Semicoarsening Multigrid on a Hypercube

Richard A. Smith
Alan Weiser

CRPC-TR91112
January, 1991

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SEMICOARSENING MULTIGRID ON A HYPERCUBE

RICHARD A. SMITH [*] AND ALAN WEISER [†]

Abstract.
A semicoarsening multigrid algorithm suitable for the kinds of problems arising in reservoir simulation has been implemented on the Intel iPSC/2 hypercube. The method is an extension to nonsymmetric problems of a method in (Dendy *et al. SPE 18409*). It performs well for strongly anisotropic problems and problems with strongly discontinuous coefficients. For a test set of reservoir simulation problems, residual reduction factors for a full-multigrid V-cycle range from .0022 to .19. The current codes achieve about 50% parallel efficiency in two dimensions and about 30% parallel efficiency in three dimensions with about $\frac{\sqrt{N}}{8}$ processors for a grid with N unknowns.

[*] Exxon Production Research Company, Houston, TX 77252-2189.
[†] Center for Research on Parallel Computation, Rice University, Houston, TX 77251.

1. Introduction. Multigrid was first applied to reservoir simulation in the early 1980's [3]. For three-dimensional problems, multigrid was not found to be competitive with other solution methods because of the large expense per cycle of performing alternating direction smoothing by plane in the three coordinate directions x, y, and z. Subsequently, Dendy [6] reduced the cost of the method by using more efficient 2-d multigrid solvers for the smoothing subproblems. More recently, Dendy *et al.* [8] presented a semicoarsening version of multigrid for symmetric problems which performs well for reservoir simulation problems, with about the same (sequential) CPU cost per cycle but much simpler coding requirements than the previous multigrid methods.

We present an extension of this semicoarsening method to nonsymmetric problems. Among the features of the method are promising intergrid transfer operators due to Schaffer [7]. We investigate ways to implement the method efficiently on hypercube-type parallel processors, both to take advantage of current cost/benefit efficiencies of such machines and to predict how method performance scales with future massively parallel distributed memory computers.

In the following five sections respectively we describe our method, describe several ways of implementing it on hypercube machines, present convergence results, present parallel timing results, and indicate future plans for this project.

2. The sequential method. Consider a one-dimensional problem resulting from the one-dimensional pressure equation in reservoir simulation or any similar scalar second-order elliptic partial differential equation. The nonzero structure of the resulting tridiagonal matrix is depicted in Figure 1. Let P be the permutation matrix ordering the unknowns red-black. The resulting red-black matrix P^TAP is depicted in Figure 2.

In block form, the red-black linear system is

$$Ax = b \tag{1}$$

or

$$\begin{pmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{pmatrix} \begin{pmatrix} x_r \\ x_b \end{pmatrix} = \begin{pmatrix} b_r \\ b_b \end{pmatrix}. \tag{2}$$

The nonzeroes in Figures 1 and 2 are labeled so that each nonzero diagonal of A_{rr}, A_{rb}, A_{br}, or A_{bb} is labeled with a different letter. One way to solve (2) is to form and solve the smaller Schur complement linear system

$$A_s x_b = b_s \tag{3}$$

where

$$A_s = A_{bb} - A_{br}A_{rr}^{-1}A_{rb} \tag{4}$$

and

$$b_s = b_b - A_{br}A_{rr}^{-1}b_r, \tag{5}$$

and then backsolve

$$x_r = A_{rr}^{-1}(b_r - A_{rb}x_b). \tag{6}$$

Note that in this one-dimensional case A_s is tridiagonal like A.

