INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **28**: 1421–1440 (1998)

PARALLEL CONJUGATE GRADIENT PERFORMANCE FOR LEAST-SQUARES FINITE ELEMENTS AND TRANSPORT PROBLEMS

G.F. CAREY*, Y. SHEN AND R.T. MCLAY

Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin, TX, USA

SUMMARY

In this study we consider parallel conjugate gradient solution of sparse systems arising from the least-squares mixed finite element method. Of particular interest are transport problems involving convection. The least-squares approach leads to a symmetric positive system and the conjugate gradient scheme is directly applicable. The scheme is applied to both the convection–diffusion equation and to the stationary Navier–Stokes equations. Here we demonstrate parallel solution and performance studies for a representative MIMD parallel computer with hypercube architecture. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: parallel; conjugate gradient; least squares; FEM

1. INTRODUCTION

Most numerical methods for solving transport problems involving convection lead to non-symmetric algebraic systems. Frequently these problems involve high-resolution grids, and iterative methods such as the biconjugate gradient scheme are utilized. For large-scale applications these problems are best addressed on distributed parallel computer systems. Least-squares finite element schemes generate symmetric positive systems and the conjugate gradient method can be applied directly. The performance of this iterative solution strategy for least-squares finite element approximation of flow problems on distributed parallel computers is clearly of relevance to computational fluid dynamics (CFD). Recently, considerable attention has been focused on the performance of iterative methods for solving finite difference and finite element systems on distributed memory multiprocessors. The solution algorithms include gradient-type methods such as the conjugate gradient method as well as the multigrid method. Parallel schemes have been implemented, based either on domain decomposition [1,2] or matrix decomposition [3,4]. In the latter case, when unstructured grids are used, the sparse structure may be quite irregular and this leads to irregular, long distance communication requirements that severely degrade the algorithms. On the other hand, when domain decomposition can be applied as in the applications of interest here, it is optimal for uniformly balanced structured grids and appropriate node or element numbering. Likewise, parallel schemes based on domain decomposition can be extended to unstructured grids if used with suitable partitioning strategies for parallel load balancing [5]. Most studies for parallel CG are based on standard

* Correspondence to: Department of Aerospace Engineering and Engineering Mathematics, WRW-305, University of Texas at Austin, TX 78712, USA.

CCC 0271–2091/98/201421–20$17.50
© 1998 John Wiley & Sons, Ltd.
Received 23 July 1996
Revised 10 December 1997

finite difference and finite element methods and relatively little is known for the least-squares mixed finite element method considered here.

There are several studies in the literature where schemes based on collocation, least-squares and other weighted-residual ideas have been discussed [6–8]. Generally, the Galerkin approach is preferable because of the reduced smoothness assumptions on the approximation and the ability to exploit natural boundary conditions. If the problem is recast as a lower order system then mixed methods may be used. This mixed approach for the Galerkin formulation is popular among certain researchers because the flux or stress now enters explicitly in the formulation [9–11]. It also leads to a preferable formulation for the least-squares method for similar reasons and the smoothness restrictions are relaxed accordingly [12–14]. Moreover, in both theoretical and numerical studies we have demonstrated [15,16] that the least-squares finite element formulation circumvents the consistency constraints (LBB or inf–sup restrictions) of the mixed Galerkin finite element method (FEM) [10,17]. Finally, the least-squares approach automatically generates a symmetric positive system.

Hence there are several attractive features to the least-squares approach. However, there are a number of issues and open questions that still need to be addressed. These include, e.g. the question of weight selection in the least-squares residual statement, mesh sensitivity of the method, h and p refinement issues and the numerical performance of the method. In a previous paper [18] we developed a class of multilevel preconditioning strategies based on the approach of Chan and Vassilevski [19] and showed for some simple scalar elliptic problems that the preconditioned system converged essentially independent of the mesh size. In the present work we first investigate the conditioning of the basic least-squares system and then consider parallel solution with simpler preconditioners than in Reference [19]. Parallel performance studies for both convection–diffusion and Navier–Stokes test problems are included.

The outline of the treatment is as follows: In Section 1 we briefly present the least-squares formulation in a general operator form. Next, the basic conjugate gradient algorithm and simple preconditioners of interest here are summarized together with the convergence properties. A Cartesian domain decomposition for topological unions of rectangles, which is used as one of the basic data structures in our parallel software package for preconditioning conjugate gradient (PCG) [20,21], is described in Section 2. Results are presented in Section 3 for a representative linear convection–diffusion problem and for a viscous flow problem. These include parallel performance studies for both shared memory and distributed memory parallel architectures.

2. LEAST-SQUARES FINITE ELEMENT FORMULATION AND CONJUGATE GRADIENT SOLUTION

Most standard schemes (such as the Galerkin finite element method and finite difference method) lead to non-symmetric algebraic systems on discretization of convective problems. This implies that non-symmetric iterative solution schemes such as biconjugate gradient (BCG) rather than conjugate gradient must be applied. BCG is known to 'breakdown' or stagnate and both failures have been observed in our previous studies for Navier–Stokes using Galerkin FEM. On the other hand, one can prove convergence of CG for symmetric positive systems.

For convenience, let us consider a linear elliptic boundary value problem cast as the first-order system

$$\boldsymbol{L}\boldsymbol{z} = \boldsymbol{f} \quad \text{in } \Omega, \tag{1}$$

© 1998 John Wiley & Sons, Ltd.

with appropriate data $\boldsymbol{B}\boldsymbol{z} = \boldsymbol{g}$ on the boundary $\partial\Omega$. Here $\boldsymbol{L}$ is a first-order differential operator for the system and $\boldsymbol{B}$ is a corresponding operator on the boundary. For example, a second-order convection–diffusion problem

$$-\nabla \cdot (a\nabla u) + \boldsymbol{c} \cdot \nabla u = d \tag{2}$$

can be recast into a first-order system as

$$\begin{bmatrix} \boldsymbol{c} \cdot \nabla & \nabla \cdot \\ a\nabla & 1 \end{bmatrix} \begin{bmatrix} u \\ \boldsymbol{\sigma} \end{bmatrix} = \begin{bmatrix} d \\ \boldsymbol{0} \end{bmatrix}, \tag{3}$$

which corresponds to Equation (1) with $\boldsymbol{z} = (u, \boldsymbol{\sigma})$ where u is a scalar and $\boldsymbol{\sigma}$ is a vector. This form is typical of most mixed problems of interest where $\boldsymbol{\sigma}$ denotes the flux and is related to u through a constitutive equation. However, more general forms are permitted. Here we assume, of course, that variables have first been dimensionally scaled.

For admissible trial function $\boldsymbol{z}$ we can introduce the associated residuals for the differential equation and boundary equations. For simplicity of exposition, let us assume that the boundary conditions are satisfied *a priori*. The interior residual is then $\boldsymbol{R} = \boldsymbol{L}\boldsymbol{z} - \boldsymbol{f}$ and the least-squares functional is

$$I = \frac{1}{2} \int_{\Omega} \boldsymbol{R}^T \boldsymbol{R} \, \mathrm{d}\boldsymbol{x}. \tag{4}$$

Applying the stationary condition $\delta I = 0$ implies

$$\int_{\Omega} (\boldsymbol{L}\boldsymbol{y})^T (\boldsymbol{L}\boldsymbol{z}) \, \mathrm{d}\boldsymbol{x} = \int_{\Omega} (\boldsymbol{L}\boldsymbol{y})^T \boldsymbol{f} \, \mathrm{d}\boldsymbol{x}, \tag{5}$$

where $\boldsymbol{y} = \delta\boldsymbol{z}$ is the test function. Substituting for $\boldsymbol{L}$ and expanding $\boldsymbol{z} = (u, \boldsymbol{\sigma})$, $\boldsymbol{y} = (v, \boldsymbol{\tau})$, we get

$$\int_{\Omega} [(\nabla \cdot \boldsymbol{\tau} + \boldsymbol{c} \cdot \nabla v)(\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{c} \cdot \nabla u) + (\boldsymbol{\tau} + a\nabla v) \cdot (\boldsymbol{\sigma} + a\nabla u)] \, \mathrm{d}\boldsymbol{x} = \int_{\Omega} [(\nabla \cdot \boldsymbol{\tau} + \boldsymbol{c} \cdot \nabla v)d] \, \mathrm{d}\boldsymbol{x}, \tag{6}$$

where $v = \delta u$, $\boldsymbol{\tau} = \delta\boldsymbol{\sigma}$. We write Equation (6) in standard notation as: find $(u, \boldsymbol{\sigma})$ satisfying the specified boundary conditions and such that

$$a(u, \boldsymbol{\sigma}; v, \boldsymbol{\tau}) = b(v, \boldsymbol{\tau}), \tag{7}$$

for all admissible $\boldsymbol{v} = (v, \boldsymbol{\tau})$ where $a(\cdot\,;\cdot)$ and $b(\cdot)$ denote the respective functionals in Equation (6). Then Equation (6) or equivalently (7) provides an alternative least-squares variational statement and an approximate formulation can be developed within this mathematical framework. In particular, a mixed least-squares finite element scheme can be constructed instead of a mixed Galerkin scheme.

