There is a variety of different types of integral in equations (2. 5) and (2. 6) depending on the elements in which P and Q are situated. The integrals may be either non-singular, weakly singular, strongly singular or hypersingular. The details of how these are handled are given by Gray [2] and we shall not repeat them here.

The importance of the Galerkin approach is that if equation (6) is used on that part of the boundary on which a Dirichlet condition holds and the *negative* of equation (7) is used on that part of the boundary on which a Neumann condition holds then the resulting boundary element algebraic equations are symmetric. The equations generated from (2. 5), (2. 6) may be written as

$$\mathbf{Hu} - \mathbf{Gq} = \mathbf{0} \tag{2. 7}$$

which, in block form, are

$$\begin{bmatrix} \mathbf{h}^{dd} & \mathbf{h}^{dn} \\ -\mathbf{h}^{nd} & -\mathbf{h}^{nn} \end{bmatrix} \begin{bmatrix} \mathbf{u}^d \\ \mathbf{u}^n \end{bmatrix} - \begin{bmatrix} \mathbf{g}^{dd} & \mathbf{g}^{dn} \\ -\mathbf{g}^{nd} & -\mathbf{g}^{nn} \end{bmatrix} \begin{bmatrix} \mathbf{q}^d \\ \mathbf{q}^n \end{bmatrix} = \mathbf{0}.$$

Here the partition superscripts indicate the distinction between the parts of the boundary on which Dirichlet and Neumann conditions hold. Since $\mathbf{u}^d$ and $\mathbf{q}^n$ are known, we can re-arrange the system so that only the unknown boundary values appear on the left. Thus if $\mathbf{x}$ denotes $(\mathbf{q}^d,\ \mathbf{u}^n)$ we can obtain the overall system of equations in the form

$$\mathbf{Ax} = \mathbf{b}$$

where

$$\mathbf{A} = \begin{bmatrix} -\mathbf{g}^{dd} & \mathbf{h}^{dn} \\ \mathbf{g}^{nd} & -\mathbf{h}^{nn} \end{bmatrix}. \tag{2. 8}$$

Green's function has the properties

$$\begin{aligned} G(P,Q) &= G(Q,P) \\ \frac{\partial G}{\partial n}(P,Q) &= \frac{\partial G}{\partial n}(Q,P) = -\frac{\partial G}{\partial N}(P,Q) = \frac{\partial G}{\partial N}(Q,P) \\ \frac{\partial^2 G}{\partial N \partial n}(P,Q) &= \frac{\partial^2 G}{\partial N \partial n}(Q,P) \end{aligned}$$

and by virtue of these it follows that $\mathbf{A}$ is symmetric.

3 Linear solvers for the Galerkin method

The $N \times N$ linear system **Ax** = **b** arising in the Galerkin method may be solved by a variety of methods. Of the direct approaches (i.e. those which transform or factorise the given system) the Gaussian elimination or Gauss-Jordan methods are always applicable. (We prefer the Gauss-Jordan approach for reasons which will be mentioned in the later section on parallelization.) Choleski factorization is a more efficient method (requiring only about half as much arithmetic and storage), but it requires **A** to be both symmetric and positive definite. While the symmetry of (2. 8) is assured there is no guarantee, unfortunately, that it will be positive definite. An iterative method which can be used to exploit the symmetry of the system without assuming positive definiteness is the method of conjugate gradients.

3.1 Conjugate gradient methods for Galerkin systems

The conjugate gradient (CG) approach, as first proposed by Hestenes and Stiefel [12], was intended for definite, rather than indefinite systems of equations. It proceeds via iterations of the form

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)} \tag{3. 1}$$

where the *search directions* $\mathbf{p}^{(0)},\ \mathbf{p}^{(1)}, ...\mathbf{p}^{(k)}, ..$ are constructed to satisfy the conjugacy property

$$\mathbf{p}^{(i)T}\mathbf{A}\mathbf{p}^{(j)} = 0 \quad \text{when } \ i \neq j. \tag{3. 2}$$

If $\alpha^{(k)}$ in (3. 1) is calculated by

$$\alpha^{(k)} = -\frac{\mathbf{p}^{(k)T}\mathbf{r}^{(k)}}{\mathbf{p}^{(k)T}\mathbf{A}\mathbf{p}^{(k)}} \tag{3. 3}$$

where $\mathbf{r}$ denotes the residual $\mathbf{A}\mathbf{x} - \mathbf{b}$ then it follows that

$$\mathbf{p}^{(k)T}\mathbf{r}^{(k+1)} = 0. \tag{3. 4}$$

This, when combined with the conjugacy property (3. 2), means that after k iterations

$$\mathbf{p}^{(j)T}\mathbf{r}^{(k+1)} = 0, \quad \text{for } \ j = 0, 1, .., k. \tag{3. 5}$$

This result implies *finite termination* — i.e. CG methods solve **Ax** = **b** in at most N iterations.

