SEMICOARSENING MULTIGRID ON DISTRIBUTED MEMORY MACHINES

PETER N. BROWN , ROBERT D. FALGOUT AND JIM E. JONES*

Abstract. This paper presents the results of a scalability study for a three dimensional semicoarsening multigrid solver on a distributed memory computer. In particular, we are interested in the scalability of the solver; how the solution time varies as both problem size and number of processors are increased. For an iterative linear solver, scalability involves both algorithmic issues and implementation issues. We examine the scalability of the solver theoretically by constructing a simple parallel model and experimentally by results obtained on a IBM SP. The results are compared with those obtained for other solvers on the same computer.

1. Introduction. This paper focuses on our work in developing a parallel solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation,

$$-\nabla \cdot (D\nabla u) + \sigma u = f, \tag{1}$$

on logically rectangular grids. In R^n, the diffusion coefficient D is a symmetric positive definite $n \times n$ matrix and $\sigma \geq 0$. We restrict our attention to discretizations that produce "nearest neighbor" stencils. In 2D, for example, the linear system can be represented by a 9-point stencil at each grid point of the form

$$\begin{pmatrix} A_{nw} & A_n & A_{ne} \\ A_w & A_c & A_e \\ A_{sw} & A_s & A_{se} \end{pmatrix}.$$

In 3D, the stencil is 27-point. Applications where such a solver is needed include radiation diffusion and flow in porous media. In these applications, the coefficients in the problem are often anisotropic and discontinuous. Anisotropy may be present in the PDE, i.e.

$$D = \begin{bmatrix} D_{11} & 0 \\ 0 & D_{22} \end{bmatrix}, D_{11} \ll D_{22},$$

or may be caused by grids with unequal mesh sizes in the different coordinate directions. The nature of the anisotropy can change within the domain, in some regions we may have $D_{11} \ll D_{22}$ and in others $D_{22} \ll D_{11}$. The discontinuities in the coefficients are typically due to material interfaces where the diffusion coefficient can jump several orders of magnitude.

As we are interested in solving very large systems on massively parallel computers, it is important that our solver be scalable, or very nearly so. Scalability can be defined in various ways. In this paper we are concerned with how the solver performs as both the size of the problem and the number of processors are increased. Let $T(N,P)$ be the time to solve a linear system with N unknowns on a computer using P processors. The *scaled efficiency* of a solver is defined as

$$E(N,P) \equiv T(N,1)/T(PN,P). \tag{2}$$

* Center for Applied Scientific Computing, L-561, Lawrence Livermore National Laboratory, Livermore, California 94550.

One would like $E(N,P) = 1$. This would mean one could double the size of the problem and the number of processors while keeping the solution time constant. For our definition of scalability, we will require only that the scaled efficiency be bounded away from zero, i.e. a solver is *scalable* iff

$$\exists E_N > 0 \text{ such that } E(N,P) \geq E_N \text{ for all } P. \tag{3}$$

In iterative methods for solving linear systems, solver scalability can be divided into two aspects. The first is *algorithmic scalability* which requires that the computational work per iteration is a linear function of problem size and that the convergence factor per iteration is bounded below 1 with bound independent of problem size. The second aspect is *implementation scalability* which requires that a single iteration is scalable on the parallel computer. Both algorithmic and implementation scalability are required for the iterative solver to be scalable.

For our problem, standard iterative methods like Jacobi, Gauss-Seidel or SOR are not algorithmically scalable. The convergence factors of such methods approach 1 as problem size is increased. Multigrid methods can be algorithmically scalable, but in our applications, special attention must be paid to the potential anisotropic and discontinuous coefficients. Our multigrid algorithm is based on the work of Steve Schaffer [11]. This algorithm's parallel performance has been investigated by other authors on various parallel machines: Bandy, Dendy and Spangenberg [3] (2D and 3D results on the CM-5); Dendy, Ida and Rutledge[5] (2D results on the CM-2); Smith and Weiser [12](2D results on the Intel iPSC/2 hypercube). Our work differs from these previous studies in its emphasis on 3D problems and the scalability of the method on distributed memory machines. Alternative multigrid methods which are robust for anisotropic and discontinuous coefficients include algebraic multigrid [10], black box multigrid [4], and multiple semicoarsened multigrid [9].