Accordingly, let us construct a finite element partition of the domain and introduce a corresponding basis to define the approximation trial and test subspaces. Then the approximate formulation becomes: find u_h, $\boldsymbol{\sigma}_h$ satisfying the boundary conditions and such that

$$a(u_h, \boldsymbol{\sigma}_h; v_h, \boldsymbol{\tau}_h) = b(v_h, \boldsymbol{\tau}_h), \tag{8}$$

for all admissible v_h, $\boldsymbol{\tau}_h$.

Introducing finite element expansions $u_h = \Sigma_{j=1}^{J} u_j \phi_j$, $\boldsymbol{\sigma}_h = \Sigma_{m=1}^{M} \boldsymbol{\sigma}_m \boldsymbol{\chi}_m$ and setting $v_h = \phi_i$, $\boldsymbol{\tau}_h = \boldsymbol{\chi}_l$ where ϕ_j, $\boldsymbol{\chi}_m$ denote the finite element basis functions and u_j, $\boldsymbol{\sigma}_m$ are the nodal unknowns, the least-squares algebraic system from Equation (7) has the form

© 1998 John Wiley & Sons, Ltd.

$$\begin{bmatrix} \boldsymbol{A}^{uu} & \boldsymbol{A}^{u\sigma} \\ \boldsymbol{A}^{\sigma u} & \boldsymbol{A}^{\sigma\sigma} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\sigma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{u} \\ \boldsymbol{b}^{\sigma} \end{bmatrix}, \tag{9}$$

where

$$A_{ij}^{uu} = \int_{\Omega} [(\boldsymbol{c} \cdot \nabla\phi_i)(\boldsymbol{c} \cdot \nabla\phi_j) + (a\nabla\phi_i) \cdot (a\nabla\phi_j)]\,\mathrm{d}\boldsymbol{x},$$

$$A_{im}^{u\sigma} = \int_{\Omega} [(\boldsymbol{c} \cdot \nabla\phi_i)(\nabla \cdot \boldsymbol{\chi}_m) + (a\nabla\phi_i) \cdot \boldsymbol{\chi}_m]\,\mathrm{d}\boldsymbol{x},$$

$$A_{lj}^{\sigma u} = \int_{\Omega} [(\nabla \cdot \boldsymbol{\chi}_l)(\boldsymbol{c} \cdot \nabla\phi_j) + \boldsymbol{\chi}_l \cdot (a\nabla\phi_j)]\,\mathrm{d}\boldsymbol{x},$$

$$A_{lm}^{\sigma\sigma} = \int_{\Omega} [(\nabla \cdot \boldsymbol{\chi}_l)(\nabla \cdot \boldsymbol{\chi}_m) + \boldsymbol{\chi}_l \cdot \boldsymbol{\chi}_m]\,\mathrm{d}\boldsymbol{x},$$

$$b_i^u = \int_{\Omega} [(\boldsymbol{c} \cdot \nabla\phi_i)d]\,\mathrm{d}\boldsymbol{x},$$

$$b_l^\sigma = 0,$$

for $i,\ j = 1, 2, \ldots, J$ and $l,\ m = 1, 2, \ldots, M$. Finally, we enforce the specified boundary conditions in Equation (9) and the resulting system can be written compactly as

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{10}$$

where $\boldsymbol{A}$ is the block matrix and $\boldsymbol{x}$ is the vector of nodal solution unknowns from Equation (9).

The least-squares property in Equation (4) implies $\boldsymbol{A}$ is symmetric positive definite, which suggests that a preconditioned conjugate gradient iterative solver for Equation (10) might be effective [22–24]. Accordingly, let us introduce a preconditioner $\boldsymbol{Q}$ so that the preconditioned system is equivalent to $\boldsymbol{Q}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{Q}^{-1}\boldsymbol{b}$. The basic preconditioned conjugate gradient algorithm is then:

For $\boldsymbol{x}^{(0)}$ given, construct

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \lambda_n \boldsymbol{p}^{(n)}, \quad n = 0, 1, \ldots, \tag{11}$$

where the direction $\boldsymbol{p}^{(n)}$ and step size λ_n are given by

$$\boldsymbol{p}^{(n)} = \begin{cases} \boldsymbol{\delta}^{(0)}, & n = 0 \\ \boldsymbol{\delta}^{(n)} + \alpha_n \boldsymbol{p}^{(n-1)}, & n \neq 0 \end{cases},$$

$$\alpha_n = \frac{(\boldsymbol{r}^{(n)}, \boldsymbol{\delta}^{(n)})}{(\boldsymbol{r}^{(n-1)}, \boldsymbol{\delta}^{(n-1)})},$$

$$\lambda_n = \frac{(\boldsymbol{r}^{(n)}, \boldsymbol{\delta}^{(n)})}{(\boldsymbol{p}^{(n)}, \boldsymbol{A}\boldsymbol{p}^{(n)})},$$

with

$$\boldsymbol{\delta}^{(n)} = \boldsymbol{Q}^{-1}\boldsymbol{r}^{(n)},$$

$$\boldsymbol{r}^{(n)} = \begin{cases} \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, & n = 0 \\ \boldsymbol{r}^{(n-1)} + \lambda_{n-1}\boldsymbol{A}\boldsymbol{p}^{(n-1)}, & n \neq 0 \end{cases}.$$

As stopping test we use the relative residual

© 1998 John Wiley & Sons, Ltd.

$$\frac{\|\mathbf{r}^{(n)}\|_0}{\|\mathbf{r}^{(0)}\|_0} \leq \zeta, \tag{12}$$

where ζ is a specified tolerance Some simple preconditioners based directly on the structure of $\mathbf{A}$ can be specified as follows: first assume $\mathbf{A}$ is expressed in the form $\mathbf{A} = \mathbf{D} - \mathbf{C}_{\mathrm{L}} - \mathbf{C}_{\mathrm{U}}$ where $\mathbf{D}$ is a diagonal matrix, $\mathbf{C}_{\mathrm{L}}$ is a strictly lower triangular matrix and $\mathbf{C}_{\mathrm{U}}$ is a strictly upper triangular matrix. Then we can introduce, for example, the following basic preconditioners

$$\mathbf{Q} = \begin{cases} \mathbf{I} & \text{(RF)} \\ \mathbf{D} & \text{(J)} \\ \dfrac{1}{2-\omega}\left(\dfrac{1}{\omega}\mathbf{D} - \mathbf{C}_{\mathrm{L}}\right)\left(\dfrac{1}{\omega}\mathbf{D}\right)^{-1}\left(\dfrac{1}{\omega}\mathbf{D} - \mathbf{C}_{\mathrm{U}}\right) & \text{(SSOR)} \\ p_s(\mathbf{A})^{-1} & \text{(LSP)} \end{cases} \tag{13}$$

Here the RF–CG scheme (Richardson iteration) reduces simply to the basic conjugate gradient method, J–CG is Jacobi preconditioning, SSOR–CG uses a symmetric overrelaxation preconditioner and LSP–CG uses a least-squares polynomial preconditioner where the polynomial p_s is selected such that $f(z) = 1 - p_s(z)z$ has minimum norm over a domain that contains the spectrum of $\mathbf{A}$.