Now consider a two-level multigrid method for the original system, where the black unknowns are the coarse grid unknowns. The main steps in such a method are:

i) smooth on the fine grid
ii) transfer the fine grid residual to the coarse grid and solve for the coarse grid correction
iii) transfer the correction back to the fine grid and add it in to the current solution
iv) smooth again on the fine grid

In the black-box multigrid framework [5] it is customary to construct the coarse grid system $A_c x_c = b_c$ as follows:

$$\begin{pmatrix} T_{br} & I_{bb} \end{pmatrix} \begin{pmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{pmatrix} \begin{pmatrix} T_{rb} \\ I_{bb} \end{pmatrix} \begin{pmatrix} x_c \end{pmatrix} = \begin{pmatrix} T_{br} & I_{bb} \end{pmatrix} \begin{pmatrix} b_r \\ b_b \end{pmatrix} \tag{7}$$

Here b_r and b_b denote the current residuals, and are only equal to the original right hand side if the current iterate is zero. I_{bb} denotes the coarse grid identity matrix: Pure injection is used for interpolating coarse grid unknowns to black unknowns on the fine grid.

A key observation is that if

$$T_{br} = -A_{br}A_{rr}^{-1} \tag{8}$$

and

$$T_{rb} = -A_{rr}^{-1}A_{rb} \tag{9}$$

then

$$\begin{aligned} A_c &= T_{br}A_{rr}T_{rb} + T_{br}A_{rb} + A_{br}T_{rb} + A_{bb} && (10)\\ &= A_{br}A_{rr}^{-1}A_{rb} - A_{br}A_{rr}^{-1}A_{rb} - A_{br}A_{rr}^{-1}A_{rb} + A_{bb} && (11)\\ &= -A_{br}A_{rr}^{-1}A_{rb} + A_{bb} && (12)\\ &= A_s && (13) \end{aligned}$$

and

$$b_c = -A_{br}A_{rr}^{-1}b_r + b_b = b_s, \tag{14}$$

so that the coarse grid multigrid system is identical to the Schur complement system. If the smoother on the fine grid solves exactly for the red unknowns in terms of the black unknowns, e.g., the smoother is red-black Gauss-Seidel, then the two-level multigrid solver is an exact solver. Furthermore if the coarse grid system itself is solved by a two-level multigrid solver, and so on recursively until the coarsest grid system with one unknown is solved directly, the resulting multigrid V-cycle is a direct solver.

Now consider a two-dimensional problem with a nine-point operator. The resulting nine-diagonal matrix is depicted in Figure 3. Let P be the permutation matrix ordering lines of unknowns red-black. The resulting red-black matrix P^TAP is depicted in Figure 4.

As in the one-dimensional case, if $T_{br} = -A_{br}A_{rr}^{-1}$ and $T_{rb} = -A_{rr}^{-1}A_{rb}$, then the coarse grid system for two-level multigrid is exactly the Schur complement system $A_s x_s = b_s$. However, the Schur complement matrix is dense. Fill occurs for two reasons. First, A_{rr}^{-1} has dense diagonal blocks, so T_{br} and T_{rb} have dense diagonal blocks. Second, extra fill occurs in direction 1. For instance, the black unknowns corresponding to (i1,i2) = (2,1) and (4,3) are both connected to the red unknown corresponding to (i1,i2) = (3,2) via T_{rb} and T_{br} terms. Elimination of red terms results in direct connections between (2,1) and (4,3) in A_s even though their indices differ by more than 1 in direction 1. The only way to avoid this extra fill is to allow connections in T_{br} and T_{rb} in direction 2 only. The locations in Figure 4 corresponding to connections in direction 2 only are the locations for diagonals c, g, k, and p.

Some new notation must be introduced to deal with connections in direction 2 only. Let an "l" (resp. "r") superscript appended to A_{rb} or A_{br} denote the result of zeroing out all entries except those corresponding to connection of a black unknown to a red unknown with a smaller (resp. larger) index in direction 2. Then $A_{rb} = A_{rb}^l + A_{rb}^r$ and $A_{br} = A_{br}^l + A_{br}^r$. For example, in Figure 4, A_{rb}^l may contain nonzeroes on diagonals c, d, and f, A_{rb}^r may contain nonzeroes on diagonals p, q, and r, A_{br}^l may contain nonzeroes on diagonals g, h, and m, and A_{br}^r may contain nonzeroes on diagonals k, l, and o. Similarly, let $diag_{rb}^l(v)$ (resp. $diag_{rb}^r(v)$, $diag_{br}^l(v)$, $diag_{br}^r(v)$) denote the matrix with the same nonzero structure as A_{rb} (resp. A_{rb}, A_{br}, A_{br}) with the only nonzero entries obtained from vector v and put on the diagonal corresponding to connections of black unknowns to red unknowns with lower (resp. higher, lower, higher) index in direction 2 and with the same index in direction 1. For example, in Figure 4, $diag^l_{rb}(v)$ (resp. $diag^r_{rb}(v)$, $diag^l_{br}(v)$, $diag^r_{br}(v)$) puts nonzeroes in the locations of diagonal c (resp. p, g, k).