There are a number of ways of generating the conjugate sequence $\{\mathbf{p}^{(k)}\}$. The original CG algorithm takes $\mathbf{p}^{(0)} = -\mathbf{r}^{(0)}$ and then uses the two-term recurrence relation

$$\mathbf{p}^{(k+1)} = -\mathbf{r}^{(k+1)} + \beta^{(k)}\mathbf{p}^{(k)} \quad \text{with } \ \beta^{(k)} = \frac{\mathbf{r}^{(k+1)T}\mathbf{r}^{(k+1)}}{\mathbf{r}^{(k)T}\mathbf{r}^{(k)}}. \tag{3. 6}$$

This is only guaranteed to be stable when **A** is a positive- (or negative-) definite matrix. To deal with the indefinite case we can replace (3. 6) by a three-term recurrence relation [14]

$$\mathbf{p}^{(k+1)} = -\mathbf{A}\mathbf{p}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)} + \beta^{(k)}\mathbf{p}^{(k-1)} \tag{3. 7}$$

where $\alpha^{(k)},\ \beta^{(k)}$ are chosen so that

$$\mathbf{p}^{(k+1)T}\mathbf{A}\mathbf{p}^{(j)} = 0 \quad \text{for } \ j = k, k-1.$$

Other extensions of the CG method to cover the indefinite and non-symmetric case include the *conjugate residual* method and the *bi-conjugate gradient* approach. The former is equivalent to applying the CG method to the system $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$, while the latter is based on applying the CG method to a $2N \times 2N$ system of the form

$$\begin{pmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{x}} \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \tilde{\mathbf{b}} \end{pmatrix}.$$

A fuller account of CG methods in the context of boundary integral methods is given in [13].

The finite termination property ensures that the CG method is an $O(N^3)$ process, since the work on each iteration is chiefly the N^2 multiplications needed to form the matrix-vector product $\mathbf{A}\mathbf{p}^{(k)}$. This workload can be considerably reduced if **A** is sparse; but this is not relevant to us since the Galerkin approach typically generates dense matrices. However, even in the dense case, the CG method becomes more competitive if the eigenvalues of **A** are "bunched". Specifically it can be proved that, if **A** has only K distinct eigenvalues then convergence occurs in, at most, K iterations.

The preceding observation motivates the *pre-conditioned* CG method. In this approach we seek a matrix $\mathbf{M}$ such that $\mathbf{MAM}^T$ has eigenvalues more tightly grouped than the orginal matrix $\mathbf{A}$. The basic CG method can then be applied to solve the system $\mathbf{MAM}^T\mathbf{y} = \mathbf{Mb}$ after which we set $\mathbf{x} = \mathbf{M}^T\mathbf{y}$. A relatively simple choice for $\mathbf{M}$ is the so-called diagonal pre-conditioner

$$\mathbf{M} = \mathbf{diag}(\sqrt{|a_{ii}|}^{-1}). \tag{3. 8}$$

If we use (3. 8) the pre-conditioned matrix $\mathbf{MAM}^T$ has all diagonal elements equal to plus or minus one. This can sometimes cause the eigenvalues also to be clustered around ± 1. If $\mathbf{A}$ is positive (negative) definite then the diagonal terms of $\mathbf{MAM}^T$ will be all positive (negative). More sophisticated preconditioners can be obtained by finding $\mathbf{M}$ through "incomplete" $\mathbf{LU}$ or Cholesky factorizations.

We now compare the performance of the Gauss-Jordan method (GJ) with that of the CG method using the recurrence relation (3. 6) and the bi-conjugate gradient method Bi-CGSTAB [15]. The test set of equations comes from applying the Galerkin approach to a Dirichlet problem in the ellipse given parametrically by

$$x = 1.5\cos\theta \qquad y = \sin\theta, \qquad 0 \leq \theta < 2\pi.$$

We shall call this Example 1 and Table 1 shows the run-times needed to solve the different sizes of linear system corresponding to different discretizations.

N	T_{CG}	T_{stab}	T_{GJ}
64	0.052	0.046	0.224
128	0.138	0.166	1.603
256	0.460	0.662	12.22
512	1.74	2.53	95.91

Table 1: Comparison of linear solvers on Example 1

Here the plain CG method does appreciably better than the Gauss-Jordan approach, and usually outperforms the more complicated Bi-CGSTAB algorithm. Its remarkable performance on the larger problems is due to the fact that it converges to the required accuracy in only four iterations for every value of N. Even if we make the CG method converge to higher accuracy it only requires eight iterations (in the case when $N = 64$).

The good performance of the CG method suggests that the coefficient matrix (2. 8) of the Galerkin linear system for Example 1 has repeated or "bunched" eigenvalues. We can easily investigate this, using the Jacobi method, since the symmetry of $\mathbf{A}$ means that all the eigenvalues are real; and in the case $N = 64$, for instance, we find that there are quite a few instances of "pairing-up" where eigenvalues agree to several significant figures. However, in order to give a more precise measure of grouping, let us suppose that the eigenvalues λ_i are numbered so that $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$. We shall say that λ_i is "$\eta-$*distinct*" if

$$|\lambda_i - \lambda_{i-1}| > \eta|\lambda_i| \quad \text{and} \quad |\lambda_{i+1} - \lambda_i| > \eta|\lambda_i|$$

For the Galerkin matrix of Example 1 with $N = 64$ and with $0.005 \leq \eta \leq 0.01$, the above definition implies that about 58% of the eigenvalues are distinct. This does not go very far towards explaining the exceptionally good behaviour of the CG method. Therefore we can also consider another way of measuring the grouping of eigenvalues and we shall say that λ_i is "$\eta-$*separated*" if

$$|\lambda_i - \lambda_{i-1}| > \eta|\lambda_{max}| \quad \text{and} \quad |\lambda_{i+1} - \lambda_i| > \eta|\lambda_{max}|$$

where λ_{max} is the largest absolute eigenvalue. With η between 0.01 and 0.005, this definition means that the separated eigenvalues of the Galerkin matrix for Example 1 (with $N = 64$) account for

between 22% and 28% of the total. This is somewhat more consistent with the number of CG iterations taken – although it still does not fully account for the speed of convergence.

It is worth noting that diagonal pre-conditioning based on (3. 8) does not improve the performance of the CG method in this case; and analysis of the pre-conditioned matrix shows a similar distribution of eigenvalues.

