COMMUNICATIONS IN APPLIED NUMERICAL METHODS, Vol. 8, 573–583 (1992)

A COMPARISON OF FAC AND PCG METHODS FOR SOLVING COMPOSITE GRID PROBLEMS

MICHAEL A. HEROUX
Mathematical Software Research Group, Cray Research, Inc., 655F Lone Oak Drive, Eagan, MN 55121, U.S.A.

AND

J. W. THOMAS
Department of Mathematics, Colorado State University, Ft. Collins, CO 80523, U.S.A.

SUMMARY

The fast adaptive composite grid method (FAC) is a multi-level adaptive method for the discretization and solution of partial differential equations. This paper compares the speed of solution of the resulting discrete equations by the FAC algorithm and a preconditioned conjugate gradient scheme. Comparisons and information concerning the solutions are given for several model problems.

1. INTRODUCTION

The fast adaptive composite grid method (FAC) is a multi-level adaptive method for discretization and solution of partial differential equations. The scheme provides a discretization of the given partial differential equations on a composite grid which consists of an underlying uniform grid and patches of uniform finer grids aligned with some underlying coarser grid. The method also specifies an algorithm for the fast solution of the discrete equations on the composite grid.

The FAC solution algorithm consists of solving on the various levels several times. The speed of the FAC algorithm depends on the speed of the solvers used on the different uniform grids. The advantage of using FAC is that all of the solving is done on uniform grids. However, there is always a concern that it may be possible to solve the FAC discrete equations faster by other methods. It is of special interest to see if there are library packages that are competitive with the FAC algorithm. Of course, for that to be the case, the package would have to be capable of handling large, sparse, non-uniform matrices.

In this paper we compare FAC as a solution scheme with the preconditioned conjugate gradient solver SITRSOL. SITRSOL is a Cray library package designed to accept large, sparse, unstructured matrices. Written by one of the authors, SITRSOL has been optimized to run on the Cray YMP.

After giving a short description of the preconditioned conjugate gradient algorithm, we provide a brief description of the residual correction, linear FAC algorithm. We then briefly describe the preconditioned conjugate gradient solver SITRSOL. In Section 4 we introduce several model problems that we will solve by FAC (using a preconditioned conjugate gradient method for regular grid problems as the solver on both the coarse and fine grids) and by using SITRSOL directly on the composite grid problems. For one of the problems (one for which

0748–8025/92/090573–11$10.50

Received May 1991
Revised October 1991

we have an exact solution), we carefully calculate truncation error. We do this so that we can solve the problem by both methods to truncation error and use the results to illustrate that it would not be advantageous to use a FAC preconditioned conjugate gradient for these problems. This is because two FAC steps are sufficient to solve the problem to within truncation error. Thus, any conjugate gradient acceleration would introduce needless overhead.

The solution results show that, while the FAC solution takes many more conjugate gradient steps than the SITRSOL solution, the FAC solution is always faster than the direct SITRSOL solution. It is generally about 25 to 45 per cent faster to solve the problems by FAC than to solve them directly using SITRSOL. One of the interesting results – and we do not know whether this would generally be the case – is that for Example Problem 1, the condition number of the composite grid matrix is approximately one and a half times bigger than either the fine grid or coarse grid matrices. So not only does solving the composite grid equations directly force one to handle non-uniform matrices, it also requires that a more difficult linear system be solved.

