SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 6, pp. 1460–1469, November 1992

011

A SEMICOARSENING MULTIGRID ALGORITHM FOR SIMD MACHINES *

J. E. DENDY, JR.†, M. P. IDA‡, AND J. M. RUTLEDGE§

Abstract. A semicoarsening multigrid algorithm suitable for use on single instruction multiple data (SIMD) architectures has been implemented on the CM-2. The method performs well for strongly anisotropic problems and for problems with coefficients jumping by orders of magnitude across internal interfaces. The parallel efficiency of this method is analyzed, and its actual performance is compared with its performance on some other machines, both parallel and nonparallel.

Key words. multigrid, parallel computing

AMS(MOS) subject classifications. 65N20, 65W05

1. Introduction. Some previous papers have examined multigrid methods for their suitability for calculation on machines with SIMD architectures. Frederickson and McBryan [FM] developed and analyzed a method that was designed to keep all the processors busy. Decker analyzed [D1] the performance on SIMD machines of more traditional multigrid methods. Both of these papers were restricted to Poisson's equation with periodic boundary conditions; neither paper attempted to address the sort of problem we are interested in, namely,

$$-\nabla \cdot (D(x,y)\nabla U(x,y)) + \sigma(x,y)U(x,y) = F(x,y) \tag{1.1}$$

in a bounded region Ω of R^2, where $D = (D^1, D^2)$, D^i is positive, $i = 1, 2$, and D^i, σ, and F are allowed to be discontinuous across internal boundaries Γ of Ω; moreover, $D_1 \gg D_2$ and $D_1 \ll D_2$ in different subregions of Ω is possible.

Another method was advocated by Hackbusch [H] and was shown to be robust for constant coefficient, periodic, anisotropic problems. It can be argued that this method, or at least its precursor, may be found in [T]. We refer to this method as the Brandt–Hackbusch–Ta'assan method, if only to arrive at the acronym BHT. The BHT method, like the Frederickson–McBryan algorithm, preserves the busyness of the processors, and has the added advantage of only needing point relaxation for anisotropic problems. However, as shown in §4, the BHT method, as described by Hackbusch, does not handle problems like (1.1), and it is unclear how to give the method this capability.

Both the Frederickson–McBryan and BHT methods have the presumed advantage of keeping all the processors busy. However, idleness of processors is unimportant; what is important is the convergence factor per machine cycle. If keeping all the processors busy led to a significantly smaller convergence factor, then the busyness of processors would indeed be important. Also, busyness of processors may be an important issue only on grids with less than one point per processor (virtual processor ratio (VP ratio) less than 1). When the VP ratio is greater than 1, the CM-2 makes use of virtual processors. In effect, serial do loops (VP loops) are created on each

* Received by the editors March 25, 1991; accepted for publication (in revised form) October 25, 1991. The work of the first two authors was performed under the auspices of the U.S. Department of Energy under contract W-7405-ENG-36 and was partially supported by the Center for Research on Parallel Computation through National Science Foundation Cooperative Agreement CCR-8809615.

† Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

‡ California Institute of Technology, Pasadena, California 91125.

§ Chevron Oil Field Research Company, P. O. Box 446, La Habra, California 90631.

processor. Thus, when the VP ratio is greater than 1, "busy processor" methods actually incur a substantial computational penalty. Since moderate-size problems easily exceed VP ratios of 1 on today's machines, "busy processor" methods would seem to be, at best, methods for the future.

What about the more traditional methods of dealing with (1.1)? The first multigrid method to handle such problems successfully was given in [ABDP] and expanded in [D2]; it used standard coarsening (discussed below), interpolation induced by the operator, Galerkin coarsening, and alternating red-black line relaxation. An alternative was first discussed in [DMRRS] for three-dimensional problems. (However, we must point out that the robustness of line relaxation coupled with semicoarsening for constant coefficient anisotropic problems was first reported in [W].) The method discussed in this paper is the two-dimensional analogue of the method in [DMRRS]; it uses semicoarsening in y, interpolation induced by the operator, Galerkin coarsening, and red-black line relaxation by lines in x. Additionally, the method in this paper uses a technique due to Schaffer [S]; without this technique, the semicoarsening method would not be competitive. The method discussed in this paper is largely the same as the method given in [SW]. This fact should not be too surprising, since both papers had their genesis in a code written by Dendy.