If this sort of behaviour is typical of other problems then it appears that the CG approach will be very efficient. To test this conjecture we now quote results for three other boundary element problems. For each of the following examples we generate and solve the Galerkin linear system for different values of N and observe the number of CG iterations required, both with and without the use of diagonal pre-conditioning. Unless otherwise stated, the results given below were obtained using the convergence criterion

$$||\mathbf{Ax} - \mathbf{b}|| \leq \epsilon ||\mathbf{b}|| \tag{3. 9}$$

with $\epsilon = 10^{-5}$. (As an important practical point we mention that – in order to obtain a meaningful comparison of a pre-conditioned solution with the original one – the convergence test for the pre-conditioned system must be based on achieving $||\mathbf{MAM}^T\mathbf{y} - \mathbf{Mb}|| \leq \epsilon ||\mathbf{Mb}||$.)

Example 2 involves the problem

$$\nabla^2 u = 0$$

subject to the boundary conditions

$$\begin{aligned} u(0,y) &= 0 \qquad 0 \leq y \leq 1 \\ u(x,0) &= x \qquad 0 \leq x \leq 1 \\ u(1,y) &= 1 \qquad 0 \leq y \leq 1 \\ u(x,1) &= x \qquad 0 \leq x \leq 1 \end{aligned}$$

Results for Example 2 are given in Table 2.

N	no preconditioner	preconditioner
64	17	14
128	26	17
256	32	24
512	92	94

Table 2: Numbers of CG iterations for Example 2.

The results here are rather less dramatic than for Example 1 but even so the numbers of iterations lie between $N/4$ and $N/8$. The use of diagonal preconditioning yields improvements of between 10% and 30%. CG run-times are typically one-third (or better) of the times required by Gaussian elimination (GE); and even if we increase the accuracy of the solutions by setting $\epsilon = 10^{-8}$ in (3. 9) the conjugate gradient solutions for the $N = 256$ and $N = 512$ cases are more than twice as fast as GE.

Example 3 involves the solution of

$$\nabla^2 u = 0$$

subject to the boundary conditions

$$\begin{aligned} q(0,y) &= 0 \qquad 0 \leq y \leq 1 \\ u(x,0) &= 0 \qquad 0 \leq x \leq 1 \\ q(1,y) &= 0 \qquad 0 \leq y \leq 1 \\ u(x,1) &= 1 \qquad 0 \leq x \leq 1 \end{aligned}$$

N	no preconditioner	preconditioner
64	50	37
128	85	68
256	153	136

Table 3: Numbers of CG iterations for Example 3.

Results for Example 3 are summarised in Table 3.

Once again the eigensystem of the Galerkin matrix **A** seems reasonably suitable for the CG approach, since we obtain convergence in appreciably less than N iterations, especially with the benefit of pre-conditioning. (These remarks still hold when the convergence criterion is (3. 9) with $\epsilon = 10^{-8}$.) However, the speed advantage of the CG approach is not now particularly striking. Its run-times are, at best, about 75% of those for GE; and sometimes the iterative solution takes a little longer.

In Example 4 we solve

$$\nabla^2 u = 0$$

subject to the boundary conditions

$$\begin{aligned} u(0,y) &= 300 \qquad 0 \le y \le 6 \\ q(x,0) &= 0 \qquad 0 \le x \le 6 \\ u(6,y) &= 0 \qquad 0 \le y \le 6 \\ q(x,6) &= 0 \qquad 0 \le x \le 6 \end{aligned}$$

Results for this problem are shown in Table 4

N	no preconditioner	preconditioner
64	35	42
128	67	79
256	148	161
512	235	375

Table 4: Numbers of CG iterations on Example 4.

It is slightly surprising to note that the diagonal pre-conditioning strategy always leads to a deterioration in performance on Example 4. Nevertheless, Table 4 shows that, as for previous examples, the number of CG iterations needed is still appreciably less than N. This means, for the larger problems, that the (un-preconditioned) CG method takes only about 60% of the time needed by GE. This advantage is lost, however, if the convergence tolerance is set to $\epsilon = 10^{-8}$.

From the above results it is now clear that the exceptional behaviour displayed in Example 1 cannot be taken for granted. Nevertheless, the Galerkin matrices in Examples 2 - 4 do seem to suit the CG method to some extent. In order to relate the results in Tables 2 - 4 to the eigenvalue distributions of these Galerkin matrices, Table 5 shows percentages of "distinct" eigenvalues for the $N = 64$ cases of Examples 2 - 4. Similarly, Table 6 shows percentages of "separated" eigenvalues.

Both the tables show that pairing and grouping of eigenvalues is quite common for the Galerkin matrices arising in the example problems. However, it has to be acknowledged that there is not particularly good agreement, in detail, between numbers of distinct or separated eigenvalues and the numbers of iterations recorded in Tables 2 - 4. The $\eta = 0.005$ column in Table 6 is quite consistent with unpreconditioned CG performance on Examples 2 and 4, where the numbers of iterations are about $N/4$ and $N/2$ respectively. For Example 3, however, it is the number of distinct eigenvalues

	$\eta = 0.01$	$\eta = 0.005$	$\eta = 0.0025$
Example 2 un-preconditioned	67%	69%	71%
Example 2 preconditioned	71%	71%	73%
Example 3 un-preconditioned	65%	76%	76%
Example 3 preconditioned	66%	74%	82%
Example 4 un-preconditioned	71%	79%	81%
Example 4 preconditioned	66%	74%	82%

Table 5: Percentages of distinct eigenvalues of Galerkin matrices for Examples 2 - 4.