Convergence of the PCG scheme depends on the condition number $\kappa(\mathbf{A})$. In the present work for least-squares systems, $\mathbf{A}$ is symmetric positive and we apply symmetric positive preconditioners $\mathbf{Q}$. Then, the condition number for the preconditioned system is the ratio of the maximum and minimum eigenvalues of $\mathbf{Q}^{-1}\mathbf{A}$, i.e. $\kappa = \lambda_{\max}/\lambda_{\min}$. Now let $\mathbf{x}$ be the exact solution of the linear system. The iterate error $\mathbf{e}^{(n)} = \mathbf{x} - \mathbf{x}^{(n)}$ satisfies the Chebyshev bound [25]

$$\frac{\|e^{(n)}\|_A}{\|e^{(0)}\|_A} \leq 1/\cosh\left[n \log\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)\right], \tag{14}$$

where $\|\cdot\|_A$ denotes the A-norm of the vector. We have the following result for the algebraic residual $\mathbf{r}^{(n)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(n)}$ in terms of the condition number κ for $\mathbf{Q}^{-1}\mathbf{A}$ and $\hat{\kappa}$ for $\mathbf{A}$,

$$\frac{\|\mathbf{r}^{(n)}\|_0}{\|\mathbf{r}^{(0)}\|_0} \leq \sqrt{\hat{\kappa}}\,\frac{\|\mathbf{e}^{(n)}\|_A}{\|\mathbf{e}^{(0)}\|_A}. \tag{15}$$

Now for large κ,

$$1/\cosh\left[n \log\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)\right] \approx 2\left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^n \approx 2\left[1 - \frac{2}{\sqrt{\kappa}}\right]^n.$$

Using this in the bound (14) for a tolerance ζ, $2[1 - 2/\sqrt{\kappa}]^n \leq \zeta$ implies $n \geq 1/2\sqrt{\kappa}\,\log(2/\zeta)$ so the number of iterations grows as $\sqrt{\kappa}$. A similar calculation for the bound in Equation (15) yields $n \geq \sqrt{\kappa}\,\log(2\sqrt{\hat{\kappa}}/\zeta)/2$.

Recall that for standard finite difference and Galerkin linear finite element schemes applied to second-order elliptic problems, the condition number varies as $O(h^{-2})$ so the number of iterations grows with mesh refinement as $O(h^{-1})$ where h is the mesh size [26]. Similarly, if the first-order system is discretized by first-order differencing or a least-squares linear finite element scheme is applied, the condition number is again $O(h^{-2})$ and the iteration count varies as $O(h^{-1})$. This can be contrasted with the least-squares algebraic system (normal equations) for the discretized higher order problem which has condition number $O(h^{-4})$. Further discussion with numerical results and performance studies are provided later in Section 3. In the next section we briefly outline a parallel partitioning strategy and simple data structure.

© 1998 John Wiley & Sons, Ltd.

3. PARALLEL PCG SOLUTION

In previous studies we have developed a class of element-by-element preconditioned conjugate gradient algorithms and parallel variants of these methods [1]. We have also applied these ideas for least-squares finite element systems and serial processors [13,14,27]. Since these approaches involve either a parallel element-by-element strategy directly or a parallel subdomain form of this algorithm they are not limited to structured grids or to Cartesian processor partitionings. Similarly, the present least-squares finite element scheme can be applied to domain decompositions by elements or nodes involving unstructured grids. However, for simplicity and clarity of presentation in the present work we will consider the structured grid case with a simple Cartesian block partitioning. This permits us to explore and analyse the effect of communication/computation more explicitly and to demonstrate the 'cache management' issue and treatment for the test cases on the Intel parallel system. Our scheme can be applied directly to partitionings on the reference domain for mapped structured grids and block structured grids.

In the present implementation, we locally assemble the nodal contributions so that a stencil-based data structure can be used. Since the grid is structured there is an equivalent (I, J) or (I, J, K) Cartesian reference grid in 2D or 3D respectively. Then a simple Cartesian domain decomposition with natural ordering of the processors can be constructed in the reference domain. Load balancing between processors is also easily achieved. For example, in 2D if there are $G = G_x \times G_y$ grid points and $P = P_x \times P_y$ processors, then a partitioning to approximately $G/P = g$ points per processor is desirable, where $g = g_x \times g_y$, $g_x = G_x/P_x$ and $g_y = G_y/P_y$. This approach will be exact if G_x, G_y are multiples of P_x and P_y. Otherwise, some adjustment of this approximate processor partition will be needed and the number of grid points per processor may vary slightly.

As an example, a 4×4 partition of a square reference domain for 16 processors is shown in Figure 1 with associated subgrids g_i, $i = 0, 1, \ldots, 15$. For each processor subdomain the least-squares finite element matrix and vector contributions are computed. Since the processor

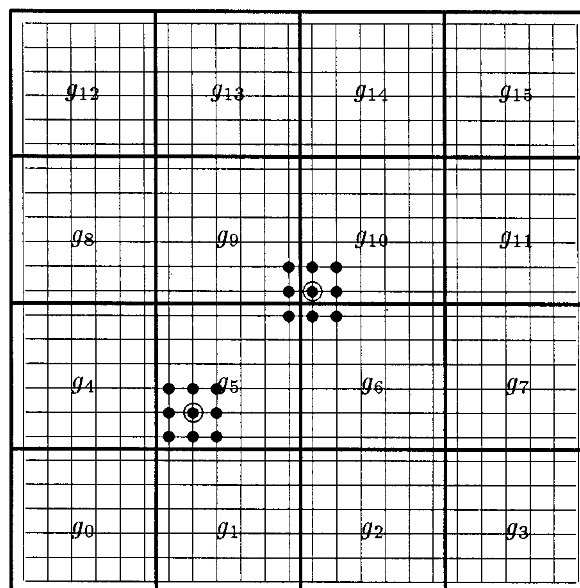

Figure 1. A grid partition with 16 processors.

© 1998 John Wiley & Sons, Ltd.

partition boundaries intersect certain elements, these elements reside in more than one processor. Hence element matrix contributions for these elements are duplicated in parallel on adjacent processors in the partition. This partition could also be interpreted as an overlapping domain decomposition by elements with single layer overlap for the matrix and vector computation. An alternative non-overlapping partitioning by elements could also be used with the processor interface now aligned with element boundaries, which would imply that those nodes on the processor interface would be shared.

As indicated above, a stencil-based data structure is used in the present work. Here the element contributions in each processor subdomain are locally assembled at the nodes. While this approach lacks the convenience of the element-by-element strategy it reduces the number of multiply operations in the conjugate gradient calculations. For the bilinear element basis this implies that contributions from the patch of elements surrounding each interior node are accumulated to yield a centered nine-point stencil. As seen in Figure 1, except for the boundary nodes, which have zero padded stencils, the stencils corresponding to the periphery nodes of each subdomain will overlap the adjacent subdomain, e.g. the stencil marked in g_5 is local, while the one marked in g_{10} overlaps with the nodal values to be supplied from g_5, g_6 and g_9. Since we are dealing with a first-order system and least-squares mixed method this is a block stencil with block size equal to the number of nodal unknowns. For higher degree elements the stencil size will vary locally depending on whether the node is a vertex node, edge node or element interior node. For example, the biquadratic element leads to a 25-point block stencil for each vertex (corner) node, a 15-point block stencil for each edge node and a nine-point block stencil for the interior node. In practice it is simplest to zero-pad the stencil to fixed size since this simplifies the array treatment and efficiency is only slightly degraded. Hence the matrix $\boldsymbol{A}$ may be conveniently stored by stencils (rows) for each processor subdomain. However, we emphasize that the ideas can be extended to treat an irregular partitioning for unstructured grids as shown in our other work on parallel CFD.

Let us first consider parallel distributed computation of the sparse system (10) in the stencil format. Our least-squares formulation is element-based but the processor partition is by grid points as indicated in Figure 1. Thus elements containing the processor partition lines are effectively shared by adjacent processors. Since the data structure and solver are stencil-based, this implies that computations for shared elements are duplicated. A strip of fictitious elements is added at the exterior boundary to avoid special coding for boundary subdomains. Thus the basic computation for parallel matrix formation over the processor subdomains proceeds as follows: for element $e = 1, \ldots, E$ associated with subdomain p_i,

- Compute the element matrix and vector contributions $A_e(i_n, j_n, i_d, j_d)$, $b_e(i_n, i_d)$, where i_n, $j_n = 1, \ldots, N_e$, $i_d, j_d = 1, \ldots, D_p$ with N_e the number of grid points per element and D_p the number of degrees of freedom per grid point.
- Enforce the boundary conditions at the element level in A_e and b_e above.
- Assemble element contributions to stencils for the grid points to generate the submatrix and vector contributions $A(i_x, j_y, i_d, j_d, i_s)$, $b(i_x, i_y, i_d)$ with $i_x = 1, \ldots, g_x$, $i_y = 1, \ldots, g_y$, $i_s = 1, \ldots, S$ where S is the number of points in the stencil and g_x, g_y define the processor grid as before. Natural ordering of grid points is presumed within each processor subdomain.