The rest of this paper is organized as follows. In section 2 we describe the semicoarsening multigrid algorithm. In section 3 we describe our parallel implementation and develop a model for predicting its performance. In section 4 we present numerical results investigating the solver's algorithmic and implementation scalability. We compare the implementation scalability of our solver to that of ParFlow multigrid [1] and a simple matrix vector product. In Section 5, based on the model and numerical experiments, we present some conclusions about the scalability of the solver.

2. SMG: semicoarsening multigrid. The semi-coarsening algorithm used is based on the work by Schaffer [11] and we will briefly discuss this particular multigrid algorithm. To simplify the discussion, we focus on the 2D algorithm (commenting on the 3D extension) and on those features that differentiate it from standard multigrid methods. Let $AU = F$ be the given linear system to solve, where the unknown U and right-hand side F are vectors defined on a logically rectangular grid. We will use an h superscript to denote quantities defined on the given grid. The matrix A is symmetric, positive definite and connections have the standard "nearest-neighbor" 9-point stencil form. The multigrid algorithm of Schaffer uses a combination of semicoarsening, line-relaxation, and operator-based interpolation. The resulting algorithm is efficient and robust with respect to anisotropic and widely variable coefficients in the matrix A.

As the grid is logically rectangular, there is a unique index (i,j) for each point on the grid, and the grid can be given a "red/black" line coloring. All unknowns $\{(i,j), j \text{ odd}\}$ are considered "red" and will be used for the coarse grid. We will use a $2h$ superscript to denote quantities defined on the coarse grid. This is called semi-coarsening (as opposed to full or standard coarsening) as the coarse grid is only coarser in one of the dimensions. Red/black line relaxation involves updating the solution at all red lines to satisfy their equations (a tridiagonal solve for each red line) followed by a similar update for the black lines. Because of the 9-point stencil, there is no dependence between lines of the same color and they can be updated in parallel.

An important, unique feature of the SMG algorithm is the definition of the interpolation operator I_{2h}^{h} used to transfer an error correction from the coarse to the fine grid. The definition is motivated by the relationship between error on red and black lines after a black line relaxation sweep. To briefly describe the approach, let

$$A_{J,J-1}U_{J-1} + A_{J,J}U_J + A_{J,J+1}U_{J+1} = F_J \tag{4}$$

be the equations for the J^{th} line. Here $U_J = (U_{i,J}, i = 1, \ldots, n_x)$ and similarly for $U_{J\pm 1}$. After relaxing this line, the error equation is

$$A_{J,J-1}e_{J-1} + A_{J,J}e_J + A_{J,J+1}e_{J+1} = 0, \tag{5}$$

so

$$e_J = -A_{J,J}^{-1}A_{J,J-1}e_{J-1} - A_{J,J}^{-1}A_{J,J+1}e_{J+1}. \tag{6}$$

After black line relaxation this relationship describes how the error at black lines is related to the error at red (coarse) lines; it gives the "ideal" interpolation formula. However, using equation (6) leads to non sparse interpolation operators. In the SMG algorithm, sparse approximations to these ideal interpolation operators are used. The matrices $-A_{J,J}^{-1}A_{J,J-1}$ and $-A_{J,J}^{-1}A_{J,J+1}$ are approximated by diagonal matrices with the same action on constant vectors. The computation of these interpolation operators involves two tridiagonal solves for each black grid line.

With this definition for the interpolation operator I_{2h}^{h}, its transpose is used for the restriction operator I_{h}^{2h} (used in transferring residuals from the fine to the coarse grid), and the coarse grid versions of A are defined by the Galerkin condition, i.e. $A^{2h} = I_h^{2h}A^hI_{2h}^h$. These components are computed in a setup phase and are then used in a standard multigrid V-cycle as outlined below.

$V(\nu_1, \nu_2)$-cycle

1. Pre-relaxation on $A^hU^h = F^h$. Perform ν_1 sweeps of red/black line relaxation.
2. Set $F^{2h} = I_h^{2h}(F^h - A^hU^h)$.
3. "Solve" $A^{2h}U^{2h} = F^{2h}$ by recursion.
4. Correct $U^h \leftarrow U^h + I_{2h}^hU^{2h}$.
5. Post-relaxation on $A^hU^h = F^h$. Perform ν_2 sweeps of black/red line relaxation.

The equation to be solved in step 3 has the same form as the original grid h problem. It is solved by applying the same algorithm using a still coarser grid $4h$. Eventually, a coarse grid is reached that has a single grid line and line relaxation is a direct solver.