	$\eta = 0.01$	$\eta = 0.005$	$\eta = 0.0025$
Example 2 un-preconditioned	19%	25%	34%
Example 2 preconditioned	18%	25%	40%
Example 3 un-preconditioned	20%	26%	29%
Example 3 preconditioned	29%	34%	43%
Example 4 un-preconditioned	31%	46%	57%
Example 4 preconditioned	22%	28%	40%

Table 6: Percentages of separated eigenvalues of Galerkin matrices for Examples 2 - 4.

in Table 5 which corresponds much better with observed performance. Table 6 is also misleading about the effect of preconditioning since it suggests that faster convergence will occur on Example 4 but not on Example 3. This is precisely the reverse of the observed behaviour!

The above remarks show that there is scope in this area for further research into the behaviour of different implementations of the CG approach. One obvious avenue to explore, in view of the fact the Galerkin matrices we have considered have turned out to be indefinite, would be the use of the more stable three-term recurrence relation (3. 7). A more substantial line of research would be to try more sophisticated pre-conditioning techniques with a view to better exploitation of the favourable eigenvalue structure that seems to arise naturally in our problems. However, the widely used *incomplete factorization* approaches are of most benefit for problems where $\mathbf{A}$ is sparse, and only relatively few elements of the factor matrices need to be calculated. This is not likely to be the case for the Galerkin matrix (2. 8). Improvements on diagonal preconditioning could, however, be based on *quasi-Newton* estimates of $\mathbf{A}^{-1}$. The quasi-Newton approach will now be outlined.

3.2 Quasi-Newton methods for Galerkin systems

Suppose that the diagonal matrix $\mathbf{J}^{(0)}$ is an estimate of $\mathbf{A}^{-1}$. (We might use $\mathbf{J}^{(0)} = diag(a_{ii}^{-1})$ or $\mathbf{J}^{(0)} = \mathbf{I}$.) Then, for any vector $\mathbf{p}$ – such that $(\mathbf{p} - \mathbf{J}^{(0)}\mathbf{A}\mathbf{p})^T\mathbf{A}\mathbf{p} \neq 0$ – the matrix

$$\mathbf{J}^{(1)} = \mathbf{J}^{(0)} + \frac{\mathbf{v}^{(0)}\mathbf{v}^{(0)T}}{\mathbf{v}^{(0)T}\mathbf{A}\mathbf{p}}, \quad \text{where } \mathbf{v}^{(0)} = \mathbf{p} - \mathbf{J}^{(0)}\mathbf{A}\mathbf{p}, \tag{3. 10}$$

is an estimate of $\mathbf{A}^{-1}$ in the sense that it satisfies the quasi-Newton condition

$$\mathbf{J}^{(1)}\mathbf{A}\mathbf{p} = \mathbf{p}. \tag{3. 11}$$

The update (3. 10) is called the *symmetric rank-one formula* and it is well known that, if used N times for distinct vectors $\mathbf{p}$, it will yield $\mathbf{J}^{(N)} = \mathbf{A}^{-1}$. A reasonably cheap preconditioner, therefore, might be obtained by taking

$$\mathbf{M} = \mathbf{J}^{(k)}$$

for some small k, say $k = 1, 2$, and then applying a CG method to the system

$$\mathbf{M}\mathbf{A}^T\mathbf{A}\mathbf{M}\mathbf{y} = \mathbf{M}\mathbf{b}.$$

The use of quasi-Newton updates like (3. 10) has been proposed by Broyden [16] as a way of approximating the Jacobian matrix within a Newton-like method for solving nonlinear equations. (The form (3. 10) is specialised for a symmetric Jacobian; but there are similar updates for the non-symmetric case.) Such quasi-Newton methods can, of course, also be used for solving linear systems – but they are usually reckoned to be uncompetitive with methods like conjugate gradients which do not incur the computational overheads which follow from using a second matrix $\mathbf{J}$ as well as $\mathbf{A}$. In waht follows, however, we report some experimental results obtained with an algorithm which does use (3. 10) to solve Galerkin systems.

The Quasi-Newton (QN) methods we propose have certain features in common with the CG approach described in the preceding sub-section. In particular, each iteration uses a search direction, $\mathbf{p}^{(k)}$ and obtains a new solution estimate of the form

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}$$

where α may be chosen using the formula

$$\alpha^{(k)} = -\frac{\mathbf{p}^{(k)T}\mathbf{r}^{(k)}}{\mathbf{p}^{(k)T}\mathbf{A}\mathbf{p}^{(k)}} \quad \text{where } \mathbf{r}^{(k)} = \mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}.$$

The initial search direction is usually $\mathbf{p}^{(0)} = -\mathbf{r}^{(0)}$; and subsequently we use

$$\mathbf{p}^{(k)} = -\mathbf{J}^{(k)}\mathbf{r}^{(k)}$$

where $\mathbf{J}^{(k)}$ is obtained using the update (3. 10). Specifically, at the end of iteration k we let

$$\delta^{(k-1)} = \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} = \alpha\mathbf{p}^{(k-1)} \quad \text{and} \quad \gamma^{(k-1)} = \alpha\mathbf{A}\mathbf{p}^{(k-1)}$$

and then (3. 10) gives

$$\mathbf{J}^{(k)} = \mathbf{J}^{(k-1)} + \frac{\mathbf{v}^{(k-1)}\mathbf{v}^{(k-1)T}}{\mathbf{v}^{(k-1)T}\gamma^{(k-1)}} \quad \text{where } \mathbf{v}^{(k-1)} = \delta^{(k-1)} - \mathbf{J}^{(k-1)}\gamma^{(k-1)}$$