One potential liability of the method considered in this paper is the necessity to perform line relaxation. The BHT method, were it robust, would avoid this difficulty. The suggestion was made in [B] that anisotropies could be avoided by the use of local grid refinement, under the assumption that physical problems are isotropic and that anisotropies arise from nonuniform gridding. One way to avoid nonuniform gridding is to use local grid refinement. In [D3], it was shown how to generalize [D2] to the case of local grid refinement. However, many person-years have been invested in codes that do not use local grid refinement, and all these codes would have to be rewritten to use this approach. Moreover, it is not clear how local grid refinement algorithms will perform on SIMD machines. Finally, there are important physical problems which are strongly anisotropic; an example is petroleum reservoir engineering [L]. For these problems, local grid refinement, although it may be desirable for other reasons, does not lead to isotropic problems on the local grids. Thus it appears that the issue of anisotropic problems must be directly attacked, not avoided.

Yet another method, due to Mulder [M], seems to have possibilities. The idea is that each grid has two offspring, one obtained by semicoarsening in x, the other by semicoarsening in y. When two offspring are of the same size in x and y, they are declared to be the same offspring. This method is discussed further in §2.

2. Standard versus semicoarsening. In [ABDP] and [D2] standard coarsening was used; that is, given a Cartesian grid, the coarser grid is obtained by deleting the even x- and y-lines. In this paper semicoarsening is used; that is, the coarser grid is obtained by deleting the even y-lines. To handle general anisotropic situations with standard coarsening seems to require alternating line relaxation, whereas with semicoarsening only, line relaxation by lines in x is required. (We note that there are some situations in both cases that cannot be handled by these choices of relaxation [ABDP], but for our purposes, these cases are pathological.) What is the sequential relaxation work for the two methods? Given an $nx \times ny$ grid, the relaxation work for semicoarsening is of the order of $2(nx)(ny)(1+1/4+\cdots) = (8/3)(nx)(ny)$. For semicoarsening, the relaxation work is of the order of $(nx)(ny)(1+1/2+\cdots) = 2(nx)(ny)$.

To do this same counting argument for a SIMD architecture requires a short discussion of the assumptions. We first consider the case where the VP ratio is less

than or equal to 1. In the simplest model for the CM-2, it is important that the arrays be "compatible," that is, of the same size; otherwise, great inefficiencies in communication result. Thus, given a fine grid and a coarser grid, it is assumed that the data for each grid are stored in compatible arrays. Thus every other row of the coarse grid matrix contains no useful information. Moreover, on a relaxation sweep, a mask is employed which makes the processors for these rows idle. The assumption, therefore, is that for every grid, the amount of work required to solve the collection of tridiagonal systems on that grid depends only on the number of x- or y-points on the finest grid. Thus the relaxation work in the standard coarsening case is $2W(nx)\log_2 ny$, and the relaxation work in the semicoarsening case is $W(nx)\log_2 ny$, where $W(nx)$ is the work to solve an $nx \times ny$ tridiagonal system. (There are $\log_2 ny$ grids, and on each grid the relaxation work is $2W(nx)$ or $W(nx)$, respectively.) If sparse Gaussian elimination (a.k.a. the Thomas algorithm) is used, $W(nx) = O(nx)$. If straightforward cyclic reduction is used, $W(nx) = O(\log_2 nx)$.

The above argument is correct for the tridiagonal solver currently implemented in CMSSL (CM Scientific Subroutine Library). However, we can write a tridiagonal solver that yields the following relaxation work estimate for the standard coarsening case:

$$\begin{aligned} 2\left(\log_2 nx + \log_2 \frac{nx}{2} + \cdots\right) &= 2(\log_2 nx + (\log_2 nx - 1) + \cdots) \\ &= 2\left(\log_2 nx \log_2 ny - \frac{1}{2}(\log_2 ny - 1)(\log_2 ny)\right) \\ &\simeq \log_2 nx \log_2 ny. \end{aligned}$$

The point is that when we know that, for example, every other processor is idle, the cyclic reduction algorithm can be started further along, eliminating every fourth point instead of every second point. For a VP ratio less than 1, however, communication dominates computation, so it is unlikely that this improved algorithm will be twice as fast as using the CMSSL tridiagonal solver, particularly since the latter is written in carefully optimized assembly language.