The 3D algorithm is analogous to the 2D one presented above. Essentially "lines" are replaced by "planes". The coarsening is done in only the z-direction, relaxation is red/black plane relaxation, and the interpolation operator has the same action as the "ideal" interpolation on functions constant in a plane. In the 3D algorithm all plane solves are done approximately by one V-cycle of the 2D algorithm. Note also that cyclic reduction, the method we use for the tridiagonal line solves, can be viewed as a 1D version of this algorithm. Essentially "lines" are replaced by "points". Here there is no problem using the "ideal" interpolation operators, and a single $V(1,0)$-cycle (or $V(0,1)$-cycle) with relaxation only performed at the black points is a direct method.

3. Parallel SMG. In implementing the SMG algorithm on a distributed memory computer, the simplest approach is domain partitioning. The fine grid Ω^h is distributed among the available processors so that each processor q has a subgrid of operation Ω_q^h where $\Omega^h = \cup \Omega_q^h$ and $r \neq q \Rightarrow \Omega_r^h \cap \Omega_q^h = \emptyset$. The distribution of the fine grid naturally induces distributions on coarser grids, a coarse grid point "belongs" to processor q iff the corresponding fine grid point does. In the parallel algorithm, each processor q executes those operations that result in changes at points on its subgrid. Note that on coarser grids, some processors may go idle as they have no points left in their subgrid. In this domain partitioning approach, there are steps in the algorithm where data must be communicated. As a simple example, to calculate the residual of the 3D problem at one of the points on the boundary of Ω_q^h, processor q must receive current values of the approximate solution U from its neighboring processors. A more complicated example is in the relaxation of the 3D problem. The relaxation is red/black plane relaxation and we have organized our code so that the planes of the same color are solved simultaneously. In say the red plane solve, when communication is required between processors q and r, a single message is sent that contains data on all red planes shared by the two processors rather than a single message for each such plane. Similarly, a single message sent between two processors in line relaxation contains data on all lines being relaxed that are shared between them. In our code, we have used this domain partitioning approach and used MPI [6] to handle the communication.

There are reasons for using the domain partitioning approach to parallelizing a multigrid algorithm other than its simplicity. Application codes often use this approach and using it within the multigrid solver as well eliminates the need for data redistribution. The domain partitioning approach can also yield very efficient parallel multigrid solvers, for example, see [1]. Novel, parallel multigrid algorithms have been proposed in the literature to break the sequential processing of grid levels; to reduce communications within a standard V-cycle; and to create additional, useful work for processors that would otherwise go idle on coarser grids. See [7] for a brief survey. In our implementation, we have not made any algorithmic changes. The parallel and serial codes produce identical numerical results.

In investigating the parallel performance of the SMG code, it is useful to construct a simple model to see what performance should be expected. In the model we will consider a 3D problem distributed among p^3 processors so that each processor's subgrid has size N^3 and the total problem size, denoted by $\bar{N}$, is $(pN)^3$. In the model we assume a assume a $V(1,0)$-cycle and consider only the relaxation process which is the dominant operation in the algorithm. In the model we assume that the time to access n doubles from non-local memory is

$$\alpha + \beta n,$$

and the time to perform a floating-point operation is f.

The model is somewhat complicated because of the recursive nature of the relaxation process: relaxing the 3D problem involves plane solves using the 2D algorithm which it turn uses the 1D algorithm (cyclic reduction). We begin by looking at the communication and computation involved in the 1D line solves, we assume the lines are in the x-coordinate direction. Consider performing line relaxation where each processor has a subgrid of size $N \times \sigma_y N \times \sigma_z N, 0 < \sigma_y, \sigma_z \leq 1$. At each level of the cyclic reduction algorithm, a processor must relax its black points and pass solution values to neighboring processors on each of two boundaries. The time can roughly be modeled as

$$T^{1D}_{l_x}(\sigma_y, \sigma_z) = 2\alpha + 2\sigma_y\sigma_z N^2\beta + \sigma_y\sigma_z N^2 2^{-l_x} N A^{1D}_{l_x} f, \tag{7}$$

where l_x is the level in the cyclic reduction algorithm ($l_x = 0$ is finest) and $A^{1D}_{l_x}$ is the stencil size. The first two terms are due to communication, and the last term to computation. Summing over the number of 1D levels, $L_x = \log_2(pN)$, yields

$$T^{1D}(\sigma_y, \sigma_z) = 2L_x\alpha + 2L_x\sigma_y\sigma_z N^2\beta + 6\sigma_y\sigma_z N^3 f, \tag{8}$$