With this notation, we define the following transfer operators due to Steve Schaffer at the New Mexico Institute of Mining and Technology [7]:

$$T_{rb} = -(diag^l_{rb}(A_{rr}^{-1}A^l_{rb}e) + diag^r_{rb}(A_{rr}^{-1}A^r_{rb}e)) \tag{15}$$

$$T_{br} = -(diag^l_{br}(e^T A^l_{br}A_{rr}^{-1}) + diag^r_{br}(e^T A^r_{br}A_{rr}^{-1})) \tag{16}$$

where e is the vector of all ones.

We use (15) and (16) for transfer operators. We take the initial guess for the solution to (1) on a given grid to be the best constant solution

$$x_0 = \frac{b^T e}{e^T A e} e \tag{17}$$

rather than zero. Here $(e^T Ae)^{-1}$ is precomputed and calculation of $b^T e$ is very cheap, involving only adds. For our smoother we use red-black line Gauss-Seidel with lines oriented in direction 1. This completes the specification of our basic two-level two-dimensional semicoarsening multigrid method. The adjective semicoarsening is used because the coarse grid is only coarsened in direction 2, while usual "full coarsening" multigrid involves coarsening in both directions 1 and 2 simultaneously.

The method works particularly well for strongly anisotropic problems. Suppose connections are much stronger in direction 1 than direction 2. Then line Gauss-Seidel is a good iterative method in direction 1. Also, entries in A_{rb} and A_{br} are small, so entries in T_{rb} and T_{br} are small, and by (7) A_c is a good approximation to A_s (and A_{bb}). Conversely suppose connections are much stronger in direction 2 than direction 1. Then A_{rr} is approximately diagonal, and A^l_{rb}, A^r_{rb}, A^l_{br}, and A^r_{br} are well-approximated by $diag^l_{rb}(A^l_{rb})$, $diag^r_{rb}(A^r_{rb})$, $diag^l_{br}(A^l_{br})$, and $diag^r_{br}(A^r_{br})$, respectively. Thus T_{rb} and T_{br} are very good approximations to $-A_{rr}^{-1}A_{rb}$ and $-A_{br}A_{rr}^{-1}$ and hence the coarse grid system is very close to the Schur complement.

This good convergence for strongly anisotropic problems is borne out in Tables 2, 3, and 4. (Here *ratio* is the anisotropy ratio in (18).)

Now consider a three-dimensional model problem with a 15-point operator (a 3 point operator in direction 3 tensored with a 5 point operator in directions 1 and 2). The resulting 15-diagonal matrix is depicted in Figure 5. Let P be the permutation matrix ordering planes of unknowns red-black. The resulting red-black matrix $P^T AP$ is depicted in Figure 6.

Again the same two considerations cause fill in the Schur complement matrix. Our notation to deal with connections in direction 3 is similar to the two-dimensional case. For Figure 6, A^l_{rb} has nonzeroes only in diagonals a and b, A^r_{rb} has nonzeroes only in diagonals c and d, A^l_{br} has nonzeroes only in diagonals e and f, and A^r_{br} has nonzeroes only in diagonals g and h. Also, $diag^l_{rb}(v)$ (resp. $diag^r_{rb}(v)$, $diag^l_{br}(v)$, $diag^r_{br}(v)$) has nonzeroes only in diagonal a (resp. c, e, g).

We would again like to use transfer operators (15) and (16). However, now A_{rr} is a 5-point operator which is expensive to invert. Hence, following Schaffer [7], we use one cycle of our two-dimensional semicoarsening multigrid method to approximately solve the A_{rr} systems needed in constructing T_{rb} and T_{br}.