Before describing the two solution techniques, we first give a brief summary of the preconditioned conjugate gradient method since it will play an important role in both techniques. For a more detailed description, see Golub and Van Loan[1] Given a linear system $\mathbf{Ax} = \mathbf{b}$ of dimension n, an initial guess, $\mathbf{x}^0$ for $\mathbf{x}$, and a preconditioner $\mathbf{M} \sim \mathbf{A}^{-1}$, the algorithm $\mathbf{PCG(A, M, x^0, b)}$ is as follows:

PCG$(\mathbf{A}, \mathbf{M}, \mathbf{x}^0, \mathbf{b})$ algorithm

$$\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\,\mathbf{x}^0$$
For $k = 0, \ldots,$ do
$$\mathbf{z}^k = \mathbf{M}\mathbf{r}^k$$
$$\beta_k = \frac{(\mathbf{r}^k, \mathbf{z}^k)}{(\mathbf{r}^{k-1}, \mathbf{z}^{k-1})},\ \beta_0 = 0$$
$$\mathbf{p}^k = \mathbf{z}^k + \beta_k \mathbf{p}^{k-1}$$
$$\mathbf{q}^k = \mathbf{A}\mathbf{p}^k$$
$$\alpha_k = \frac{(\mathbf{r}^k, \mathbf{z}^k)}{(\mathbf{q}^k, \mathbf{p}^k)}$$
$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$$
$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k \mathbf{q}^k$$
End for

One should note that the performance of PCG $(\mathbf{A}, \mathbf{M}, \mathbf{x}^0, \mathbf{b})$ is completely characterized by the dimension n of $\mathbf{A}$ and by the performance of the kernals that compute $\mathbf{z}^k$ (preconditioning) and $\mathbf{g}^k$ (matrix–vector product). Typically, computing $\mathbf{z}^k$ and $\mathbf{q}^k$ are the most costly steps of the algorithm. Also, these two steps, which we call the A-dependent steps, are the only ones that depend explicitly on the linear operator $\mathbf{A}$ (and $\mathbf{M}$ which is derived from $\mathbf{A}$). Thus, the performance of PCG$(\mathbf{A}, \mathbf{M}, \mathbf{x}^0, \mathbf{b})$ depends heavily on the sparsity pattern of $\mathbf{A}$ (and $\mathbf{M}$).

For the examples shown in Section 4, we use two types of preconditioning. The first is Jacobi, where $\mathbf{M} = diag(\mathbf{A})^{-1}$ is the inverse of the diagonal of $\mathbf{A}$. The second is $\mathbf{M} = P(\mathbf{A})$, where P is the tenth degree least-squares polynomial approximation of $\mathbf{A}^{-1}$ as described in Saad.[2]

2. DESCRIPTION OF FAC

FAC is a method for solving partial differential equations where there are one or more local regions that require a high degree of resolution and accuracy. In this Section we give a short description of the notation and the fundamental ideas of FAC. For simplicity we formulate FAC in two dimensions using one spatial refinement region.

Suppose that we are given a partial differential equation whose spatial domain Ω is a rectangle in $\Re^2$ and suppose that Ω contains a proper subregion Ω_F which, when solved numerically, requires a finer spatial resolution than the rest of Ω. (Determination of such a region could be made by having prior knowledge of the solution behaviour or by using some technique for estimating the error – see McKay and Thomas.[3]) Place a coarse grid G on Ω with a spatial mesh size Δx and Δy sufficiently fine to provide a good approximation to the solution on $\Omega - \Omega_F$. Then place a finer grid $\mathcal{G}_F$ with mesh sizes δx and δy, where $\mathcal{G}_F$ is aligned with the coarse grid and $\delta x = \Delta x/m_x$ and $\delta y = \Delta y/m_y$, so that $\mathcal{G}_F$ covers the region Ω_F and is large enough so that the solution of the boundaries of $\mathcal{G}_F$ is approximated sufficiently well by the coarse grid mesh. Then the composite grid $\mathcal{G}$ is defined to be the union of G and $\mathcal{G}_F$, where $\mathcal{G}_F$ covers the subregion Ω_F. Figure 1 shows an example where $\delta x = \delta y = \Delta x/2 = \Delta y/2$.

Assume we are given an approximation to the partial differential equation and boundary conditions on the composite grid $\mathcal{G}$ that can be written as

$$\mathcal{L}u = \mathcal{F} \tag{1}$$

where $\mathcal{L}$ is the resulting linear operator and $\mathcal{F}$ depends on the inhomogeneous terms in the equation and the boundary conditions.