The important point to note is that, if we choose $\mathbf{J}^{(0)} = \mathbf{I}$, the calculation of $\mathbf{p}^{(k)}$ can be done without forming $\mathbf{J}^{(k)}$ explicitly and incurring the cost of a matrix-vector product. Providing we have stored the vectors $\mathbf{v}^{(0)}, .., \mathbf{v}^{(k-1)}$ we can write

$$\mathbf{p}^{(k)} = -\mathbf{r}^{(k)} - \sum_{j=0}^{k-1} \frac{\mathbf{v}^{(j)T}\mathbf{r}^{(k)}}{\mathbf{v}^{(j)T}\gamma^{(j)}}\mathbf{v}^{(j)}$$

which can be done using about kN multiplications. When $k << N$ this is considerably fewer than the N^2 which would be needed to obtain $\mathbf{J}^{(k)}\mathbf{r}^{(k)}$ directly. A similar calculation scheme can be used to evaluate the product $\mathbf{J}^{(k-1)}\gamma^{(k-1)}$ in (3.2).

The QN approach we have just described can be viewed as an iterative process for minimizing the quadratic function $\mathbf{r}^T\mathbf{A}^{-1}\mathbf{r}$; and in fact the CG method can also be viewed in the same way. Hence QN should converge to the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ in the same number of iterations as the CG method. In practice, the CG method is quite susceptible to the effects of rounding error in finite precision arithmetic; but the QN approach is more robust in this respect. Hence we can reasonably expect the QN method to converge in appreciably less than N iterations on the Galerkin problems; and so the fact that the search direction calculation (3.2) is more expensive than that for the CG approach may not be too serious.

N	no preconditioner	preconditioner
64	22	15
128	31	19
256	39	24
512	99	68

Table 7: Numbers of QN iterations for Example 2.

The tables and discussions below deal with the performance of the QN method on Examples 2 -4 from the previous subsection.

For Example 2 the QN approach does slightly more iterations than the CG method *except* when $N = 512$ and preconditioning is used. In this case QN uses two-thirds the number of steps and this is enough to make it the fastest algorithm, by a fairly small margin. It is also worth noting that, generally speaking, pre-conditioning has a relatively more significant effect on the QN method than on conjugate gradients.

N	no preconditioner	preconditioner
64	43	34
128	71	55
256	112	95

Table 8: Numbers of QN iterations for Example 3.

On Example 3 the QN approach consistently needs fewer iterations than the CG method. However, it is the CG method that uses less arithmetic overall; and the QN approach only outperforms Gaussian elimination as regards run-time in the case $N = 256$ when pre-conditioning is used.

N	no preconditioner	preconditioner
64	35	35
128	64	59
256	111	100
512	178	178

Table 9: Numbers of QN iterations on Example 4.

On Example 4 the advantage of the QN method over the CG approach (in terms of iteration count) becomes more significant as N increases. The additional overheads of QN, however, mean that the unpreconditioned CG solution is still the fastest, for the level of accuracy given by (3. 9). In fact, for $N = 512$, CG takes about 60% of the time needed by Gaussian elimination whereas QN uses about 80%. However, when the required accuracy is increased, so that $\epsilon = 10^{-8}$ in (3. 9), the greater robustness of the QN scheme can be observed: QN still converges in slightly less time than GE, but CG needs 20% longer.

It is worth mentioning in passing that we have also tried a slightly different implementation of the QN approach, in which the step length α is chosen in a different way. In this version the QN approach can be regarded as seeking the minimum of $\mathbf{r}^T\mathbf{r}$, rather than $\mathbf{r}^T\mathbf{A}^{-1}\mathbf{r}$. Some differences in performance were observed, but overall the effectiveness of the method is about the same as the one we have described above. An alternative strategy which had *not* yet been tried would be a composite approach which performs a moderate number of QN iterations (say about $N/3$) and then switches to CG when the matrix-vector calculations become expensive.

Several interesting questions about linear solvers for Galerkin systems have been raised in this section and more work needs to be done to resolve them. For the remainder of this paper, however, we shall confine ourselves to the use of the CG method in the context of investigating some wider benefits of running Galerkin boundary element method problems in a parallel environment.

4 Implementation on a multiprocessor parallel computer

The Galerkin boundary element method may be computationally expensive for the solution of large problems. The generation and solution of the large matrix systems resulting from the discretization of the boundary often requires significant storage and computational power. In such problems, where a large number of similar operations is performed [4], a parallel machine is a particularly attractive option since it can be used to exploit the inherent parallelism of the Galerkin boundary element algorithm. Distributed memory machines, although requiring extra programming, can provide truly scalable performance, at considerably lower cost than that of current vector supercomputers.

In this Section we deal with the parallel implementation on a multiprocessor parallel computer of the Galerkin boundary element method described in Section 2. The code involves three phases: the generation of the influence matrices and the assembly of the system matrix and the right-hand-side; the solution of the system of equations; the recovery of internal potentials. In the implementation of the method, phases one and two are computationally and numerically intensive and the storage and computational costs grow rapidly as the problem size increases. Natarajan and Krishnaswamy [5] state that, if N is the number of nodal unknowns, then the storage requirement for the coefficient matrix is $O(N^2)$ while the computational cost for phases one and two are $O(N^2)$ and $O(N^3)$ respectively.