Having assembled the system locally in parallel in the stencil data structure, solution by conjugate gradient iteration involves repeated matrix–vector products, dot products and DAXPY operations. More specifically, each conjugate gradient iteration involves one matrix–vector product (MV, $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$), two dot products (DOT, $\alpha = \boldsymbol{x} \cdot \boldsymbol{y}$), two DAXPY and one DAYPX operations (DAXY, $\boldsymbol{y} = \boldsymbol{y} + \alpha\boldsymbol{x}$, $\boldsymbol{y} = \boldsymbol{x} + \alpha\boldsymbol{y}$). Assume the subdomain vector length of

© 1998 John Wiley & Sons, Ltd.

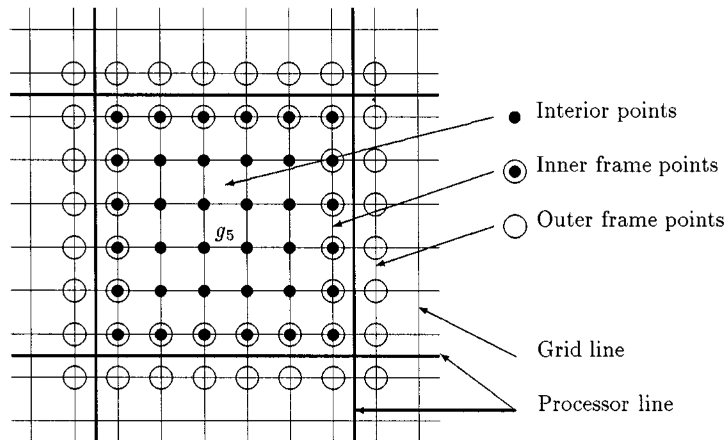

Figure 2. Picture frames on a representative subgrid.

x and ***y*** is $N_v = g_x \times g_y \times D_p$ on each processor. Then the operation counts per CG iteration are $C_{\rm mv} = 2SD_pN_v$ for a matrix–vector product, $C_{\rm dot} = 4N_v$ for two dot products, and $C_{\rm daxy} = 6N_v$ for two DAXPY and one DAYPX computations. Hence, the total operation count for CG with $N_{\rm cg}$ iterations is $C_{\rm cg} = (2SD_p + 10)N_vN_{\rm cg}$. The relative fractions of the operations for MV, DAXY, DOT over CG are $1/(1 + 5/SD_p)$, $3/(SD_p + 5)$ and $2/(SD_p + 5)$. For example, in solving a convection–diffusion problem ($D_p = 3$) using bilinear elements ($S = 9$), The percentages of the operations are 84.385, 9.375, and 6.250% for MV, DAXY, DOT, respectively. Clearly, the matrix–vector product requires the greatest fraction of the computation, and this fraction will increase with increasing D_p.

Since the processor partition associates each grid point with a specific processor, the DAXPY and DAYPX operations can be computed locally in parallel without communication. Local dot products are computed in parallel on the processors and scalar results are then accumulated across processors using global summation with fan-in followed by a broadcast with fan-out (minimum span tree) [28]. The communication for the dot product will increase logarithmically with increasing number of processors. The matrix–vector products are computed in parallel using a special cache mirror treatment together with overlapping of communication and computation [29].

Now, let us examine the relative computation and communication costs related to the matrix–vector product in some detail. To accommodate the stencil overlap at the processor partition boundaries we introduce a related domain decomposition which is referred to here for convenience in interpretation as the 'picture frame' representation shown in Figure 2. For the bilinear elements considered here, any grid point is connected to, at most, its eight neighbors in the associated four-element patch. The grid points of Figure 2 interior to the picture are marked (●) and their stencils are local to the processor. Those in the inner frame marked (⊙) are referenced by stencils centered on neighbor processors and those in the outer frame marked (○) are referenced by inner frame stencils centered on the present processor. Hence the inner-frame consists of the single row (or column) of nodes next to each of the processor lines. These data for the inner-frame points can be exchanged with the data for the outer frame points at the eight adjacent neighbor processors as indicated in Figure 3.

This exchange is first performed asynchronously as follows: first post the receive buffers as targets for the neighbor processors and then send the values on the inner picture frame nodes.

© 1998 John Wiley & Sons, Ltd.

In order to prevent the data from arriving before the receive buffer is posted, a zero byte synchronization is utilized between posting the receives and performing the sends. The MV computation for the interior points is initialized as communication starts so that the communication and computation are overlapped. After completing the part of HV for the interior points and receiving all the data from the outer frame points, the remaining MV calculations for the inner frame points are carried out.

To begin, let us assume the communication and computation do not overlap, i.e. the synchronous algorithm is used; the time to compute n floating point operations is $n\gamma$; the time to exchange n values is $\alpha + n\beta$, where α is mean start-up time and β is per item mean transfer time. Here, the mean start-up or transfer time can be taken as the average between any two processors (neighbor or non-neighbor) since worm-hole routing applies in the iPSC/860 hypercube architecture used in the present work. As an example, consider the grid partition with 16 processors in Figure 1; the time to exchange data from all neighbor processors is

$$T_{\mathrm{mv}c} = 2[2\alpha + (g_x + g_y)D_p\beta] + 4(\alpha + D_p\beta),$$

where the first part is for the four edge neighbors and the second part is for the four corner neighbors, as shown in Figure 3. The computation time for the matrix–vector product is

$$T_{\mathrm{mv}i} = 2(g_x - 2)(g_y - 2)D_pS\gamma,$$

for the interior points and

$$T_{\mathrm{mv}if} = 2(g_x + g_y - 2)D_pS\gamma,$$

for the inner frame points. Hence, the total time for MV with synchronous communication would be

$$T_{\mathrm{mv}sy} = T_{\mathrm{mv}if} + T_{\mathrm{mv}i} + T_{\mathrm{mv}c}. \tag{16}$$

Each subgrid on a 2D grid will communicate with at most eight neighbor subgrids, therefore, no extra communication cost will be introduced when $P \geq 16$. Hence, this algorithm is effective when using large numbers of processors to solve large scale problems.

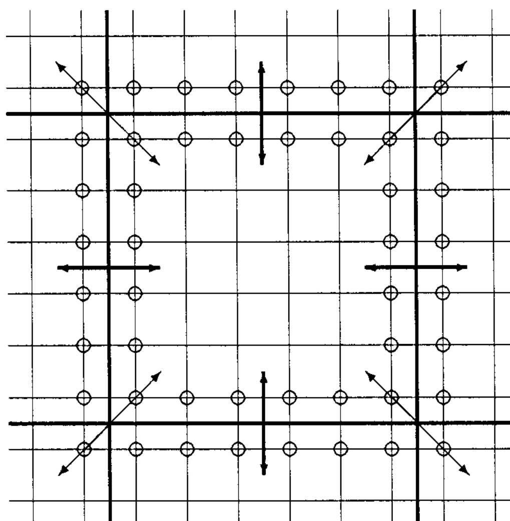

Figure 3. Communications for inner frame points.

© 1998 John Wiley & Sons, Ltd.

Table I. Estimated timing for MV with 16 processors

Grid ($g_x \times g_y$)	Time (ms)			
	T_{mv}	T_{mvif}	T_{mvi}	T_{mvc}
10×10	1.98	0.078	0.277	1.62
23×23	4.03	0.190	1.91	1.93
24×24	4.25	0.199	2.09	1.96
50×50	13.0	0.423	9.95	2.58
90×90	37.7	0.769	33.4	3.54

Next, let us consider the case in our implementation, where the communication time T_{mvc} and the computation time T_{mvi} are overlapped using asynchronous communication as discussed previously. The total time for MV is then reduced to

$$T_{mvasy} \approx T_{mvif} + \max(T_{mvi}, T_{mvc}). \tag{17}$$

In the estimate shown later, $T_{mvi} > T_{mvc}$ for medium to large size problems and communication cost can be ignored.