The composite grid can be partitioned so that $\mathcal{G} = \mathcal{G}_c \cup \mathcal{G}_I \cup \mathcal{G}_F$, where $\mathcal{G}_F$ is the grid over the refinement region as above, $\mathcal{G}_I$ consists of the coarse grid points along the boundary of Ω_F, and $\mathcal{G}_c$ the coarse grid points outside Ω_F. The partitioning is illustrated in Figure 1.

The coarse grid, G, can be partitioned similarly as $G = G_C \cup G_I \cup G_F$, where G_C consists of the coarse grid points outside of G_F ($G_C = \mathcal{G}_C$), G_I the coarse grid points on the boundary of Ω_F ($G_I = \mathcal{G}_I$), and G_F the coarse grid points corresponding to interior points of $\mathcal{G}_F$ ($G_F \subset \mathcal{G}_F$). $\mathcal{G}$, $\mathcal{G}_F$ and G are the principal grids used in the FAC algorithm. To be able to pass information between $\mathcal{G}$ and G, assume that a prolongation or interpolation operator, P, and a restriction operator, R, have been defined so that

$$P: \mathbf{G} \to \mathcal{G}, R: \mathcal{G} \to \mathbf{G}$$

Using P and R, an operator, $\mathbf{L}$, on the coarse grid can be defined according to the Galerkin

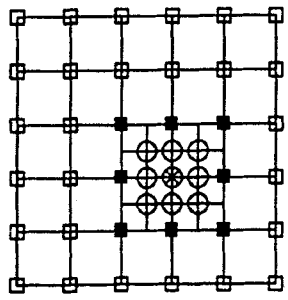

Figure 1. Example of a composite grid: $\mathcal{G} = \square \cup \blacksquare \cup \otimes \cup \bigcirc$, $\mathcal{G}_C = \square$, $\mathcal{G}_I = \blacksquare$, $\mathcal{G}_F = \otimes \cup \bigcirc$, $G = \square \cup \blacksquare \cup \otimes$, $G_C = \square$, $G_I = \blacksquare$ and $G_F = \otimes$

condition

$$\mathbf{L} = R \circ \mathscr{L} \circ P$$

Based on the above partitioning of the grids $\mathscr{G}$ and $\mathbf{G}$, we can partition $u, \mathbf{u}, \mathscr{F}, \mathscr{L}$ and $\mathbf{L}$ as

$$u = (u_C, u_I, u_F),\ \mathbf{u} = (\mathbf{u}_C, \mathbf{u}_I, \mathbf{U}_F),\ \mathscr{F} = (\mathscr{F}_C, \mathscr{F}_I, \mathscr{F}_F)$$

$$\mathscr{L} = \begin{pmatrix} \mathscr{L}_{CC} & \mathscr{L}_{CI} & \Theta \\ \mathscr{L}_{IC} & \mathscr{L}_{II} & \mathscr{L}_{IF} \\ \Theta & \mathscr{L}_{FI} & \mathscr{L}_{FF} \end{pmatrix}$$

and

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{CC} & \mathbf{L}_{CI} & \Theta \\ \mathbf{L}_{IC} & \mathbf{L}_{II} & \mathbf{L}_{IF} \\ \Theta & \mathbf{L}_{FI} & \mathbf{L}_{FF} \end{pmatrix}$$

Note that $\mathscr{L}_{CC} = \mathbf{L}_{CC}, \mathscr{L}_{IC} = \mathbf{L}_{IC}$ and $\mathscr{L}_{CI} = \mathbf{L}_{CI}$. The significant difference between $\mathbf{L}$ and $\mathscr{L}$ is in the IF and FI blocks, where $\mathscr{L}$ is 'reaching' for grid points in $\mathscr{G}_I$ and in $\mathscr{G}_F$, respectively. How this is done is what ultimately defines the composite grid operator $\mathscr{L}$. We use a finite-volume technique as described in Heroux[4] to define these equations.