When the VP ratio is greater than 1, a hybrid algorithm [J] becomes the algorithm of choice for solving the tridiagonal systems; this algorithm performs the traditional Thomas algorithm sequentially on the serial loop part of the i-index on each physical processor and uses cyclic reduction to solve between physical processors; it is in fact implemented in the tridiagonal solver in CMSSL. This algorithm is also used in [SW]. It is ironic that the same algorithm is the efficient one for these different architectures (SIMD and MIMD (multiple instruction, multiple data)); however, the algorithm used in [SW] is actually a SPMD (single program, multiple data) algorithm.

When the VP ratio is much greater than 1, the serial work on each processor dominates, and the work estimate for relaxation reverts to the serial case. In any case, the semicoarsening algorithm appears to be more efficient for all VP ratios than the standard coarsening algorithm. The semicoarsening algorithm is also conceptually simpler, particularly in three dimensions, when the alternative to coarsening in z and performing xy-plane relaxations (using multigrid) [DMRRS] is standard coarsening, performing alternating plane relaxation [D5]. Bandy and Brickner [BB], however, are investigating the standard coarsening approach on the CM-2; hence, we should eventually be able to compare directly the two approaches.

We briefly discuss how the interpolation operators are derived, even though the description in [SW] is excellent. Let us denote the interpolation operator from the

coarse grid G^{k-1} to the fine grid G^k by I_{k-1}^k. (The coarse grid operator L^{k-1} is given by Galerkin coarsening from the fine grid operator L^k by forming $(I_{k-1}^k)^* L^k (I_{k-1}^k)$. We are interested in five-point or nine-point discretizations of (1.1); hence, we want the coarse grid operators also to be five- or nine-point operators.) In [ABDP], [D1], and [D3], I_{k-1}^k was described as follows: At coarse grid points coinciding with fine grid points, I_{k-1}^k is just the identity. At a fine grid point lying vertically between two coarse grid points, let the template of the operator be given by

$$\begin{pmatrix} NW & N & NE \\ W & C & E \\ SW & S & SE \end{pmatrix}. \tag{2.1}$$

Then I_{k-1}^k at $v_{i,j}$ is given by $av_{i,j-1} + bv_{i,j+1}$, where

$$a = -(SW + S + SE)/(W + C + E) \quad \text{and} \quad b = -(NW + N + NE)/(W + C + E).$$

That is, we think of summing away the x-dependence to obtain a three-point relation. A problem with this approach, when using standard coarsening, is that if $\rho = C - NW - N - NE - W - E - SW - S - SE$ is small, then instead of using $W + C + E$ in (2.1), we should use $SW + S + SE + NW + N + NE$ instead; this point is discussed in [D3]. With standard coarsening, results are relatively insensitive to switching between formulas based on the size of ρ; however, for semicoarsening this is not the case. With standard coarsening, interpolation is also being performed in the x-direction; hence, there is a possibility of coefficient variations being averaged out in that direction. In the semicoarsening case, some mechanism for averaging in the x-direction is apparently needed. Schaffer [S] also came to this conclusion and discovered the following scheme: Let

$$A^- v^- + A^0 v^0 + A^+ v^+ = 0$$

be the equation that would give the row $v^0 = (v_{i,j}, i = 1, \cdots, nx)$ in terms of the rows $v^- = (v_{i,j-1}, i = 1, \cdots, nx)$ and $v^+ = (v_{i,j+1}, i = 1, \cdots, nx)$. Then

$$v^0 = -(A^0)^{-1}(A^- v^- + A^+ v^+). \tag{2.2}$$

Unfortunately, use of (2.2) would lead to a nonsparse interpolation, leading to nonsparse coarse grid operators. Schaffer's idea is to assume that

$$-(A^0)^{-1}A^- \quad \text{and} \quad (-A^0)^{-1}A^+$$

can each be approximated by diagonal matrices in the sense that B^- and B^+ are diagonal matrices such that