Finally, the two DAXPY and one DAYPX computations require

$$T_{daxy} = 6g_x g_y D_p \gamma, \tag{18}$$

and the two dot products require

$$T_{dot} = 4g_x g_y D_p \gamma + 2\log_2(p)(\alpha+\beta+\gamma) + 2\log_2(p)(\alpha+\beta), \tag{19}$$

where the first part is for the local dot product computation, the second part is for the global summation of the scalar result using the minimum span tree (fan-in), and the last part is for the broadcast of the scalar result using the minimum span tree (fan-out).

As an example, consider the solution of a convection–diffusion problem using the least-squares mixed method ($D_p = 3$) using a 4×4 partition of processors with $g_x = g_y$. Assuming $\alpha = 170$ μs, $\beta = 2$ μs and $\gamma = 0.08$ μs estimated from the CG timing experiment (see Tables V and VI, γ is from the CG rate of 12.54 Mflops s^{-1} with one processor; α and β are from the DOT rate of 9.21 Mflops s^{-1} with 16 processors). The estimated time for one matrix–vector product with synchronous communication is listed in Table I. When $g_x = g_y = 24$, the computation time for interior points is about the same as the frame communication time, which is 46% of the overall MV time. For this medium problem, using asynchronous communication yields approximately a 46% improvement over using synchronous communication. For a larger subdomain problem with $g_x = g_y = 90$, the communication time is 9% of the overall MV time, and the asynchronous communication still offers some improvement. Hence, the benefits of

Table II. Estimated timing for CG with 16 processors

Method	Time (ms)	Proc. rate (Mflops s^{-1})	Overall rate (Mflops s^{-1})
CG	59.96	11.19	179
MV	37.77	11.33	181
DAXY	11.66	12.50	200
DOT	10.53	9.23	148

© 1998 John Wiley & Sons, Ltd.

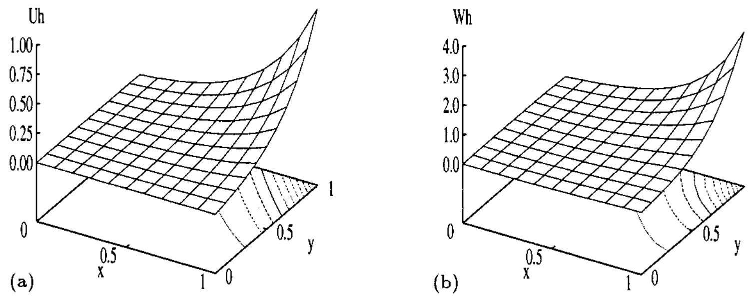

Figure 4. LSFECG solution for a convection–diffusiion problem.

asynchronous communication will be most noticeable for the fixed problem size speed-up as g_x, g_y reduce with increasing number of processors.

For the same problem, the estimated time and rates for one CG iteration with synchronous communication are listed in Table II for $g_x = g_y = 90$, MV, DAXY and DOT take 63, 19 and 18% of the CG time respectively. Compared with our previous operation count percentages of computation as 84.385, 9.375 and 6.250%, we conclude that the MV implementation is relatively efficient, since 84% of CG computation is estimated to take only 63% of the CG time. The DAXY and DOT are less efficient, compared with MV. The processor rate for CG is 11.19 Mflops s^{-1}. MV runs at 11.33 Mflops s^{-1}. The DAXY rate (12.5 Mflops s^{-1}) is better than the MV rate, and the DOT rate (9.23 Mflops s^{-1}) is worse. When more processors are used, the MV and DAXY rates will remain the same, but the DOT rate will decrease. For 16 processors with synchronous communication, the overall rate for CG is 179 Mflops s^{-1} and this will improve with the use of asynchronous communication on the matrix–vector product.

4. RESULTS

4.1. Test problems and LSFECG solutions

The first test problem is the stationary convection–diffusion problem

$$-\nabla \cdot (a\nabla u) + \boldsymbol{c} \cdot \nabla u = 0, \qquad (x, y) \in [0, 1] \times [0, 1], \tag{20}$$

$$u(x, 0) = 0, \qquad u(x, 1) = e^{2x-2} \sinh(x)/\sinh(1), \quad x \in [0, 1],$$

$$u(0, y) = 0, \qquad u(1, y) = e^{y-1} \sinh(2y)/\sinh(2), \quad y \in [0, 1],$$

where $a = 1$, $\boldsymbol{c} = (4, 2)^T$ and the exact solution is

$$u(x, y) = e^{2x+y-3} \frac{\sinh(x)\sinh(2y)}{\sinh(1)\sinh(2)}.$$

The corresponding first-order system is Equation (3) and the least-squares finite element solutions compare favorably with the exact solution [27]. Representative solutions for u_h and w_h ($\boldsymbol{\sigma} = (v, w)^T$) are shown in Figure 4 as a surface plot and as projected contours for uniprocessor computations on a 10×10 mesh of linear elements with 2×2 Gauss integration.

The next problem considered is stationary viscous flow governed by

$$\nabla \cdot \boldsymbol{u} = 0, \qquad \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho} \nabla p - \nu \Delta \boldsymbol{u} = \boldsymbol{f}, \tag{21}$$

© 1998 John Wiley & Sons, Ltd.

where $\boldsymbol{u}$ is velocity, p is pressure, v is kinematic viscosity, ρ is constant density and $\boldsymbol{f}$ is the body force term. We introduce the vorticity and rewrite this as the first-order system

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \qquad u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial x}+v\frac{\partial \zeta}{\partial y}=f,$$

$$\zeta-\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}=0, \qquad u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial y}-v\frac{\partial \zeta}{\partial x}=g. \tag{22}$$

Other formulations are also admissible, e.g. the stress components can be introduced explicitly [13].

As a representative example, the flow of glycerine with $v=6$ cm^2 s^{-1} in a cavity of size 10×6 cm was computed for the case where the vertical side walls are moving in opposite directions at a velocity of 10/6 cm s^{-1}. The equations are dimensionally scaled and the solution is computed on a 20×20 mesh of bilinear elements (equal order basis for all variables). Representative solutions are shown in Figure 5 for (a) streamlines and (b) vorticity, where the streamlines are computed from the velocity field. Further results and mixing calculation are given in References [30,27].

The problem is non-linear; therefore an iterative strategy is introduced. Here we use successive approximation (i.e. set $u(\partial u/\partial x)$ as $u^{(s)}(\partial u^{(s+1)}/\partial x)$) directly in the residual and then the least-squares system is constructed for this linearized problem. For iterate s, the linear algebraic system (10) becomes

$$\boldsymbol{A}^{(s)}\boldsymbol{x}^{(s+1)}=\boldsymbol{b}^{(s)}, \tag{23}$$

and PCG is applied to this system as before. Then $\boldsymbol{A}^{(s)}$ is updated and the PCG step is repeated using the current non-linear iterate as starting vector. As non-linear successive approximation proceeds, the number of PCG iterations reduces accordingly.

4.2. PCG convergence

For the purpose of the present work, the iterative performance was first compared with banded Gaussian elimination on a sequence of meshes with $h=1/20$, 1/30, 1/40, 1/50 for linear elements on the model convection–diffusion problem (20). Uniprocessor results of this comparison study on the CRAY/YMP are given in Figure 6 for computations with bilinear elements. Performance results with biquadratic elements are essentially the same.

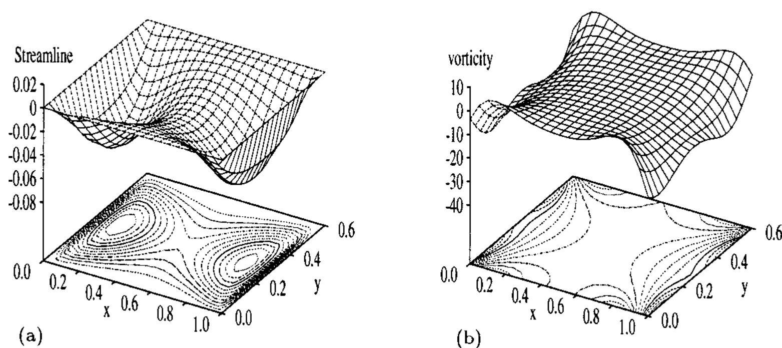

Figure 5. LSFECG solution for a viscous flow problem.

© 1998 John Wiley & Sons, Ltd.