Using the above notation, a residual correction FAC cycle can be written as follows:

$$\left.\begin{array}{ll} \text{Step 1.} & \Delta\mathbf{u}^{n+1} = \mathbf{L}^{-1} R(\mathscr{F} - \mathscr{L} u^n) \\ \text{Step 2.} & u^{n+1} = u^{n+1} + P\,\Delta\mathbf{u}^{n+1} \\ \text{Step 3.} & u_F^{n+1} = \mathscr{L}_{FF}^{-1}(\mathscr{F}_F - \mathscr{L}_{FI} u_I^{n+1}) \end{array}\right\}$$

In the above FAC algorithm, we compute approximations to the solution of equation (1). McCormick and Thomas[5] and Heroux[4] give conditions under which the above iterative process converges for equation (1). In practice, we have seen that two FAC cycles are usually sufficient with an initial guess of $u^\circ \equiv 0$.

Solving the coarse and fine grid problems, i.e. inverting the operators $\mathbf{L}$ and $\mathscr{L}_{FF}$ in Steps 1 and 3, respectively, typically consumes most of the computation time. However, since $\mathbf{L}$ and $\mathscr{L}_{FF}$ are defined on uniform, rectangular grids, fast solvers can be used providing good machine performance. For the problems presented in this paper, we solve the coarse and fine grid problems using the preconditioned conjugate gradient method defined in Section 1. We first solve the coarse grid problems using PCG($\mathbf{L}$, diag($\mathbf{L}$)$^{-1}$, $u^\circ \equiv 0, R\mathscr{F}$). Then after Step 2, we solve the fine grid problem using PCG($\mathscr{L}_{FF}$, diag($\mathscr{L}_{FF}$)$^{-1}$, Pu_F^{-1}, $(\mathscr{F}_F - \mathscr{L}_{FI} u_I^{n+1})$). Subsequent iterations proceed similarly. However, in some cases, the polynomial preconditioner replaces the Jacobi preconditioner as noted in Section 4 since, in these cases, it performs better.

3. DESCRIPTION OF SITRSOL

The second method we use to solve the composite grid equations is the preconditioned conjugate gradient method applied directly to equation (1). Note that the composite grid operator is not defined on a uniform grid. Thus, the kernels for the A-dependent steps of PCG() must handle more general sparsity patterns.

In order to solve equation (1) directly we use a Cray library routine called SITRSOL (see References 6 and 7), which accepts a matrix in a compressed sparse column format along with a right-hand side and an initial guess. It then solves the linear system using one of a variety

of preconditioned iterative solvers. In particular, via SITRSOL, we use PCG($\mathscr{L}$, diag($\mathscr{L}$)$^{-1}$, $u^\circ \equiv 0$, $\mathscr{F}$) or, in certain cases, PCG($\mathscr{F}$, $P(\mathscr{L})$, $u^\circ \equiv 0$, $\mathscr{F}$).

For general-patterned sparse matrices such as equation (1), the most expensive steps in the PCG() algorithm still include the A-dependent preconditioning and matrix–vector product routines, but there is typically a third step which is also expensive. This step is the analysis and conversion of the user's matrix into form(s) which are more suitable for efficient computation of the A-dependent steps. In our case, that involves converting the user matrix from the commonly used compressed sparse column (CSC) format to a more complicated but more efficient jagged diagonal (JAD) format (see References 6, 8–10). This conversion is done because the matrix–vector product routine is much more efficient for the JAD format than for the CSC format on vector/parallel machines. SITRSOL does this conversion internally.

It is possible to eliminate the conversion from CSC format to JAD format in two ways. The first would be to have the user matrix already in JAD format. However, the JAD format is not a natural sparse matrix format in the sense that an application program which generates a sparse linear system cannot easily generate the matrix in JAD format. Thus, if conversion to JAD format is done, this cost should be included in the cost accounting. The second way would be to perform the A-dependent operations using the CSC format. However, because of the poor performance of the A-dependent operations using the CSC format, this approach is typically very bad if more than a few iterations are performed. In other words, the time spent converting from CSC format to JAD format is typically recovered after a few iterations because of the improved performance of the A-dependent operations. Thus, in Section 4, it is appropriate to include the CSC to JAD conversion time in the results.