$$-(A^0)^{-1}A^- e = B^- e \quad \text{and} \quad (-A^0)^{-1}A^+ e = B^+ e,$$

where e is the vector $(1, \cdots, 1)$. To find B^- and B^+ requires just two tridiagonal solves. The interpolation formula using B^- and B^+ is

$$v^0 = B^- v^- + B^+ v^+.$$

At first blush it would appear that the SIMD relaxation work of Mulder's algorithm [M] is comparable to the SIMD relaxation work of our method, since the number

of grids in Mulder's method, if $nx = ny$, is approximately $(\log_2 ny)^2$. In [NR], however, Van Rosendale and Naik (to be identified with the author of [D1]) show that the subgrids in the Mulder method can be organized so that relaxation on all grids simultaneously (concurrent relaxation) can be done efficiently. This approach leads to a degradation in convergence factor as well as processor-to-processor communication between grids for interpolation and residual weighting. In two dimensions [NR], the grids can be packed in such a way that communication between grids is efficient, but then the efficiency of relaxation suffers. Nevertheless, implementation of the Mulder method on the CM-2 is planned, as is a comparison with the method of this paper.

3. Implementation. Implementation issues are complicated by the fact that we are aiming at a moving target. The first version of this paper was written when the compiler on the CM-2 was the bit-serial version. Subsequently, this version was replaced by the slicewise compiler. For some time there will continue to be improvements in the compiler, operating system, and CMSSL. Rather than delay publication of this paper indefinitely, we have chosen to report the current status, and to try to guess what the effects of future developments will be.

In the first version of this paper we discussed the inefficiencies present in the large VP ratio case when compatible arrays for intergrid communication are assumed. Let us denote the i-index as the tridiagonal solver direction and the j-index as the multigrid coarsening direction. If the j-index is declared parallel, then the code compiles so that the VP loop on a physical processor always remains the same size, regardless of the coarse grid size. Thus, if the j-index is declared such that the VP loop size is 64, it remains so, instead of decreasing to 32, 16, etc., thus reflecting the inactive j-direction grid elements on coarser grids. One way to avoid this difficulty is to code the VP loop by splitting the j-index into parallel and serial parts; this kind of splitting was done for the CM-2 implementation (by Gyan Bhanot of Thinking Machines) of Jameson's FLOW 67 code, a timestepping multigrid code. In the first version of this paper, this splitting was done for the relaxation routine only, since the coding for this splitting is extremely cumbersome. Moreover, this splitting does not solve the problem of wasted storage.

A better solution than splitting the j-index into parallel and serial parts is to have arrays that are not compatible on fine and coarser grids, and to use temporary arrays to achieve compatibility. For definiteness, assume that a fine grid array A is $nx \times ny$, and that a coarser grid array B is $nx \times \frac{ny}{2}$. If it is desired that A communicate with B, every other row of A needs to be placed in a temporary $nx \times \frac{ny}{2}$ array C. This is an intraprocessor move of data and can be accomplished efficiently with a routine written by Brickner. This solution has two difficulties associated with it. The first is that it creates a ragged array data structure not supported by FORTRAN 8X. (A and B in the example are really $D(k, \cdots)$ and $D(k-1, \cdots)$.) This difficulty has been cured by a routine written in C which does dynamic storage allocation. With each array $D(k, \cdots)$ is associated an array descriptor that contains the information on the dimensions of $D(k, \cdots)$. Thus the FORTRAN 8X compiler can be fooled into accepting a ragged array data structure. The second difficulty is that the current compiler aggressively monitors array layouts to assure that arrays are evenly distributed on processors. In many applications, this aggressiveness is a good strategy; however, in this application, because the semicoarsening leads to rectangular (as opposed to square) arrays, it can lead to reallocation to different processors of points which need to communicate and which should be on the same processor. We attempted to bypass this reallocation by writing routines which essentially informed the compiler to leave our arrays alone.

Unfortunately, the compiler still intervened when creating temporary arrays, yielding not only inefficiencies but also wrong answers. Our current remedy has been to code at a lower level than FORTRAN 8X; this remedy solves the problem of the compiler trying to take control of the layout, but does not generate as efficient code per processor as the compiler is capable of generating.