where we have taken $A^{1D}_{l_x} = 3$ as the stencils within a line are 3-point. Now consider the 2D plane solves which we assume are xy-planes. Consider performing plane solves where each processor has a subgrid of size $N \times N \times \sigma_z N, 0 < \sigma_z \leq 1$. In the relaxation part of the 2D solve, a processor must calculate the right hand side for the line solves, perform the line solves, and pass solution values to neighboring processors on each of two boundaries. It must perform these operations in both the red and black line relaxation steps. The time for 2D grid level l_y can roughly be modeled as

$$T^{2D}_{l_y}(\sigma_z) = 4\alpha + 4\sigma_z N^2\beta + \sigma_z N^2 2^{-l_y} N(A^{2D}_{l_y} - A^{1D}_{l_y})f + 2T^{1D}(2^{-l_y-1}, \sigma_z). \tag{9}$$

where $A^{2D}_{l_y}$ is the 2D stencil size. The first two terms are due to communication, and the last two to computation. Summing over the number of 2D levels yields

$$T^{2D}(\sigma_z) = 4L_y(L_x+1)\alpha + 4\sigma_z N^2(L_x + L_y)\beta + 20\sigma_z N^3 f, \tag{10}$$

where we have taken $A^{2D}_{l_y} - A^{1D}_{l_y} = 2$ on the finest 2D level (the 2D plane stencils are 5-point, the 1D stencils are 3-point), and $A^{2D}_{l_y} - A^{1D}_{l_y} = 6$ on coarser 2D levels (the 2D plane stencils are 9-point, the 1D stencils are 3-point). Now consider the full 3D solves. Similar to the 2D case, the time for plane relaxation on 3D grid level l_z can roughly be modeled as

$$T^{3D}_{l_z} = 4\alpha + 4N^2\beta + N^2 2^{-l_z} N(A^{3D}_{l_z} - A^{2D}_{l_z})f + 2T^{2D}(2^{-l_z-1}). \tag{11}$$

Summing over the number of 3D levels yields

$$T^{3D} = 4L_z(1 + 2L_y(L_x+1))\alpha + 4N^2(L_z + 2L_x + 2L_y)\beta + 48N^3 f. \tag{12}$$

where we have taken $A^{3D}_{l_z} - A^{2D}_{l_z} = 2$ on the finest 3D level (the 3D stencils are 7-point, the 2D stencils are 5-point), and $A^{3D}_{l_z} - A^{2D}_{l_z} = 10$ on coarser 3D levels (the 3D stencils are 15-point, the 2D stencils are 5-point).

Using equation (12), one can predict the scaled efficiency of the algorithm. Figure 1 shows the predicted scaled efficiency for various values of N. In these predictions we used the following values for the model parameters

$$\alpha = 230\mu\text{sec}, \beta = .16\mu\text{sec/double}, f = .074\mu\text{sec/flop}.$$

These parameters are meant to model the initial delivery IBM SP ASCI Blue machine on which the numerical results in the next section were obtained. The values for α

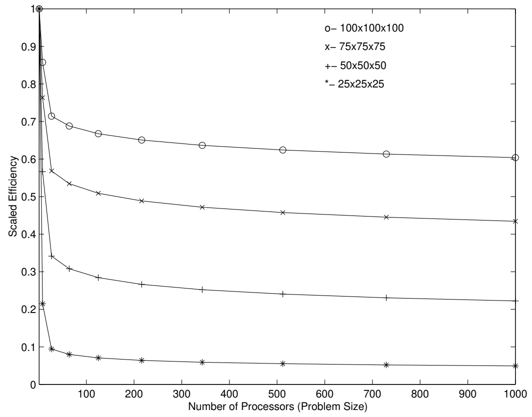

FIG. 1. *Scaled efficiency as predicted by model: equation (12)*

and β were obtained from data in [8]. The value for f reflects the measured 13.5 Mflop rate of the relaxation routine obtained on a single processor for a problem on a $50 \times 50 \times 50$ grid. The model suggests that the scaled efficiency of the SMG solver depends strongly on the problem size per processor. Like most solvers, efficiency is higher when problems are larger per node. The model suggests also that the SMG solver is not scalable in the sense defined in equation (3). The number of communication events for the solver depends on the number of multigrid levels which in turn depends logarithmically on the total problem size. As the total problem size and number of processors grows, the first two terms in equation (12) cause the model's scaled efficiency to fall off. As seen in the plots, the predicted drop off in scaled efficiency is rapid for small numbers of processors after which the scaled efficiency almost levels out, although it does continue to degrade at an exceedingly slow rate. In the limit as total problem size goes to infinity, the predicted scaled efficiency behaves like $\log^{-3}(\bar{N})$. A solver that performed as this model would be scalable for all practical purposes; $E(N, P)$ would be well bounded away from zero for all P one would encounter in practice.