4. EXAMPLES

In this Section we discuss results from two example problems. These problems are simple but display some of the characteristics of more difficult problems and allow some analysis to be done without the complexity associated with a more difficult problem. Both problems are generated using vertex-centred finite-volume discretizations of a second-order, self-adjoint, elliptic operator in two space dimensions. The details of the problems are as follows.

4.1. Problem 1 (Poisson problem with local support)

The first problem is

$$\nabla^2 u(x, y) = f(x, y), \ (x, y) \in \Omega = (0, 1) \times (0, 1)$$
$$u(x, y) = g(x, y), \ (x, t) \in \partial\Omega$$

where f and g are chosen so that $u(x, y) = \phi(x)\psi(y)$ with

$$\phi(x) = \begin{cases} C_q \exp\left(\dfrac{-1}{x - x_0} - \dfrac{1}{x_1 - x}\right), & x \in (x_0, x_1) \\ 0, \text{ elsewhere} \end{cases}$$

$$\psi(y) = \begin{cases} C_2 \exp\left(\dfrac{-1}{y - u_0} - \dfrac{1}{y_1 - y}\right), & y \in (y_0, y_1) \\ 0, \text{ elsewhere} \end{cases}$$

where

$$x_0 = y_0 = 0{\cdot}3, x_1 = y_1 = 0{\cdot}7$$

and

$$C_1 = \exp\left(\frac{4}{x_1 - x_0}\right), \; C_2 = \exp\left(\frac{4}{y_1 - y_0}\right)$$

This problem is a minor variation of a problem studied in Ewing *et al.*[11]

To construct a composite grid for this problem, we place a fine grid on the region $(0{\cdot}3, 0{\cdot}7) \times 0{\cdot}3, 0{\cdot}7)$ with a mesh spacing $\delta x = \delta y = \Delta x/2 = \Delta y/2$. We use four different values for $\Delta x = \Delta y$ as shown in Table I. Table I also shows the size or number of unkowns on each grid as well as two measures of the accuracy, ε_r and ε_2, and the 2-norm condition number κ_2. The first measure of accuracy is the relative error

$$\varepsilon_r = \frac{\| \hat{u} - u \|_\infty}{\| u \|_\infty}$$

where $\hat{u}$ is the approximation to the exact solution u on the appropriate grid. This measure indicates the approximate number of correct significant digits of the approximate solution $\hat{u}$ in the sense that if $\varepsilon_r \approx 10^{-p}$ then the largest component of $\hat{u}$ has approximately p correct significant digits. The second measure of accuracy is simply

$$\varepsilon_2 = \| \hat{u} - u \|_2$$

TABLE I. Problem 1 statistics

Problem 1A. $\Delta x = \Delta y = 1/10$				
Grid	Size	ε_r	ε_2	κ_2
Coarse	81	2·036	2·6786	39·3
Fine	49	0·166	0·2220	25·4
Composite	121	0·166	0·2424	95·9
Problem 1B. $\Delta x = \Delta y = 1/20$				
Grid	Size	ε_r	ε_2	κ_2
Coarse	361	0·165	0·2554	155·6
Fine	225	0·395E−01	0·0997	99·8
Composite	537	0·395E−01	0·1002	399·4
Problem 1C. $\Delta x = \Delta y = 1/30$				
Grid	Size	ε_r	ε_2	κ_2
Coarse	841	0·724E−01	0·1379	349·4
Fine	529	0·172E−01	0·6214E−01	223·8
Composite	1249	0·172E−01	0·6534E−01	910·2
Problem 1D. $\Delta x = \Delta y = 1/40$				
Grid	Size	ε_r	ε_2	κ_2
Coarse	1521	0·395E−01	0·1002	620·6
Fine	961	0·960E−02	0·4301E−01	397·4
Composite	2257	0·960E−02	0·4864E−01	1628·0

For this problem, we also computed the eigenvalue distribution for the coarse, fine and composite grid discrete operators for each value of $\Delta x = \Delta y$. In each case we found that the composite grid operator had one particularly small eigenvalue, which is the main reason why κ_2 is much larger for the composite grid operator than for either the coarse or fine grid operators. We are not sure yet if this is an anomaly or not.