One important aspect of this work has been to identify this compiler shortcoming. (We must temper these whinings with the observation that the slicewise compiler was created in an incredibly short time.) Ours has not been the only application in which it is desirable for the programmer to take away control of the layouts from the compiler, and indeed, new versions of the compiler have been promised which will provide this capability. In the standard coarsening case, however, we may not need this new compiler capability; in this case, the compiler's choice of layout may be acceptable.

A final comment for the large VP ratio case is that line relaxation performs with nearly the same efficiency as point relaxation, since most of the work is serial work done on each processor. For large problems, the work performed on the grids with the VP ratio less than or equal to 1 is a small part of the overall calculation. Related to this issue is the question of when the coarse grids calculation should be done on the front-end machine. For a powerful front-end machine, the coarse grids may have to be fairly fine before it even pays to invoke the CM-2's power. This issue is addressed further in §4.

There are at least two versions of the Thomas algorithm. One computes the LU-decomposition on the tridiagonal matrix as it is needed. Other versions save the LU-decomposition (one such version was exploited in [ABDP] to avoid expensive divides on the CDC-7600). There is an analogous situation with respect to cyclic reduction. In the first version of this paper, we found that a version that saves the LU-decomposition ran two times faster on the CM-2 than a version that recomputes the LU-decomposition. However, for cyclic reduction, the LU-decomposition must be stored at each level of the parallel reduction. The result, for the i-index, is that the requirement for storage is proportional to $nx(\log_2 nx)$, where nx is the number of i-grid points. However, for the hybrid version [J], the storage requirement of the LU-decomposition is just proportional to nx, assuming that we do not save the two-cyclic LU-decomposition needed for the processor boundary grid points. (For high VP ratios, this assumption is reasonable since the cyclic reduction part of the tridiagonal solves is a small fraction of the overall computational time.)

4. Results. Many authors present gigaflop rates as a figure of merit while others report on speedup (of many processors compared to a single processor). While both these measures are useful in comparing the improved running speed of a specific algorithm, they can be misleading in determining the most efficient algorithm to solve a given problem. Point Jacobi, for example, applied to solve a discretization of (1.1), has an impressive gigaflop rate and speedup factor; however, it cannot compete with the multigrid algorithm of this paper since its convergence factor is abysmally near 1 for large problems while the multigrid convergence factor stays nicely bounded away from 1. Hence what we concentrate on in this paper is actual timing data. The convergence factors for various problems for the multigrid algorithm are reported in detail in [SW] and need not be repeated here. Finally, it is impossible to give speedup data for a CM-2 since it is impossible to access just one processor; we do, however, compare performance of one-quarter of a 2048-processor machine with one-quarter of a 1024-processor machine.

We present timing comparisons of several machines in Table 1. The timing results are given as seconds per V-cycle and were obtained by running five V-cycles (including setup time) and dividing by 5. Thus the timing results are independent of the difficulty of the problem run. All of the CM-2 timings reported in this section were done using one-quarter of a 2048-Weitek-processor machine or one-quarter of a 1024-Weitek-processor machine. (These timings may be used to address, at least partially, the issues of speedup and scalability.) The front end for the CM-2 was a Sun 4/90. The iPSC/2 machine had 64 nodes of 386-type processors; several configurations of these processors were considered for each problem size; here we have reported the timings only for the best [SW].

TABLE 1
Time per V-cycle on three machines.

Size of problem	iPSC/2	CRAY Y-MP	CM-2, one-quarter of 1024 processors	CM-2, one-quarter of 2048 processors
32×32	0.3	0.01	−−	−−
64×64	0.7	0.04	0.65	0.77
128×128	2.0	0.09	0.99	0.80
256×256	−−	0.27	1.84	1.79
512×512	−−	0.95	4.55	3.04
1024×1024	−−	3.69	++	8.11
2048×2048	−−	−−	++	25.39

++ too large
−− no information

The timing results on the CRAY Y-MP were obtained in a time-sharing environment; with a dedicated Y-MP, we could have easily run problems larger than 2048×2048; however, if we had used all of a 2048-Weitek processor CM-2 we could have also run problems larger than 4096×4096. The results on the Y-MP show that great gains are easily made from vectorization for the smaller problems, but for the larger problems, the asymptote of time being linearly proportional to problem size has nearly been reached. The code used on the CRAY Y-MP uses only standard FORTRAN and is run only in single-processor mode. Presumably, speedups could be obtained using the multiprocessor capability of the Y-MP, but this is not considered here. The 512×512 and 1024×1024 cases were too large for the current loader and used a memory manager, to some obvious disadvantage in performance.