4. Results. As pointed out in the introduction, a scalable iterative solver requires both algorithmic and implementation scalability. The focus of this section is on implementation scalability, but we include one test of algorithmic scalability. Consider the constant coefficient, potentially anisotropic PDE

$$au_{xx} + bu_{yy} + cu_{zz} = 0 \text{ in } \Omega = (0,1)^3, \tag{13}$$

$$u = 0 \text{ on } \partial\Omega, \tag{14}$$

discretized using standard finite differences on a uniform mesh, yielding a 7-point stencil at each grid point. Taking $a = 0.1, b = 1, c = 10$ and a random initial guess, we report in table 1 the number of SMG V(1,0) cycles required to reduce the l_2 norm of the residual ten orders of magnitude. The convergence histories for the various grid sizes are plotted in figure 2 with the lower line corresponding to the $40 \times 40 \times 40$ grid and the upper to the $240 \times 240 \times 240$ grid. The number of cycles needed and the asymptotic convergence factors are essentially independent of problem size. Each $V(1,0)$-cycle reduces the norm of the residual by roughly an order of magnitude even for the largest problem which has over 13 million unknowns. The algorithmic scalability of the SMG

Problem Size	40^3	80^3	120^3	160^3	200^3	240^3
cycles	8	8	8	8	8	9

TABLE 1
SMG V-cycles for constant coefficient anisotropic problem.

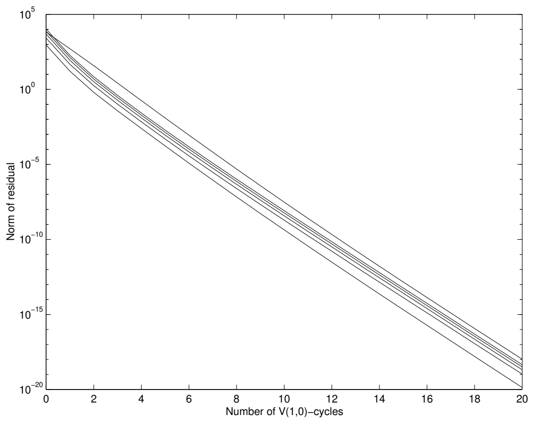

FIG. 2. *Convergence histories for constant coefficient anisotropic problem.*

algorithm for harder problems involving anisotropic and discontinuous coefficients has been demonstrated experimentally in the literature, for example, see [2]. In the rest of this paper, we assume algorithmic scalability for SMG and investigate its implementation scalability.

We investigate the implementation scalability of our SMG code on the initial delivery IBM SP ASCI Blue machine at Lawrence Livermore National Laboratory. For this study, a constant coefficient diffusion problem was discretized using finite differences yielding a 7-point stencil (except for problem 2, where a 27-point stencil was used). The problems are defined by parameters for the size of each processors subgrid ($n_x \times n_y \times n_z$) and parameters for the processor topology ($p_x \times p_y \times p_z$). For example, in the first problem each processors subgrid is a cube ($20 \times 20 \times 20$, for example) and p_x is the number of such subgrids in the x-direction, similarly for p_y and p_z. Thus the total problem size is $(p_x n_x) \times (p_y n_y) \times (p_z n_z)$. In all tests, 5 V(1,0) SMG cycles were performed. As timings on the machine tend to vary, the reported results are the median of several such 5 cycle runs.

Problem 1 Three dimensional partitions with 7-point fine grid operator.
In this problem the subgrids are cubes ($n_x = n_y = n_z = 20, 30, 40, 50$) and the processor topologies are 3 dimensional ($p_x = p_y = p_z = 1, 2, 3, 4$). As a result, processor boundaries cut each of the coordinate directions and message passing is required in the 2D plane solves and the 1D line solves. These results thus correspond to the model developed in the previous section. The timings and scaled efficiencies are reported in table 2 and the scaled efficiencies are plotted in figure 3. We see qualitative agreement (and quantitative agreement to some degree) between the results and the prediction of the model.