In the following computational results, we compare the FAC solution of Problem 1 to the SITRSOL solution. We perform FAC cycles until the FAC approximation of the composite grid solution is within the accuracy of the composite grid problem. Note that this always turns out to be two FAC cycles. Thus, to obtain the FAC solution we must solve four linear systems. We first solve the coarse grid problem via PCG() using an initial guess of zero. We then solve the fine grid problem via PCG() using the initial guess provided by the coarse grid solution. We then repeat the process. Thus, the grid sequence is coarse-fine-coarse-fine or (C, F, C, F). The SITRSOL solution is obtained by applying PCG() directly in the composite grid problem. In all cases PCG() is used to solve on each grid up to the accuracy shown in Table I.

Table II shows the computational results for both methods applied to Problem 1. We show the grid sequence, the number of PCG() iterations on each grid, the solution time in seconds, the composite grid residual, and the FAC convergence factor, ρ, which is the ratio of the residuals between the first and second cycle. In most cases, Jacobi preconditioning was used for PCG(). However, if the number of iterations is preceded by a '*', then polynomial preconditioning was used because it was better (faster) than Jacobi. It is clear from the results that, if we solve to truncation error on the coarse and fine grids, then two FAC cycles are sufficient. Also, at least for smaller sizes, it is clear that if more than two cycles were needed,

TABLE II. Problem 1 results

Problem 1A. $\Delta x = \Delta y = 1/10$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F, C, F)	(13, 9, 12, 8)	1·66E–03	0·6E–06	7·5E–02
SITRSOL	Composite	15	1·77E–03	0·8E–03	

Problem 1B. $\Delta x = \Delta y = 1/20$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F, C, F)	(28, 17, 19, 2)	3·93E–03	0·7E–03	9·2E–02
SITRSOL	Composite	28	6·31E–03	0·9E–03	

Problem 1C. $\Delta x = \Delta y = 1/30$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F, C, F)	(41, 26, 17, 1)	8·39E–03	0·6E–03	1·1E–01
SITRSOL	Composite	43	1·53E–02	0·8E–03	

Problem 1D. $\Delta x = \Delta y = 1/40$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F, C, F)	(*7, *4, 2, 1)	1·44E–02	0·8E–03	7·5E–02
SITRSOL	Composite	60	3·08E–02	0·9E–03	

then FAC would not be competitive with the direct solution technique. Also, it is clear that if FAC were accelerated by some other scheme in the outer iteration, e.g. if FAC were used as a preconditioner for the conjugate gradient method on the composite grid, then solution times would increase with no improvement in the solution accuracy, at least for this simple problem.

4.2. Problem 2 (wall/3 well problem)

The second problem we consider has three wells (a source and two sinks), and an impermeable retaining wall as shown in Figure 2. The wells are modelled by discrete Dirac delta functions. The source well has strength 1·0 and the sinks have strength 0·5. The retaining wall is modelled in the finite volume discretization by now allowing any flow across the wall in the x-direction. We use zero Dirichlet boundary conditions. Thus, except for the modified stencil at the well, this is a Poisson problem. Four patches are placed, one on each well and one on the wall as shown in Figure 2. All patches use a mesh spacing $\delta x = \delta y = \Delta x/2 = \Delta y/2$. We use two coarse grid mesh spacings, namely $\Delta x = \Delta y = 1/80, 1/100$. Table III shows results for the two solution techniques. Note that now one FAC cycle includes one coarse grid solve and four fine grid solves, one on each fine grid, giving a grid sequence of (C, F1, F2, F3, F4). The results in Table III again show that, even though many more iterations are performed for the FAC solution, the regularity of the grids and the smaller problem sizes allow the FAC solution to be computed more cheaply than the direct solution.