In the first version of this paper, we compared two versions of the hybrid tridiagonal solver: a version which recomputes the LU factorization on each call and a version which saves the LU factorization; the latter was about two times faster than the former. The figures in Table 1 use the CMSSL tridiagonal solve routine. This routine, which recomputes the LU factorization on each call, is as fast as the faster of our two versions since it is written in carefully optimized assembly language. If a factor-of-two speedup can be expected from the CMSSL tridiagonal solver which saves LU factorizations (not yet available), then we can expect to see nearly a factor-of-two decrease in the timings in Table 1 for the CM-2.

We report in Table 2 the effect of changing the size of the coarsest grid direct solve. The direct solve is done with a band solver on the front end with the LU-decomposition of that matrix being precomputed once. The problem in Table 2 has 256×256 grid points. (The timings in Table 2 use an earlier version of our code and should therefore only be considered in relation to each other and not to the timings in Table 1.) Note that there is a minimum for both the case of saving and not saving the cyclic reduction LU-decomposition. The reason for this is that on the coarser grids,

so few Weitek processors are active that the front end (which is considerably faster than one Weitek processor) is more efficient, even when the time to transfer the data from the CM-2 to the front end is taken into account.

TABLE 2
Time per V-cycle varying coarsest grid size.

Size of coarsest grid	Number of grid levels	Recompute LU	Save LU
256×1	9	8.57	3.22
256×4	7	6.53	2.66
256×8	6	5.87	2.69
256×16	5	5.98	3.50
256×32	4	8.41	6.62

One other possibility for gaining efficiency from this algorithm is to make use of the multiwire (multiple NEWS communication) library written by R. Brickner and documented in [CM]. The routines in this library allow one to do simultaneous communication and computation in each index of an array. We have investigated the use of this library in a preliminary way, but have postponed further work until the other issues above have been satisfactorily resolved.

Let us now consider the BHT method; first, for the problem

$$\begin{cases} -\triangle U + U = 1 & \text{in } (0,1) \times (0,1), \\ U & \text{doubly periodic.} \end{cases}$$

On a 16×16 grid we compare the result of using the method of [D4] (standard coarsening, operator-induced interpolation, Galerkin coarsening, and red-black point relaxation) to the method of [H]. (We use the method in [D4] because we have not yet extended the method in this paper to handle periodic boundary conditions.) Both methods achieve an average convergence factor, over ten V-cycles, of 0.05 per V-cycle.

Now let us consider the problem

$$\begin{cases} -\nabla \cdot (D \nabla U) + U = F & \text{on } (0,16) \times (0,16), \\ U & \text{doubly periodic,} \end{cases} \tag{4.1}$$

where D and F are as shown in Fig. 1. Using a 16×16 grid, we again compare [D4] to [H]; the average convergence factor per V-cycle is 0.06 vs 0.54, respectively. If we modify the method in [H] to attempt to use the same operator-induced interpolation used in [D4], we obtain an average convergence factor of 0.09 instead. The problem is that within the context of the method in [H], it is no longer clear what operator-induced interpolation should be.

Finally, let us comment that we believe that some multigrid method is likely to be the fastest algorithm for solving problems like (1.1) on SIMD machines. We intend to substantiate this belief by timing the algorithm of this paper against some possible competitors, such as preconditioned conjugate gradient methods. In this comparison, the convergence factor per unit time must be considered for various problems, since clearly conjugate gradient with no conditioning will win against multigrid on SIMD machines for well-conditioned elliptic problems. In one sense the contest is over before it starts, since we already have an example in which a preconditioned conjugate gradient method stagnates badly, but for which the method of this paper is robust.