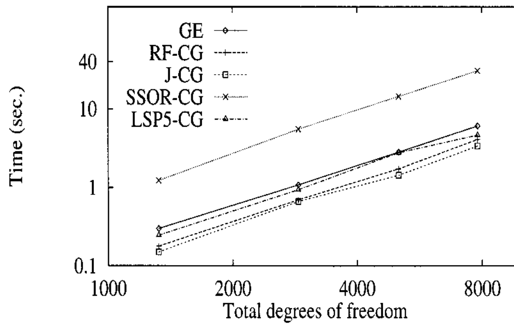

Figure 6. Timing for PCG on CRAY/YMP.

Estimates of the extreme eigenvalues $\lambda_{\max}$, $\lambda_{\min}$ can be computed using the tridiagonal part of $\boldsymbol{Q}^{-1}\boldsymbol{A}$ and used to compute a condition number estimate κ^*. A graph of this estimated condition number as a function of mesh size and the actual number of iterations (n) to achieve convergence within a tolerance of 1.5×10^{-4} is provided in Figure 7. The respective slopes are 1.9 and 0.81, which implies that the number of iterations is approximately $O(h^{-1})$ and is consistent with the estimates stated earlier.

4.3. *Performance versus grid size and layout*

Before we consider the parallel algorithm, uniprocessor performance as a function of grid size is first studied for a sample grid block $g_x \times g_y$ with $g_x = g_y$. This will give the base performance in terms of problem size. The effect of other choices of grid layout, i.e. $g_x > g_y$ or $g_x < g_y$, is then studied.

The problem size is $D_t = g_x \times g_y \times D_p$, where D_t is the total number of degrees of freedom. The convection–diffusion problem ($D_p = 3$) is first computed using $g_x = g_y = 10$, 20, 40, 60, 80 and 90. Then the viscous flow problem ($D_p = 4$) is computed with $g_x = g_y = 9$, 17, 35, 52, and 70 so that the number of total degrees of freedom D_t are close at each grid level for both problems. The matrix–vector product has been optimized here in assembly code for the iPSC/860 and Paragon processors [29]. Only the results for the iPSC/860 will be presented here. The behavior on other processors with different cache management will be different. The CG rates on a single processor of the iPSC/860 increase from 6.38 to 12.54 mflop s^{-1} as the total number of degrees of freedom increases from 300 to 24300 as indicated in Figure 8. Performance increases significantly when the problem size increases and is good when there is

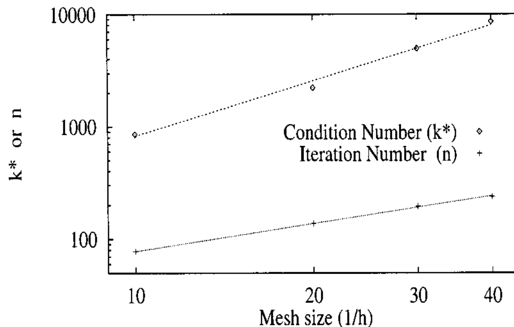

Figure 7. Condition and iteration numbers for J-CG.

© 1998 John Wiley & Sons, Ltd.

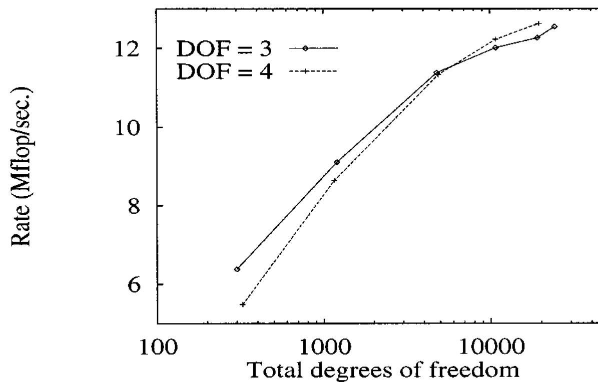

Figure 8. CG rates versus problem sizes on iPSC/860.

enough computation; e.g. the processor rate is 9.11 mflop s^{-1} on a 20×20 grid with three DOF per grid point. Moreover, the performance is not sensitive to increasing the number of degrees of freedom per grid point. This is very promising for applications involving many degrees of freedom per grid point, such as the simulation of high speed non-equilibrium chemically reacting gas flows in which there are several degrees of freedom per grid point [31].

Next, the effect of different choices of g_x and g_y is studied for the convection–diffusion problem ($D_p = 3$) on a fixed global grid size with $g = 6084$. The time and rates for 500 CG iterations on one processor of the iPSC/860 are indicated in Table III for several grid layouts. The timing shows that the CG time increases 8% as g_x is reduced from 234 to 26. The time for DAXPY, DAYPX and DOT operations are approximately the same for different choices of grid layout, but the time for MV varies by up to 10%. This is because the corresponding layout of the matrix, i_x ($= 1, \ldots, g_x$) goes first in $A(i_x, i_y, i_d, j_d, i_s)$ [21]. Hence, selecting a large g_x during mesh partitioning will give a better performance on one processor or on multi-processors if there is a full overlap. However, if the number of processors is increased to the point where communication overlap is not achieved, then the non-overlap part of communication cost for MV is minimized when $g_x = g_y$ since it is proportional to the perimeter $2(g_x + g_y)$ of the subgrid as estimated in Equation (17). As g_x is reduced in the above test, the perimeters vary as 520, 396, 312, 396, 520, and the minimum perimeter corresponds to the square grid $g_x = g_y = 78$. Since the time difference with respect to grid layout is small for CG computation alone, a better performance would be expected on a square subgrid for multi-processor computation with communication cost. The MV rates vary from 12.02–13.41 Mflop s^{-1} and

Table III. Timing and rate versus grid layout on iPSC/860

Grid ($g_x \times g_y$)	Time (s)						Rate (Mflops s^{-1})	
	DAXPY	DAYPX	DOT	MV	CG	LSFECG	MV	CG
234×26	3.57	4.20	2.16	36.83	46.80	54.07	13.41	12.50
156×39	3.57	4.15	2.15	36.97	46.89	53.92	13.36	12.48
78×78	3.57	4.15	2.15	37.57	47.49	54.54	13.14	12.32
39×156	3.57	4.47	2.15	38.68	48.92	56.09	12.77	11.96
26×234	3.57	4.12	2.16	41.08	50.99	58.16	12.02	11.48

© 1998 John Wiley & Sons, Ltd.

Table IV. Performance on a fixed global grid for C-D problem

Processor partition ($P_x \times P_y$)	LSFECG	CG		
	Time (s)	Time (s)	Proc. rate (Mflop s^{-1})	Speed-up (S_t)
1×1	62.41	53.33	12.40	1.00
1×2	32.56	27.93	11.84	1.91
2×2	17.61	15.27	10.83	3.49
1×4	17.66	15.34	10.78	3.48
2×4	10.14	8.96	9.22	5.95
1×8	10.16	8.99	9.20	5.93
4×4	6.61	6.00	6.90	8.89
4×8	4.79	4.48	4.62	11.90

CG rates vary from 11.48–12.48 Mflop s^{-1} with different choices of layout. DAXPY,DAYPX and DOT rates are near 10.26, 4.34 and 16.9 Mflop s^{-1} respectively in all cases and therefore will not impact the performance under different choices of layout.

The above computation uses our special assembly-code kernel to manage the cache so that the MV is computed with multi-output rows at a time which uses the cache more effectively [29]. The Intel i860 on the iPSC/860 has a cache which can hold 1000 double precision numbers. If a matrix–vector product generates one output row from three input rows in cache, the maximum length of the row is 250. When the vector length is less, the performance is degraded since the cache is not fully used. Under above test conditions and computing one output row at a time, MV time increases by 39%, and CG time increases by 33% as g_x is reduced. In the multi-output row implementation, the variations of MV, CG and total time (LSFECG) are only 10, 8 and 7% respectively, for different choices of grid layout.

The above results show that grid layout moderately affects the performance on a single processor for large problems. If $g_y > g_x$, then the orientation of the axes should be reversed so that the natural ordering occurs in the vertical direction. For small problems, communication is a major consideration so that a square subgrid will minimize the communication cost as we show later.