We use (17) for initial guesses and approximate red-black Gauss-Seidel plane relaxation for our fine-grid smoothing step, where the two-dimensional semicoarsening multigrid method approximately solves the plane Gauss-Seidel equations. This completes the specification of our basic two-level three-dimensional semicoarsening multigrid method.

Note that a fine grid three-dimensional 7-point operator forms 15-point operators on the coarser three-dimensional grids, and a fine grid two-dimensional 5-point operator forms 9-point operators on the coarser two-dimensional grids.

Again this method works particularly well for strongly anisotropic problems. This good convergence for strongly anisotropic problems is borne out in Table 5.

We have used several standard multigrid cycling approaches. The "V-cycle" (Figure 7) starts on the finest grid with a two-level method and solves the resulting coarser grid system recursively with another two level method, and so on until the coarsest grid system with one unknown is solved directly. The "full multigrid V-cycle" or FMV cycle (Figure 8) generates an initial guess for the fine grid system by preceding the fine-grid V-cycle recursively with another V-cycle on the next-finest grid, and so on so that the very first calculation is a direct solve on the coarsest grid. The "initial FMV cycle" or IFMV cycle takes the first cycle to be a FMV cycle and the remaining cycles to be V-cycles.

Our 2-d sequential code requires about 30N words of storage, where N is the number of unknowns on the fine grid. Our 3-d sequential code requires about 148N words of storage. This is a large storage requirement for an iterative solver. It can be reduced if the matrix is known to be symmetric, if single precision is acceptable, or if the fine grids are known to be 7-point in 3-d and 5-point in 2-d rather than 15-point in 3-d and 9-point in 2-d.

3. The parallel method. We have implemented our method on distributed memory parallel computers of hypercube type by partitioning the problem domain into subdomains and assigning a rectangular subdomain to each node (processor). For each node, we pad local arrays by one at the lower and higher bounds of each array in each direction. The padding areas are used as buffers to exchange boundary information with neighbor nodes. Because the method mainly consists of subtasks of the form

```
for all unknowns in direction i
   perform task T in directions j and k
```

or

```
for all unknowns in direction i
   for all unknowns in direction j
      perform task T in direction k
```

where (i,j,k) = some permutation of (1,2,3), the parallelization tasks required in directions 1, 2, and 3 are essentially independent. Boundary data are exchanged as needed before loops. In the following example the call to *pad* exchanges boundary data for array *b* in direction 2, as needed by loop 10.

```
call pad(2,b)
do 10 i3=i3l,i3h
do 10 i2=i2l,i2h
do 10 i1=i1l,i1h
   a(i1,i2,i3) = b(i1,i2-1,i3) + b(i1,i2+1,i3)
10 continue
```

A major aspect of parallel implementation is treatment of coarse grids with no unknowns for some nodes (such grids are designated "below C-level" in Briggs *et al.* [4]). FMV-cycles tend to be relatively more expensive than V-cycles in parallel, because a greater portion of their computations are carried on below C-level. Both Hempel and Schuller [10] and Briggs *et al.* [4] use the "sleeping nodes" approach to C-level, in which nodes which are allocated no unknowns are set idle and then re-awakened when they rise above C-level again. We have taken two different approaches to C-level in our current two- and three-dimensional codes.

2-d. Our current two-dimensional code uses a 1-d global approach. When the grid goes below C-level in direction i, global copies of the current problem are distributed to all nodes in direction i. The distribution is performed using a 1-d version of a global concatenation operation (GCOL in [11]). Each node then proceeds to handle the global problem below C-level. No more internode communication is needed in this direction until computations resume above C-level. The penalty, of course, is that sub-C computations are duplicated many times, and extra communication is required for the global broadcasts.

This 1-d global approach does not scale well. As the number of nodes grows large so does the size of the sub-C level, and hence the parallel CPU time. However, for a moderate number of nodes good parallel efficiency is achieved, balancing the size of the sub-C level with the communication performed above the sub-C level. This approach can efficiently accommodate up to the order of $\sqrt{N}$ nodes, where $N = n1 \cdot n2$ is the number of fine grid points.