4.1 Geometric domain decomposition

The domain decomposition method is commonly used for the solution of large boundary element problems by breaking them into smaller problems, and using multiprocessor parallel computers. Early parallel implementations of the boundary element method were developed on transputer networks by Davies [6] where the Gauss quadrature was parallelised and by Daoudi and Lobry [7] who distributed the elements in a cyclic fashion. Recent parallel implementations use parallel code development packages. Semeraro and Gray [8] use a block decomposition technique on the system matrix for the symmetric-Galerkin method with PVM. Kreienmeyer and Stein [9] use a collocation data decomposition technique on a Parsytec MultiCluster2 with 32 processors. They demonstrate that, provided the speed of inter-processor communication is relatively fast compared with PVM, theoretical improvements in performance are attainable. Geometric or data parallelism is the most natural subdivision of the workload for calculations over a region of space. A parallel multi-block data decomposition boundary element method solution by Davies and Mushtaq [10] requires an iterative process to update the coupling data between adjacent multiple blocks.

4.2 Multi-partition method

The multi-partition method is described by Mushtaq and Davies [4] [11]. The original boundary of the problem is subdivided into multiple boundary partitions, see Figure 3, so that each partition is distributed to a network of processors in boundary sections. Such a distribution allows phases one and three to be disjoint i.e. no inter-processor communication is necessary.

Figure 3: Illustration of the multi-partition method.

The system of equations corresponding to the partitioning strategy of Figure 3 is written as

$$\begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \mathbf{H}_3 \\ \mathbf{H}_4 \end{bmatrix} \mathbf{u} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \mathbf{G}_3 \\ \mathbf{G}_4 \end{bmatrix} \mathbf{q}$$

where the subscripts represent the partition number. On application of the boundary conditions, the system of equations may be written as

$$\begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \\ \mathbf{A}_4 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \end{bmatrix}$$

where $\mathbf{A}$ is the symmetric system matrix, $\mathbf{x}$ is the unknown vector, and $\mathbf{b}$ is the known vector as defined in Section 2. Each matrix-vector combination $\mathbf{A}$ and $\mathbf{b}$ corresponding to a boundary partition can be generated independently of the others. This is illustrated in Figure 4.

Figure 4: Illustration of the data mapping.

Phases one and three involve the parallel data distribution of the matrices and this is performed such that the coefficients of the matrices are evenly distributed to individual processors in order to balance the computational load and achieve a highly efficient parallelism. The domain decomposition method is used and the domain of the problem is divided into sub-domains, each processor being responsible for evaluating its subset of the boundary double integrals of the sub-domain. The double integrals correspond to rows and columns of the matrices. If N is the problem size and p is the number of processors, then each processor is responsible for N/p rows of the system matrix and N/p elements of the right-hand-side vector which gives a well-balanced workload. Each processor is also responsible for recovering the interior potentials of l/p interior points, where l is the number of internal points at which the internal solution is sought. The order of the matrices for each sub-domain is much smaller than that of the whole domain, and so the storage limitation can be overcome with the domain decomposition method. Also, the data distributed over the processors is smaller than that of a single processor computer and so the computational efficiency and speed is much higher.

The parallelization of phase two is performed by data partitioning which leads to an efficient parallelism. The initial experiments to be described here involve parallel versions of only three of the linear solvers mentioned in the previous section. These are the conjugate gradient method, the biconjugate gradient (stabilised) method and the Gauss-Jordan method. It should be noted that, in contrast to phases one and three, phase two requires inter-processor communication. This inter-processor communication is involved in passing data for such phase two operations as scalar products, matrix-vector products, gathering data from processors and broadcasting data to processors.

The parallel distributed computer used in this work is an $nCUBE\ 2$ with 64 processors in a hypercube configuration each having 4Mbytes local memory on each processor.

In the performance results which follow in Section 4.3 we define parallel Speed-up S_p, as

$$S_p = \frac{Computation\ time\ on\ one\ processor}{Computation\ time\ on\ p\ processors}$$

4.3 Performance results for parallel Galerkin boundary element method

To illustrate how symmetry is exploited in the multiprocessor environment we consider the Dirichlet problem described as Example 1 in the previous section.

In the Galerkin method, phase one is highly computationally intensive compared with the same phase in the collocation method. However, it is ideally suited for a distributed memory machine because of the absence of inter-processor communication. The same remark also applies to phase three.

The parallel performances in phases one and three are shown in Tables 10 and 11. We see from these tables that, as the problem size for phase one and the number of interior points for phase three is doubled, the computation time for the phases increases by a factor of four independent of the number of processors and that for a given number of processors the time is $O(N^2)$. As the number of processors doubles, however, the computation time for the phases decreases by a factor of two, independent of the problem size and the number of interior points. This means that phases one and three are highly parallel efficient, almost one hundred percent, independent of the problem size, the number of interior points and the number of processors. This is due to the fact that the load is well-balanced, i.e. the tasks involved in these phases are uniformly distributed over the processors. The desired linear speed-up for different problem sizes in phase one and different interior point numbers in phase three is obtained, which indicates the efficiency of this algorithm in a multiprocessor environment.

n	64	128	256	512
p	Time (s)			
1	8.72	35.08	140.69	563.50
2	4.36	17.54	70.35	281.75
4	2.18	8.77	35.17	140.88
8	1.09	4.39	17.59	70.46
16	0.55	2.19	8.79	35.22
32	0.27	1.10	4.40	17.61
64	0.14	0.55	2.20	8.81

Table 10: Time (seconds) taken for Phase one.

l	64	128	256	512
p	Time (s)			
1	1.23	4.92	19.70	78.78
2	0.62	2.46	9.85	39.39
4	0.31	1.23	4.92	19.70
8	0.15	0.62	2.46	9.85
16	0.07	0.31	1.23	4.93
32	0.04	0.16	0.62	2.47
64	0.02	0.08	0.31	1.23

Table 11: Time (seconds) taken for Phase three.