4.4. Parallel performance and scalability

The convection–diffusion problem (20) was next solved on multiple processors of the Intel iPSC/860 distributed system. The global grid size is fixed with $G_x = G_y = 88$, while the local grid points vary with the different processor partitions $P_x \times P_y$. For iteration tolerance 0.0001, the CG scheme converged in 444 iterations for P ranging from 1 to 32 processors. The speed-up can be simply defined as $S_t = T_1/T_p$, where T_p is the computational time using P processors. The parallel performance results are listed in Table IV. When two processors are used, the time is almost reduced to half of the uniprocessor computation, and the speed-up is 1.91 which is close to the ideal speed-up of 2. As more processors are used, the speed-up is progressively degraded because the ratio of communication time to computation time increases. This is demonstrated by the tabulated results and by the graph in Figure 9. When 32 processors are used, the speed-up is 11.90 for CG and 13.03 for the total time (LSFECG), respectively. Comparing the details of the speed-up for CG, MV, DAXPY, DAYPX and DOT, they are 1.91, 1.93, 1.98, 1.84 and 1.63 when using two processors over one processor, and they reduce to 1.34, 1.55, 1.29, 2.02 and 0.97 when using 32 processors over 16 processors. Notice

© 1998 John Wiley & Sons, Ltd.

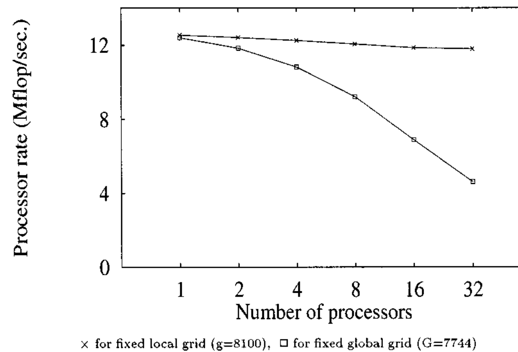

Figure 9. The scalability of CG on iPSC/860.

that DOT has no speed-up for the 32 processor computation because the communication time is longer than the computation time on each processor due to the small subgrid size. In addition, the percentage of DOT time relative to CG time increases from 4.51% (one processor) to 35.77% (32 processors), while the percentage of MV time to CG time reduces from 72.22% (one processor) to 55.39% (32 processors). Therefore, the degradation of the speed-up when using more processors mainly comes from the dot product.

Recall that the performance with one processor is better for a layout with larger g_x. Here, we see that the performance is better for a square subgrid because of the smaller number of nodes on the subgrid boundary. For example, the rate of 10.78 mflop s^{-1} for a 1×4 processor partitioning $[2(g_x + g_y) = 220]$ is not higher than the rate of 10.83 mflop s^{-1} for a 2×2 processor partitioning $[2(g_x + g_y) = 176]$. A similar situation is observed when using more processors.

The idea of scaled speed-up has been popularized as an alternative measure of parallel capability. In this case the problem size is scaled proportionally with the number of processors so that the processor grid size is fixed. Here, we take $g_x = g_y = 90$ for each processor. The scaled speed-up is then defined as $S_r = P \times R_p/R_1$, where R_p is the processor rate (Mflop s^{-1}) when using P processors. The scheme yields 94.09% of the ideal scaled speed-up for 32 processors as indicated in Table V. The time for the CG solution phase is for 500 CG iterations. Obviously, the fine global grid will require more iterations for convergence than the coarser global grids at lower P values. The processor rates are slightly reduced due to the communication, as shown in Figure 9 for this fixed local grid size computation.

Table V. Performance for fixed local grid on iPSC/860 for C-D problem

Processor partition ($P_x \times P_y$)	LSFECG	CG		
	Time (s)	Time (s)	Proc. rate (Mflop s^{-1})	Speed-up (S_r)
1×1	72.01	62.12	12.54	1.00
1×2	72.84	62.78	12.41	1.98
2×2	73.71	63.60	12.25	3.91
2×4	74.70	64.58	12.06	7.69
4×4	75.79	65.67	11.86	15.13
4×8	76.14	66.01	11.80	30.11

© 1998 John Wiley & Sons, Ltd.

Table VI. Performance detail for CG with 16 processors

Method	Time (s)	Proc. rate (Mflops s^{-1})	Overall rate (Mflops s^{-1})
CG	65.67	11.86	189.8
MV (total)	50.00	13.15	210.4
MV (interior)	45.17	13.92	222.7
HV (frame)	4.83	5.99	95.8
DAXPY	4.76	10.22	163.5
DAYPX	5.57	4.37	69.9
DOT	5.29	9.21	147.4

The performance for 16 processors is summarized in Table VI. The CG, MV, DAXY, and DOT rates in previous estimates (Table II) basically predict the real rates in Table VI. The real rate for MV is higher than the estimate due to the use of cache and asynchronous communication. The rate for DOT is about the same as the estimate. The rate for DAXY is lower than the estimate due to the implementation.

Similar parallel performance was observed for the viscous flow problem (22). Again we consider the question of scaled speed-up for a sequence of uniform grids with fixed subgrid problem size per processor. Performance results are listed in Table VII for the iPSC/860 and are comparable with those in Table V. The number of grid points per processor is fixed as $g_x = g_y = 70$, but with four degrees of freedom per node. CPU times here are again reported for 500 CG iterations. Note that the number of nodal degrees of freedom here has increased from three in the previous problem to four, but the performance is essentially the same and 94.28% of the ideal scaled speed-up is achieved for 32 processors.

The above calculations show that the processor rate is slightly reduced for $P < 16$, e.g. $\approx 5\%$ degradation for both problems in Tables V and VII. The processor rate almost remains constant with further increase in the number of processors as expected. Therefore, not only is the computation cost low in each processor with the use of cache, but the communication cost is also low with the use of asynchronous communication for MV. This algorithm is especially efficient when using a moderate to large number of processors. The scaled speed-up is 94% of the ideal speed-up, which is good in the sense of fixed memory utilization [32]. This can be compared with the NAS parallel benchmark for CG with parallelization based on matrix partition for a model unstructured grid [3]. In their study the rates in Mflops s^{-1} have been reported as: 215 for the Intel iPSC/860 with 128 processors, 457 for the Intel Paragon with 128

Table VII. Performance for fixed local grid on iPSC/860 for N-S problem

Processor partition ($P_x \times P_y$)	LSFECG	CG		
	Time (s)	Time (s)	Proc. Rate (Mflop s^{-1})	Speed-up (S_r)
1×1	68 25	63.82	12.62	1.00
2×1	68.98	64.49	12.48	1.98
2×2	69.78	65.26	12.34	3.91
2×4	70.69	66.17	12.17	7.71
4×4	71.91	67.42	11.94	15.14
4×8	72.16	67.67	11.90	30.17

© 1998 John Wiley & Sons, Ltd.

processors, 127 for the CRAY/YMP with one processor, 5178 for the CRAY/C90 with 16 processors, and 464 for the CRAY/T3D with 128 processors, respectively. Our rate for CG with parallelization based on a simple domain partition for structured grids is 377 Mflops s^{-1} for 32 processors as shown in Table V, which scales to a rate of 1384 Mflops s^{-1} for 128 processors. Our CG rate scaled to 128 processors is seven times higher than that of the NAS CG on Intel iPSC/860. This is due to two factors: (1) on each processor our cache treatment is superior. The NASCG study uses the Yale sparse matrix format which is inefficient compared with the compact stencil format. This is a gather/scatter issue and we show the need to stream the data through the cache to obtain a good performance. For the i860 class processor, implementing the cache mirror idea requires assembly coding. Our performance here is four times faster than in NASCG. (2) The second factor is relative communication performance. Our approach utilizes domain decomposition to partition the grid to processors whereas the NASCG study uses row/column matrix partitioning strategies. This implies that we can overlap stencil communication on the processor interfaces with computation in the interior. Since our partitioning implies that few off-processor references are needed, communication overhead is reduced. More specifically, our communication is between adjacent subdomains, which in practice will be few in number for any rational decomposition strategy. On the other hand, partitioning the matrix (as in NASCG) may lead to less efficient communication. For example, it has been demonstrated that matrix partitioning is an inefficient approach and does not scale independently of whether partitioning is by rows and/or columns [4]. This factor, in addition to the cache efficiency, results in the factor of seven difference in performance for the two approaches.

The approach can be extended to use an element-by-element CG scheme for unstructured grids [1,14]. Since the parallelization is still based on a simple domain partition, the performance should be similar.