Solution of the tridiagonal 1-d linear systems in our two-dimensional code is consistent with this approach. We solve the tridiagonal systems with line Gauss-Seidel using a version of two-level cyclic reduction (Johnsson [12]). The steps in two-level cyclic reduction are:

1. Forward solve local interior unknowns in each subgrid
2. Broadcast and solve a small global system for boundary unknowns in each subgrid (1-d global concatenate in direction 1)
3. Backsolve local interior unknowns in each subgrid

(Figure 9). The small global system is distributed exactly at C-level. We combine the two-level cyclic reduction approach with the burn-at-both-ends (BABE) idea, assigning an upper and a lower node to each subgrid and eliminating unknowns for the two nodes for each subgrid in parallel (Figure 10).

3-d. Our current three-dimensional code handles C-level using a 1-d local duplication approach. As a node goes below C-level, it copies the local problem from a neighboring node. In this way all nodes stay busy working on systems of small size. This approach is closely related to the superconvergent parallel multigrid approach [9]. However, identical rather than different small systems are solved, so that the numerical answer is independent of the number of nodes.

As the grid gets coarser, the increment between neighboring nodes grows. Thus on the next-to-coarsest grid there are *nnode*/2 copies of small system 1, *nnode*/2 copies of small system 2, and a nodal increment to neighbors of *nnode*/2. This situation is illustrated in Figure 11, where there are 8 nodes and 16 fine grid unknowns, and *ninc* = nodal increment to neighbors.

Solution of the tridiagonal 1-d linear systems in our three-dimensional code is consistent with our local duplication approach: We use the one-dimensional semicoarsening multigrid direct solve outlined in Section 2, which is equivalent to fully recursive cyclic reduction.

The local duplication approach scales fairly well. As the number of nodes grows large, the amount of work done by each node is proportional to $log(N)$, the number of levels. However, in practice there is a penalty for using this approach, as well as the "sleeping nodes" approach — internode communication must continue at all sub-C levels.

Our 2-d hypercube code requires about 56N words of storage, where N is the number of unknowns on the fine grid local subdomain. Our 3-d sequential code requires about 136N words of storage. So far, storage has been the bottleneck in determining the size of problems we can run on hypercubes.

4. Convergence results. *2-d.* Our two-dimensional test problems are Neumann model problems

$$-(u_1)_1 - (ratio \cdot u_2)_2 = 1 \tag{18}$$

in the unit square,

$$\frac{\partial u}{\partial n} = 0 \tag{19}$$

on the boundary, with the matrix made nonsingular by doubling the first main diagonal entry.

The next few Tables depict residual reduction factors for various runs. The initial guess x_0 for x is taken to be 0. The residual reduction factor is averaged over 5 iterations as

$$(||b - Ax_5||_2/||b||_2)^{1/5}.$$

Factors less than .005 or so are affected by roundoff and may actually represent even faster convergence.

The effect of (17) is seen for *ratio* = 1, FMV-cycles in Table 1. Based on these results, (17) is used in all other runs reported.

V-cycle, IFMV-cycle (one FMV-cycle followed by 4 V-cycles) and FMV-cycle residual reduction factors for different values of *ratio* are presented in Tables 2, 3, and 4, respectively. Residual reduction factors are generally largest for *ratio* near one and smaller for *ratio* either very large or very small. Since an FMV-cycle costs roughly twice as much as a V-cycle, the IFMV-cycle

TABLE 1
Effect of (17).

$n1 = n2$	$x_0 = 0$	(17)
10	.054	.016
20	.070	.020
40	.14	.024
80	.24	.027

generally seems most efficient, with little more cost than a V-cycle and with residual reduction factors intermediate between those of V and FMV.

3-d. We use the test set of reservoir simulation problems from Dendy *et al.* [8]. The problems are symmetric except for Problem 7, which was symmetrized before solution in [8]. Several of the problems have large coefficient discontinuities. Average residual reduction factors are given in Table 5 for the results from [8] and for our code running with V-cycles, IFMV-cycles, and FMV-cycles. In these runs our code used two-dimensional FMV cycles to do the red-black smoothing by planes.