Problem 2 Three dimensional partitions with 27-point fine grid operator.
This problem differs from problem 1 only in that the fine grid operator has a 27-point stencil. Thus there is slightly more computations involved and this results in slightly

Processor Topology	Subgrid Size	Time (sec.)	Scaled Efficiency
$1 \times 1 \times 1$	$20 \times 20 \times 20$	0.741	1.000
$2 \times 2 \times 2$	$20 \times 20 \times 20$	2.881	0.257
$3 \times 3 \times 3$	$20 \times 20 \times 20$	3.484	0.213
$4 \times 4 \times 4$	$20 \times 20 \times 20$	5.762	0.129
$1 \times 1 \times 1$	$30 \times 30 \times 30$	1.580	1.000
$2 \times 2 \times 2$	$30 \times 30 \times 30$	4.271	0.370
$3 \times 3 \times 3$	$30 \times 30 \times 30$	6.476	0.244
$4 \times 4 \times 4$	$30 \times 30 \times 30$	8.085	0.195
$1 \times 1 \times 1$	$40 \times 40 \times 40$	3.583	1.000
$2 \times 2 \times 2$	$40 \times 40 \times 40$	7.928	0.452
$3 \times 3 \times 3$	$40 \times 40 \times 40$	9.134	0.392
$4 \times 4 \times 4$	$40 \times 40 \times 40$	12.756	0.281
$1 \times 1 \times 1$	$50 \times 50 \times 50$	6.620	1.000
$2 \times 2 \times 2$	$50 \times 50 \times 50$	12.209	0.542
$3 \times 3 \times 3$	$50 \times 50 \times 50$	16.187	0.409
$4 \times 4 \times 4$	$50 \times 50 \times 50$	18.781	0.352

TABLE 2

Problem 1 results, times are for 5 V(1,0) cycles.

higher scaled efficiencies. However, qualitatively there is little difference between these results and those for the 7-point operator. Compare the two plots in figure 3. Because we are primarily concerned with how the SMG implementation scales, and this is most clearly seen in the plots, we have here omitted the table of timing results. We omit them in all subsequent problems as well.

Problem 3 Two dimensional partitions with 7-point fine grid operator.
In this problem the subgrids are again cubes ($n_x = n_y = n_z = 20, 30, 40, 50$) but the processor topologies are 2 dimensional ($p_x = 1, p_y = p_z = 1, 2, 4, 6, 8$). As a result, the processor boundaries do not cut the x-coordinate direction. Message passing is still required in the 2D plane solves, but the 1D line solves are serial. The scaled efficiencies are plotted in figure 4. Because there is less communication required, the scaled efficiencies are noticeably better than those in Problem 1.

Problem 4 One dimensional partitions with 7-point fine grid operator.
In this problem the subgrids are again cubes ($n_x = n_y = n_z = 20, 30, 40, 50$) but the processor topologies are 1 dimensional ($p_x = p_y = 1, p_z = 1, 2, 4, 8, 16, 32, 64$). As a result, the processor boundaries only cut z-coordinate direction. Message passing is not required in either the 2D plane solves or the 1D line solves. Both are serial. The scaled efficiencies are plotted in figure 4. Again, because there is less communication required, the scaled efficiencies are noticeably better than those in Problem 3 and very much better than those in Problem 1.

Problem 5 Comparison to other solvers on three dimensional partitions.
In this problem, we compare the efficiencies obtained by SMG to those obtained in [8] for a matrix vector product (MatVec) and the ParFlow linear solver. The ParFlow solver [1] uses a multigrid preconditioned conjugate gradient method. The multigrid preconditioner uses semicoarsening but point Jacobi relaxation as it was designed for applications where the direction of anisotropy was constant throughout the domain. It is generally not as robust as the SMG solver. For this problem the subgrids are

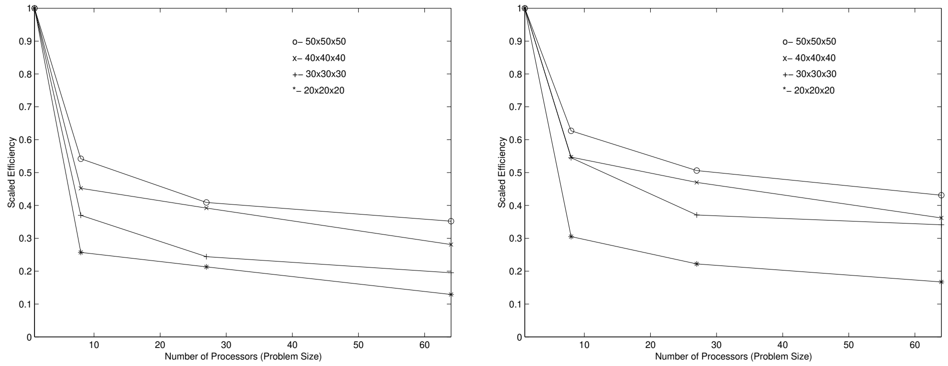

Fig. 3. *Scaled efficiency vs. problem size for three dimensional partitions. Problem 1: 7-point fine grid operator (left) and Problem 2: 27-point fine grid operator (right)*