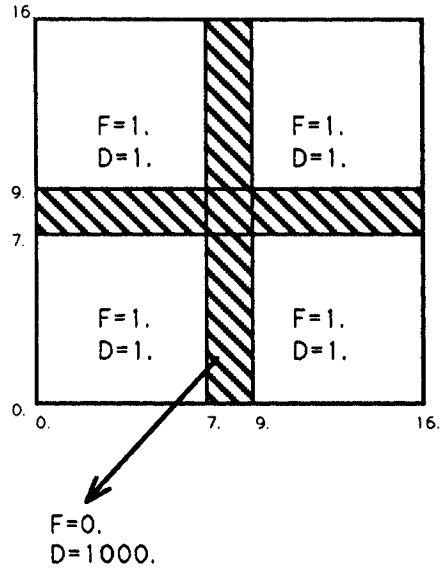

FIG. 1. *Diffusion coefficients and right-hand side for* (4.1).

Nevertheless, we hope to perform a comparison for a set of problems with a wide range of difficulty.

5. Conclusions. In this paper we have examined several multigrid methods in an attempt to find one that performs well on SIMD machines for problems with rough and anisotropic coefficients. We chose a semicoarsening multigrid algorithm for implementation on the CM-2 and have shown that it does perform well on that machine. We expect even better performance from this algorithm as compiler, operating systems, and library improvements become available.

Note added in proof. Recent advances in the version of BHT using operator-induced interpolation have led to an improved average convergence factor per V-cycle for (4.1): .09 per V-cycle instead of the .34 per V-cycle reported in this paper.

REFERENCES

[ABDP] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, JR., AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Statist. Comput. 2(1981), pp. 430–454.

[B] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31(1977), pp. 333–390.

[BB] V. BANDY AND R. BRICKNER, private communication.

[D1] N. DECKER, *On the parallel efficiency of the Frederickson–McBryan multigrid*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 208–220.

[D2] J. E. DENDY, JR., *Black box multigrid*, J. Comp. Phys., 48(1982), pp. 366–386.

[D3] ———, *A priori local grid refinement in the multigrid method*, in Elliptic Problems Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 439–452.

[D4] ———, *Black box multigrid for periodic and singular problems*, Appl. Math. Comput., 25 (1988), pp. 1–10.

[D5] ———, *Two multigrid methods for three-dimensional problems with discontinuous and anisotropic coeffcients*, SIAM Sci. Statist. Comput., 8 (1987), pp. 673–685.

[DMRRS] J. E. DENDY, JR., S. F. MCCORMICK, J. W. RUGE, T. F. RUSSELL, AND S. SCHAFFER, *Multigrid methods for three-dimensional petroleum reservoir simulation*, in Proc. Tenth Symposium on Reservoir Simulation, Houston, TX, February 6–8, 1989, pp. 19–25.

[FM] P. O. FREDERICKSON AND O. A. MCBRYAN, *Normalized convergence rates for the* PSMG *method*, SIAM J. Sci. Statist. Comput., 12 (1981), pp. 221–229.

[H] W. HACKBUSCH, *The frequency decomposition multigrid method, Part* I: *Application to anisotropic equations*, Numer. Math., 56(1989), pp. 229–245.

[J] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[L] L. W. LAKE, *The origins of anisotropy*, J. Petrol. Technology, April 1988, pp. 395–396.

[M] W. A. MULDER, *A new multigrid approach to convection problems*, J. Comput. Phys., 83 (1989), pp. 303–329.

[NR] N. NAIK AND J. VAN ROSENDALE, *The improved robustness of multigrid solvers based on multiple semicoarsened grids*, SIAM J. Numer. Anal., 31 (1993), to appear.

[S] S. SCHAFFER, private communication, manuscript.

[SW] R. A. SMITH AND A. WEISER, *Semicoarsening multigrid on a hypercube*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 1314–1329.

[T] S. TA'ASSAN, *Multigrid methods for highly oscillatory problems*, Ph. D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1984.

[W] G. WINTER, *Fourienanalyse zur Konstruktion schneller MGR-Verfahren*, Ph. D. thesis, Rheinischen Friedrich-Wilhelms-Universität zu Bonn, Bonn, Germany, 1982.