To see in more detail why this is, consider the performance statistics in Tables IV and V. Note that the routine SBXPCG performs all but the A-dependent operations of the PCG()

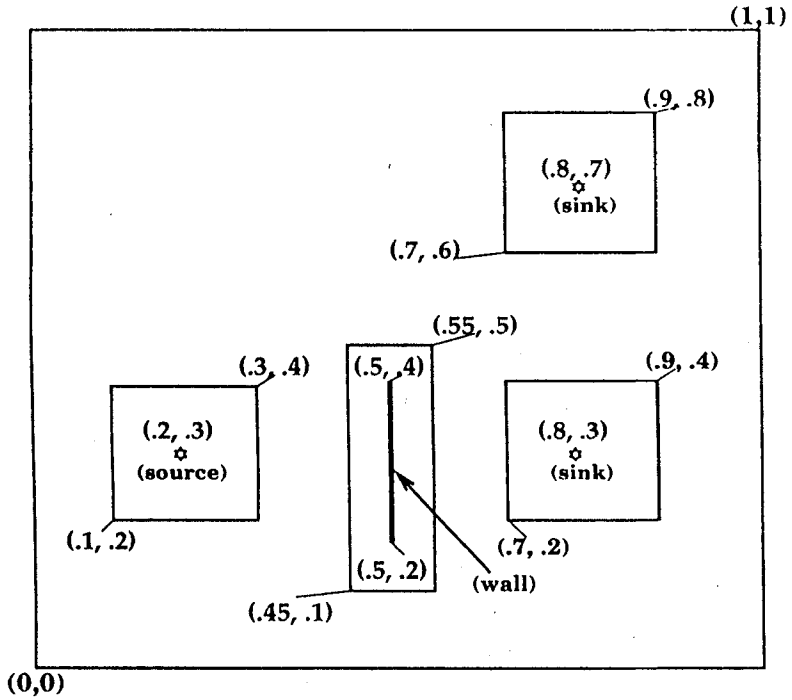

Figure 2. Domain and refinement regions for Problem 2

TABLE III Problem 2 results

Problem 2A. $\Delta x = \Delta y = 1/80$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F1, F2, F3, F4, C, F1, F2, F3, F4)	(*22, *6, *6, *6, *9, *14, 4, 4, 4, 38	1·63E–01	0·3E–03	6·0E–02
SITRSOL	Composite	*18	2·08E–02	0·2E–03	

Problem 2B. $\Delta x = \Delta y = 1/100$					
Method	Grid sequence	No. of Iters.	Time	Residual	ρ
FAC	(C, F1, F2, F3, F4 C, F1, F2, F3, F4)	(*27, *7, *7, *7, *11, 17, 3, 2, 2, *4)	3·00E–01	0·2E–02	6·0E–02
SITRSOL	Composite	*23	4·04E–01	0·2E–02	

TABLE IV Direct composite grid solution timing results

Problem 2B. Composite solution				
Routine	Time	Ex.%	Mmems	Mflops
PRECON	2·81E–01	69·6	251·2	137·4
CSC2JAD	6·09E–02	15·1	18·6	0·0
MATVEC	2·44E–02	6·0	242·5	128·4
SBXPCG	1·71E–02	4·2	281·4	238·6
OTHERS	2·06E–02	5·1	6·8	0·0
Totals	4·04E–01	100·0	206·0	113·8