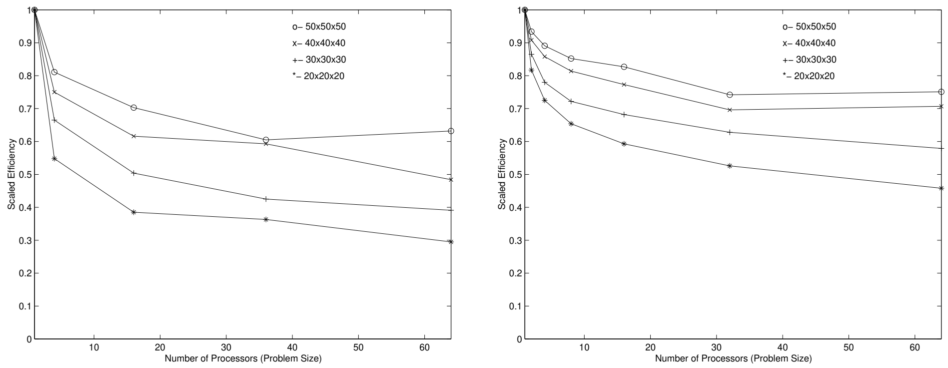

Fig. 4. *Scaled efficiency vs. problem size for 7-point fine grid operator. Problem 3: two dimensional partitions (left) and Problem 4: one dimensional partitions. (right)*

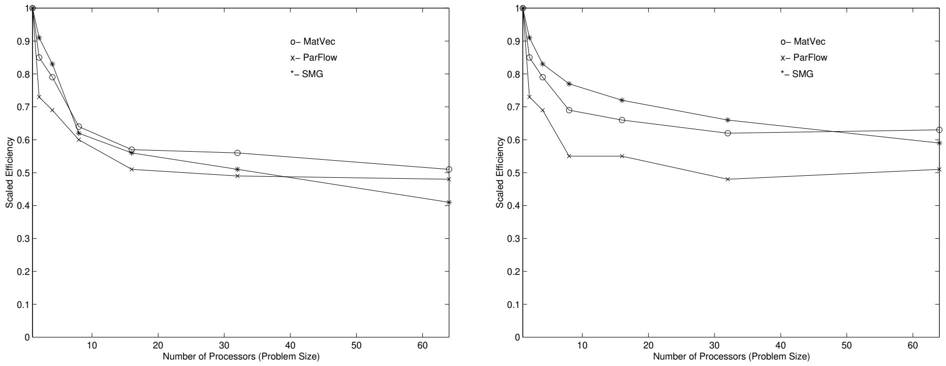

FIG. 5. *Scaled efficiency vs. problem size for 7-point fine grid operator. Problem 5: three dimensional partitions (left) and Problem 6: two dimensional partitions (right).*

rectangular ($n_x = n_y = 64, n_z = 32$) and the processor topologies are 3 dimensional ($1 \times 1 \times 1,\ 1 \times 1 \times 2,\ 1 \times 2 \times 2,\ 2 \times 2 \times 2,\ 2 \times 2 \times 4,\ 2 \times 4 \times 4,\ 4 \times 4 \times 4$). The results are plotted in figure 5. For 64 processors scaled efficiency of SMG (0.41) is sightly worse than that for ParFlow (0.48), and this can be partially explained by the larger communication requirements of SMG due to the plane and line solves in relaxation. We should emphasize that comparing scaled efficiencies of these different methods certainly does not tell you everything you want to know. For example, ParFlow requires fewer computations per V cycle than SMG and achieves a higher Mflop rate on a single processor. Thus if the two methods have comparable convergence rates for a given problem, ParFlow will solve the problem faster. Our point is that SMG is a more robust solver and its implementation scalability is not greatly different than ParFlow's or the scalability of a simple MatVec (0.51 on 64 processors).