5. CONCLUDING REMARKS

Least-squares mixed finite elements for flow and transport problems are appealing because they have interesting approximation properties and generate symmetric positive systems. It follows that conjugate gradient schemes can be applied to solve these systems and their iterative performance is therefore of interest. Part of the present study compares CG performance with different preconditioning strategies against Gaussian elimination for problems of increasing size. The conditioning of the system is shown to degrade linearly as the mesh is refined when simple Jacobi (diagonal) preconditioning is applied. We also develop and test a parallel implementation and provide scaled speed-up results. The scheme is demonstrated for representative convection–diffusion and Navier–Stokes applications.

The scalability properties of an algorithm are important because they indicate how many processors must be added so that run time remains constant as the problem size is increased. A full scalability analysis is useful because it provides a comparison of the efficiency of the algorithm on different machine architectures. However, as we see here, detailed consideration such as the design of the inner MV kernel and partition layout can obscure true parallel performance. A comparison with the NASCG results in Reference [3] shows that the present scheme is seven times faster. This is due to the superior cache treatment and communication treatment. There are also a number of different types of scalability measures that reflect speed-up and resource relations. This includes not only the fixed problem size and scaled speed-up considered here, but also the fixed run time and memory scalability, among others.

© 1998 John Wiley & Sons, Ltd.

Some of these ideas and related iso-efficiency concepts are considered in Reference [33]. We remark in closing that the approaches for cache management and communication are not restricted to the present least-squares formulation but can be used with other discretization methods and other generalised gradient-type iterative solvers. We also emphasize that the approach can be extended to a subdomain element-by-element or similar framework for unstructured grids on irregular subdomain partitionings.

ACKNOWLEDGMENTS

This research has been support in part by ARPA grant #DABT63-92-C-0024 and by the Texas Advanced Research Program. We express our appreciation to Wayne Joubert and the other members of the PCG research group.

REFERENCES

1. E. Barragy, G.F. Carey and R. van de Geijn, 'Performance and scalability of finite element analysis for distributed parallel computation', *J. Parallel Distrib. Comput.*, **21**, 202–212 (1994).
2. T. Chan (ed.), *Proc. Second Int. Symp. on Domain Decomposition Methods for PDE's*, SIAM, Philadelphia, 1988.
3. D.H. Bailey, E. Barszcz, L. Dagum and D. Simon, 'NAS parallel benchmark results 3-94', *RNR Technical Report RNR-94-006*, 1994.
4. J. Lewis and R. van de Geijn, 'Distributed memory matrix–vector multiplication and conjugate gradient algorithms', *Proc. Supercomputing '93*, Portland, OR, Nov. 15–19, 1993.
5. B. Hendrickson and R. Leland, 'Multidimensional spectral load balancing', *Technical Report SAND93-0074*, Sandia National Laboratories, January 1993.
6. J.H. Bramble, R.D. Lazarov and J.E. Pasciak, 'A least-squares approach based on a discrete minus one inner product for first order systems', submitted.
7. B.A. Finlayson, *The Method of Weighted Residuals and Variational Principles*, Academic Press, New York, 1972.
8. O.C. Zienkiewicz, D.R.J. Owen and K.N. Lee, 'Least-square finite element for elasto-static problems: Use of "reduced" integration', *Int. J. Numer. Methods Eng.*, **8**, 341–358 (1974).
9. T. Arbogast, C.N. Dawson, P.T. Keenan, M.F. Wheeler and I. Yotov, 'Implementation of mixed finite element for elliptic equations on general geometry', to appear, 1995.
10. F. Brezzi and M. Fortin, *Mixed and Nonconforming Finite Element Methods*, Springer, Berlin, 1991.
11. R.E. Ewing, R.D. Lazarov and J. Wang, 'Superconvergence of the velocity along the Gauss lines in mixed finite element methods', *SIAM J. Numer. Anal.*, **28**, 1015–1029 (1991).
12. P.B. Bochev and M.D. Gunzburger, 'Analysis of least-squares finite element methods for the Stokes equations', *Math. Comp.*, **63**, 479–506 (1994).
13. G.F. Carey, A.I. Pehlivanov, Y. Shen, A. Bose and K.C. Wang, 'Least-squares finite elements for fluid flow and transport', *Proc. Finite Elem. Fluids*, October, Italy, 1995.
14. B.N. Jiang and G.F. Carey, 'Adaptive refinement for least square finite elements with element-by-element conjugate gradient solution', *Int. J. Numer. Methods Eng.*, **24**, 569–580 (1987).
15. G.F. Carey and Y. Shen, 'Convergence studies of least-squares finite elements for first-order systems', *Comm. Appl. Numer. Methods*, **5**, 427–434 (1989).
16. A.I. Pehlivanov and G.F. Carey, 'Error estimates for least-squares mixed finite elements', *RAIRO Math. Model. Numer. Anal.*, **28**, 499–516 (1994).
17. M.G. Sheu, 'On the theory of mixed finite element approximations of boundary-value problems', *Comp. Meth. Appl. Mech. Eng.*, **4**, 333–347 (1978).
18. G.F. Carey, A.I. Pehlivanov and P.S. Vassilevski, 'Least-squares mixed finite element methods for non-selfadjoint elliptic problems: II. Performance of block-ILU factorization methods', *SIAM J. Sci. Comput.*, to appear.
19. T.F. Chan and P.S. Vassilevski, 'A framework for block-ILU factorizations using block size reduction', *Math. Comp.*, **64**, 129–156 (1995).
20. W. Joubert and G.F. Carey, 'PCG: A software package for the iterative solution of linear systems on scalar, vector and parallel computers', in R.F. Sincovec, D.E. Keyes, M.R. Leuze, L.R. Petzold, D.A. Reed (eds.), *Proc. 6th SIAM Conf. on Parallel Processing for Scientific Computing*, Philadelphia, SIAM, 1993, pp. 515–518.
21. W.D. Joubert, G.F. Carey, N.A. Berner, A. Kalhan, H. Kohli, A. Lorber, R.T. McLay and Y. Shen, 'PCG reference manual—a package for the iterative solution of large sparse linear systems on scalar, vector and parallel computers', *CNA-274*, *Center for Numerical Analysis*, University of Texas at Austin, 1995.
22. S.F. Ashby, T.A. Manteuffel and P.E. Saylor, 'A taxonomy for conjugate gradient methods', *SIAM. J. Numer. Anal.*, **27**, 1542–1568 (1990).
23. L.A. Hageman and D.M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

© 1998 John Wiley & Sons, Ltd.

24. M.R. Hestenes and E.L. Stiefel, 'Methods of conjugate gradients for solving linear systems', *J. Res. Nat. Bur. Stand.*, **49**, 409–436 (1952).
25. W. Joubert, 'Iterative methods for the solution of nonsymmetric systems of linear equations', *CNA-242*, *Center for Numerical Analysis*, The University of Texas at Austin, 1990.
26. G.F. Carey and J.T. Oden, *Finite Elements: Computational Aspects*, Vol. 3, Prentice Hall, Englewood Cliffs, NJ, 1984.
27. Y. Shen, 'Least-squares finite element solution and mixing simulation', *Ph.D. Dissertation*, University of Texas at Austin, 1993.
28. M. Barnett, R. Littlefield, D. Payne and R. van de Geijn, 'On the efficiency of global combine algorithms for 2-D meshes with wormhole routing', *J. Parallel Distrib. Comput.*, **24**, 191–201 (1995).
29. R. McLay, S. Swift and G.F. Carey, 'Maximizing sparse matrix–vector product performance in MIMD computers', *J. Parallel Distrib. Comput.*, in press (1996).
30. G.F. Carey and Y. Shen, 'Simulation of fluid mixing using least-squares finite element and particle tracing', *Int. J. Numer. Methods Heat Fluid Flow*, **5**, 549–573 (1995).
31. C. Harle, G.F. Carey and P.L. Varghese, 'Analysis of high speed nonequilibrium chemically reacting gas flows, Part II: Finite volume/finite element model and numerical studies', *Int. J. Numer. Methods Fluids*, (submitted) 1995.
32. J.L. Gustafson, G.R. Montry and R.E. Benner, 'Development of parallel methods for a 1024 processors hypercube', *SIAM J. Sci. Stat. Comput.*, **9**, 609–638 (1988).
33. G.F. Carey, J. Schmidt, V. Singh and D. Yelton, 'A prototype scalable, object-oriented finite element solver on multicomputers', *J. Parallel Distrib. Comput.*, **20**, 357–379 (1994).

© 1998 John Wiley & Sons, Ltd.