We now turn to the parallelization of phase two. We have already seen that the symmetry of the equations enables us to use the conjugate gradient method and, moreover, that for Example 1 the CG method converges in only four iterations without preconditioning. We now consider the performance of this method, along with other linear solvers, in a parallel environment.

The times taken in phase two, by the parallel solvers, conjugate gradient without preconditioning, bi-conjugate gradient (stabilised) and Gauss-Jordan for different problem size are compared in Tables 12 - 15. The Gauss-Jordan direct solver was preferred to Gaussian elimination because no back substitution is required, thus eliminating processor idle time.

From Tables 12 - 15 we make the following observations.
(i) The conjugate gradient algorithm significantly outperforms the Gauss-Jordan algorithm in all

cases, even though the Gauss-Jordan has the best speed-up factor.
(ii) Phase two clearly does not exhibit the same parallelism as phases one and three. The speed-up values are not constant; and for each value of N the computation time initially decreases as p increases but then it reaches a minimum and starts to increase. The number of processors at which a minimum occurs increases with the problem size. Table 15, in particular, shows that it has not been reached for 64 processors with $n = 512$. The reason for this increase in computation time is that for a small number of processors the communication costs are small but the solution costs are high. As the number of processors increases so too do the communication costs, whereas the solution cost decreases. We eventually reach an optimum number of processors for which the total cost of phase two is a minimum.

Taken together, Tables 10 - 15 also show that the set-up phase for the Galerkin method is the most expensive for this example. However, the computation cost in this phase, as well as in phase three, reduces linearly with the number of processors. Consequently, since eventually the cost in phase two starts to increase with N, there will be sizes of problem for which the equation solution phase will be the dominant part of the solution cost.

$n = 64$						
p	T_{CG}	T_{stab}	T_{GJ}	S_{CG}	S_{stab}	S_{GJ}
1	0.052	0.046	0.224	1.00	1.00	1.00
2	0.039	0.029	0.134	1.35	1.57	1.68
4	0.035	0.022	0.088	1.51	2.03	2.56
8	0.035	0.023	0.071	1.50	2.02	3.17
16	0.039	0.028	0.067	1.33	1.64	3.36
32	0.048	0.039	0.074	1.09	1.18	3.04
64	0.066	0.061	0.091	0.79	0.75	2.47

Table 12: Time (s) and Speed-up for Phase two.

$n = 128$						
p	T_{CG}	T_{stab}	T_{GJ}	S_{CG}	S_{stab}	S_{GJ}
1	0.138	0.166	1.603	1.00	1.00	1.00
2	0.075	0.091	0.841	1.82	1.82	1.91
4	0.053	0.054	0.461	2.62	3.05	3.48
8	0.043	0.040	0.282	3.17	4.11	5.68
16	0.042	0.039	0.204	3.30	4.28	7.85
32	0.047	0.047	0.181	2.95	3.55	8.88
64	0.059	0.067	0.194	2.32	2.47	8.25

Table 13: Time (s) and Speed-up for Phase two.

The final set of Tables 16 - 18 show the times for all three individual phases compared with the total time (seconds) taken for the complete implementation. This total time includes the three phases *together with data broadcast and gather time* which reflects the need to distribute data to the p processors at the beginning of a solution and to collect results at the end. As the number of processors increases, the *total* time taken for the implementation decreases until an optimum is reached for the number of processors for a particular problem size. This eventual increase in the implementation time is due to the fact that as the number of processors increases (for a particular problem size) so the bradcast and gather costs increase.

$n = 256$						
p	T_{CG}	T_{stab}	T_{GJ}	S_{CG}	S_{stab}	S_{GJ}
1	0.460	0.662	12.22	1.00	1.00	1.00
2	0.250	0.355	6.26	1.84	1.86	1.95
4	0.137	0.203	3.27	3.36	3.25	3.74
8	0.088	0.121	1.80	5.20	5.47	6.79
16	0.069	0.094	1.11	6.64	7.03	10.99
32	0.064	0.093	0.81	7.18	7.11	15.05
64	0.071	0.105	0.73	5.49	6.28	16.84

Table 14: Time (s) and Speed-up for Phase two.

$n = 512$						
p	T_{CG}	T_{stab}	T_{GJ}	S_{CG}	S_{stab}	S_{GJ}
1	1.74	2.53	95.91	1.00	1.00	1.00
2	0.90	1.29	48.43	1.94	1.95	1.98
4	0.47	0.68	24.62	3.67	3.73	3.90
8	0.27	0.38	12.85	6.51	6.73	7.47
16	0.17	0.24	7.09	9.98	10.66	13.53
32	0.12	0.16	4.33	14.95	15.38	22.17
64	0.11	0.15	3.11	15.90	17.24	30.87

Table 15: Time (s) and Speed-up for Phase two.

n	64		128		256		512	
p	Total Time(s)	Phase1 Time(s)	Total Time(s)	Phase1 Time(s)	Total Time(s)	Phase1 Time(s)	Total Time(s)	Phase1 Time(s)
1	10.20	8.72	40.55	35.08	161.80	140.69	646.33	563.50
2	5.46	4.36	20.73	17.54	81.56	70.35	342.21	281.75
4	3.46	2.18	11.20	8.77	41.82	35.17	163.54	140.88
8	3.22	1.09	7.19	4.39	22.70	17.59	83.98	70.46
16	4.62	0.55	6.70	2.19	14.67	8.79	45.71	35.22
32	8.34	0.27	9.49	1.20	13.69	4.40	29.63	17.61
64	16.26	0.14	16.94	0.55	19.28	2.20	27.72	8.81

Table 16: Total time (s) taken for the implementation and Phase one.