Problem 6 Comparison to other solvers on two dimensional partitions.
This problem differs from problem 5 only in the processor topologies. Here the processor topologies are 2 dimensional ($1 \times 1 \times 1,\ 1 \times 1 \times 2,\ 1 \times 2 \times 2,\ 1 \times 2 \times 4,\ 1 \times 4 \times 4,\ 1 \times 4 \times 8,\ 1 \times 8 \times 8$). The results are plotted in figure 5. SMG scaled efficiencies are higher than those in Problem 5 partially because the 1D line solves are serial. However, this is likely not a complete explanation as the MatVec efficiencies are also higher in this case.

Note that in our timings we have not included the time taken in the setup phase of the algorithm. The setup phase can be divided into two parts. The first part is *geometry dependent*: coarse grid sizes are determined, memory is allocated, communication patterns are precomputed. If a subsequent linear solve uses the same size grid and the operator has the same stencil pattern, this part need not be repeated. The second part is *coefficient dependent*: interpolation and coarse grid operators are computed. If a subsequent linear solve uses the same fine grid operator, this part need not be repeated. The work done in the coefficient dependent part of the setup phase is comparable to a single V-cycle. The geometry dependent part of the setup phase currently can take the time of several V-cycles, and we are working on optimizing this. However, many applications we are interested in are time dependent. In implicit time stepping codes where SMG is used to solve the arising linear system at each time step, this geometry dependent part can often be done only once. Thus it constitutes a small portion of the total run time.

5. Conclusions. Our real goal for the SMG code is to produce a robust, fast solver for very large problems. In this study, we have focused on the scalability of the solver as this seems to be very nearly a requirement for making effective use of machines with thousands of processors. The results in this study, both the model in Section 3 and the numerical experiments in Section 4, lead to several conclusions about the SMG algorithm's performance on distributed memory machines. One is not very surprising; the algorithm scales better when the problems are larger per processor. The numerical results also suggest that the algorithm scales better when the data is partitioned so that the 1D line solves are serial and better still when the data is partitioned so that the 2D plane solves are serial as well. The conclusion we would like to make is that the SMG solver is scalable in the sense defined in the introduction: scaled efficiencies are bounded away from zero with bound independent of the number of processors. The model of Section 3 suggests that it might nearly be so; the numerical results are somewhat less conclusive. The plots in Section 4 suggest that the scaled efficiencies may approach zero logarithmically as the model predicts or even be bounded away from zero. Certainly the results using the largest problem size per processor and partitions where the 2D plane solves are serial suggest that the solver is scalable in this case.

REFERENCES

[1] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, September 1996. Also available as LLNL Technical Report UCRL-JC-122359.

[2] C. Baldwin, P. N. Brown, R. D. Falgout, J. Jones, and F. Graziani. Iterative linear solvers in a 2d radiation-hydrodynamics code: methods and performance. Submitted to Journal of Computational Physics, 1998.

[3] V.A. Bandy, J. E. Dendy, and W. H. Spangenberg. Some multigrid algorithms for elliptic problems on data parallel machines. *SIAM J. Sci. Stat. Comput.*, 19:74–86, 1998.

[4] J. E. Dendy. Black box multigrid. *J. Comput. Phys.*, 48:366–386, 1982.

[5] J. E. Dendy, M. P. Ida, and J. M. Rutledge. A semicoarsening multigrid algorithm for SIMD machines. *SIAM J. Sci. Stat. Comput.*, 13:1460–1469, 1992.

[6] Message Passing Interface Forum, 1994. MPI: A message-passing interface standard. *Int. J. Supercomput. Applics.*, 8:159–416. http://www.mcs.anl.gov/mpi/mpi-report/mpi-report.html.

[7] J. E. Jones and S. F. McCormick. Parallel multigrid methods. In Keyes, Sameh, and Venkatakrishnan, editors, *Parallel Numerical Algorithms*, pages 203 – 224. Kluwer Academic, 1997.

[8] J. McCombs and S. Smith. Parallel performance of contaminate transport code on the IBM SP/2. unpublished manuscript.

[9] N. Naik and J. Van Rosendale. The improved robustness of multigrid solvers based on multiple semicoarsened grids. *SIAM J. Num. Anal.*, 30:215–229, 1993.

[10] J. W. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. F. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*, pages 73–130. SIAM, Philadelphia, PA, 1987.

[11] S. Schaffer. A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients. *SIAM J. Sci. Stat. Comput.* to appear.

[12] R. A. Smith and A. Weiser. Semicoarsening multigrid on a hypercube. *SIAM J. Sci. Stat. Comput.*, 13:1314–1329, 1992.