TABLE V FAC composite grid solution timing results

Problem 2B. FAC Solution				
Routine	Time	Ex.%	Mmems	Mflops
MATVEC	2·24E–01	74·7	280·7	229·8
PRECON	4·66E–02	15·5	367·5	232·4
SBXPCG	2·59E–02	8·6	286·3	240·8
OTHERS	3·50E–03	1·2	120·4	101·4
Totals	3·00E–01	100·0	292·5	229·0

algorithm. Also, it should be noted that in Table V most of the time spent in MATVEC was due to PRECON, since PRECON called MATVEC to apply the polynomial preconditioner. The results from Table IV show that we pay a penalty for the general pattern for the composite grid matrix in two ways. Firstly, we pay a penalty of 15 per cent in the overhead for data-structure conversion (CSC2JAD). However, if we left the matrix in the original compressed sparse column format the overall solution time would more than quadruple. Secondly, we pay a penalty in the performance of the matrix-vector (MATVEC) and preconditioning

(PRECON), which go at a combined rate of approximately 135 MFLOPS. This is a fairly good rate, but it is no match for the 230 MFLOPS performance of the corresponding kernels used in the FAC solution.

5. CONCLUSIONS

In this paper we have compared solving composite grid problems via FAC and direct application of the preconditioned conjugate gradient method to the composite grid problem. We have shown that FAC has several advantages which allow it to overcome the fact that it must solve many more linear systems. First of all, the operators inverted by FAC are better conditioned. However, for these simple problems this fact did not significantly reduce the total number of operations. Secondly, and most importantly, FAC uses regular grids which require no conversion to a special data structure and which allow efficient computation on vector/parallel machines. It is this fact which provides the real advantage of FAC for these problems.

We have also shown that, for the simple problems considered in this paper, two FAC cycles are sufficient if the coarse and fine grid problems are solved to truncation error. We have also shown that any attempt to accelerate FAC (for these problems) would most likely introduce unnecessary overhead. However, we believe there are cases for which acceleration of FAC by some conjugate gradient type scheme is advantageous and necessary. This is still a topic for further research.

ACKNOWLEDGEMENT

This research was supported in part by the National Science Foundation under grant DMS-8813406.

REFERENCES

1. Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, ed., 1989.
2. Youcef Sadd, 'Practical use of polynomial preconditionings for the conjugate gradient method', *SIAM J. Sci. Stat. Comput.*, **6**(4), 865–881 (1985).
3. S. McKay and J. W. Thomas, 'Application of the self adaptive time dependent fast adaptive composite grid method', in Jan Mandel, Stephen F. McCormick, Jr., J. E. Dendy, Charbel Farhat, Guy Lonsdale, Seymour V. Parter, John W. Ruge and Klaus Stüben (Eds, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, Philadelphia, 1989, SIAM, pp. 338–347.
4. Michael A. Heroux, *The Fast Adaptive Composite Grid Method for Time Dependent Problems*, PhD thesis, Colorado State University, 1989.
5. S. McCormick and J. W. Thomas, 'The fast adaptive composite grid (FAC) method for elliptic equations', *Math. Comput.*, **46**, 439–456 (1986).
6. Michael A. Heroux, Phuong Vu and Chao Wu Yang, 'A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP' *Appl. Numer. Math.*, (1991).
7. John Madsen, *UNICOS Math and Scientific Library Reference Manual*, Volume 3, Cray Research Inc., version 6.0 edition, 1990.
8. Jocelyne Erhel and Bernard Philippe, 'Multiplication of a vector by a sparse matrix on supercomputers', Technical report, INRIA/IRISA, 1988.
9. Gaia Valeria Paolini and Giuseppe Radicati Di Brozolo, 'Data structures to vectorize CG algorithms for general sparsity patterns', *BIT*, **29**, 703–718 (1989).

10. Edward Charles Anderson, 'Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations', Master's thesis, Center for Supercomputing Research and Development, University of Illinois, August 1988.
11. R. E. Ewing, R. D. Lazarov and P. S. Vassilevski, 'Local refinement techniques for elliptic problems on cell-centered grids', Technical report, Enhanced Oil Recovery Institute, University of Wyoming, 1988.