5 Conclusion

The Galerkin method is seen to be highly parallel-efficient, giving near-perfect speed-up in the system set-up phase and in the recovery of internal potentials. The problems considered in this paper are relatively small and so the equation solution time remains small compared with the set-up time. However, for larger problems, the set-up time reduces linearly with the number of processors whereas the equation-solving time does not. The symmetry of the equations developed from the Galerkin method allows us to use iterative schemes (conjugate gradients and quasi-Newton) which has been shown to have the potential to converge very rapidly for problems of this type, thus keeping the solution costs relatively low. However, there is still scope for more investigation of the properties of the coefficient matrix (2. 8) in the Galerkin system with a view to taking even more advantage of its eigenvalue structure.

n	64		128		256		512	
p	Total Time(s)	Phase2 Time(s)	Total Time(s)	Phase2 Time(s)	Total Time(s)	Phase2 Time(s)	Total Time(s)	Phase2 Time(s)
1	10.20	0.046	40.55	0.166	161.80	0.662	646.33	2.53
2	5.46	0.029	20.73	0.091	81.56	0.355	342.21	1.29
4	3.46	0.022	11.20	0.054	41.82	0.203	163.54	0.68
8	3.22	0.023	7.19	0.040	22.70	0.121	83.98	0.38
16	4.62	0.028	6.70	0.039	14.67	0.094	45.71	0.24
32	8.34	0.039	9.49	0.047	13.69	0.093	29.63	0.16
64	16.26	0.061	16.94	0.067	19.28	0.105	27.72	0.15

Table 17: Total time (s) taken for the implementation and Phase two.

l	64		128		256		512	
p	Total Time(s)	Phase3 Time(s)	Total Time(s)	Phase3 Time(s)	Total Time(s)	Phase3 Time(s)	Total Time(s)	Phase3 Time(s)
1	10.20	1.23	40.55	4.92	161.80	19.70	646.33	78.78
2	5.46	0.62	20.73	2.46	81.56	9.85	342.21	39.39
4	3.46	0.31	11.20	1.23	41.82	4.92	163.54	19.70
8	3.22	0.15	7.19	0.62	22.70	2.46	83.98	9.85
16	4.62	0.07	6.70	0.31	14.67	1.23	45.71	4.93
32	8.34	0.04	9.49	0.16	13.69	0.62	29.63	2.47
64	16.26	0.02	16.94	0.08	19.28	0.31	27.72	1.23

Table 18: Total time (s) taken for the Implementation and Phase Three.

References

[1] Brebbia C.A. and Dominquez J. *Boundary Elements, an Introductory Course*, Computational Mechanics Publications, 1992.

[2] Gray L.J. Evaluation of singular and hypersingular Galerkin integrals: direct limits and symbolic computation; *Singular Integrals in boundary element methods*, eds Sladek V. and Sladek J, Computational Mechanics Publications, 33-84, 1998.

[3] Sladek V. and Sladek J. Introductory notes on singular integrals; *Singular Integrals in boundary element methods*, eds Sladek V. and Sladek J, Computational Mechanics Publications, 1-31, 1998.

[4] Mushtaq J. and Davies A.J. Parallel boundary element implementation of the aerofoil problem on a Local Area Network multicomputer system using PVM and HPF. *Proceedings of PART '97: The 4th Australasian Conference on Parallel and Real-Time Systems*, eds Sharda N. and Tam A, Springer-Verlag, 126-133,1998.

[5] Natarajan R. and Krishnaswamy D. A case study in parallel scientific computing: the boundary element method on a distributed-memory multicomputer. *Engrg. Anal. with Bound. Elem.*, **18**, 183-193, 1996.

[6] Davies A.J. The boundary element method on a transputer network. *Boundary Elements XIII*, ed. Brebbia C.A. Computational Mechanics Publications, 985-994, 1991.

[7] Daoudi E.M. and Lobry J. Implementation of a boundary element method on distributed memory computers. *Parallel Computing*, **18**, 1317-1324, 1992.

[8] Semeraro B.D. and Gray L.J. PVM implementation of the symmetric-Galerkin method. *Engrg. Anal. with Bound. Elem.*, **19**, 67-72, 1997.

[9] Kreienmeyer M. and Stein E. Parallel implementation of the boundary element method for linear elastic problems on a MIMD Parallel Computer. *Computation Mechanics*, **15**, 342-349, 1995.

[10] Davies A.J. and Mushtaq J. The domain decomposition boundary element method on a network of transputers. *Boundary Element Technology XI*, eds Ertekin R.C. Brebbia C.A. Tanaka M. and Shaw R. Computational Mechanics Publications, 397-406, 1996.

[11] Mushtaq J. and Davies A.J. Parallel implementation of the boundary element method using PVM and HPF on a collection of networked workstations. *High Performance Computing 97*, 181-188, 1997.

[12] Hestenes M.R. and E. Steifel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bureau of Standards* 49, 409-436, 1952.

[13] Romate J.E. On the use of conjugate gradient type methods for boundary integral equations. *Computational Mechanics* 12, 214-232, 1993.

[14] Nazareth L. A conjugate gradient algorithm without line searches. *JOTA* 23, 373-387, 1977.

[15] van der Vorst H.A. BiCGSTAB: a fast and smoothly convergent version of Bi-CG for the solution of nbonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* 13, 631-644, 1992.

[16] Broyden C.G. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation* 19, 577-593, 1965