Our results for problems 6 and 7 depend strongly on ordering of axes. Other solvers that do particularly well on anisotropic problems share this property, e.g., nested factorization [2]. Table 6 presents average residual reduction factors for an FMV-cycle with our code for the various axis orderings. The results in Table 5 are given for ordering 312 for problem 6.

5. Parallel timing results.

2-d. Table 7 (resp. 8) presents timing results for a 5-iteration V-cycle (resp. FMV-cycle) of the 2-d code on the iPSC/2 hypercube at Oak Ridge with 64 Intel scalar 386 nodes. CPU is seconds of dedicated wall clock hypercube CPU time, and efficiency is

$$\frac{CPU_1}{nnodes \cdot CPU_{nnodes}}.$$

The OLM optimizing compiler was used. An S indicates that the run ran out of storage and the CPU time was estimated based on other runs. The *nnode*1 and *nnode*2 columns show the numbers of nodes in directions 1 and 2, respectively. The shown values resulted in the smallest CPU time for that value of $nnodes = nnode1 \cdot nnode2$. This usually occurred with an approximately square configuration of nodes. The * values achieve about 50% efficiency. This is obtained with up to about $n1/8$ nodes, scaling as expected. The .'s represent runs that were not made because of configuration or storage constraints.

3-d. Table 9 (resp. 11) presents timing results for a 5-iteration V-cycle (resp. FMV-cycle) of the 3-d code on the iPSC/2 64-node hypercube at Oak Ridge. Table 10 (resp. 12) presents the resulting efficiencies. For V-cycles, scaling behavior is more uniform than in the 2-d case, in that CPU time for a given grid consistently decreases as more nodes are used. Unfortunately, efficiencies are not as high as in the 2-d case. About 30% efficiency for V-cycles is achieved with $\sqrt{n1 \cdot n2 \cdot n3}/8$ nodes, with relatively higher efficiencies for finer grids. The lower efficiency in 3-d than 2-d may be partly due to the treatment of coarse grids, and partly due to the larger surface-to-volume ratio for 3-d grids than 2-d grids, with finer grids needed to get the same ratio of interior work to boundary work.

6. Plans.

Plans include trying different coarse grid approaches in 3 dimensions, timing our codes on the new generation of Intel RX hypercubes with faster communication networks, and writing a version of the codes for the Connection Machine.

We are grateful to Joel Dendy at Los Alamos for access to his sequential semicoarsening multigrid codes and for many helpful conversations; and to Mary Wheeler, Lawrence Cowsar, and Ashok Chilakapati at Rice University for helpful conversations.

Table 2
Two-dimensional V-cycle residual reduction factors.

ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.033	.079	.097	.11
100.	.078	.090	.11	.12
64.	.075	.091	.11	.12
32.	.072	.090	.10	.12
16.	.069	.086	.10	.11
8.	.061	.076	.090	.10
4.	.055	.068	.085	.099
2.	.055	.068	.084	.099
1.	.054	.067	.083	.098
.5	.053	.068	.082	.096
.25	.056	.065	.079	.093
.125	.050	.061	.077	.090
.0625	.035	.065	.073	.087
.03125	.017	.056	.071	.085
.015625	.0055	.039	.073	.080
.01	.0036	.025	.067	.078
.001	.0052	.0068	.0092	.047

Table 3
Two-dimensional IFMV-cycle residual reduction factors.

ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.0078	.017	.020	.023
100.	.028	.030	.037	.043
64.	.029	.034	.042	.048
32.	.033	.041	.048	.055
16.	.040	.048	.056	.064
8.	.045	.053	.061	.070
4.	.046	.053	.061	.071
2.	.044	.051	.059	.068
1.	.042	.050	.057	.066
.5	.042	.048	.056	.064
.25	.040	.046	.054	.062
.125	.032	.045	.052	.060
.0625	.020	.044	.049	.058
.03125	.0083	.036	.048	.056
.015625	.0030	.024	.047	.050
.01	.0035	.016	.043	.030
.001	.0051	.0069	.0092	.030

Table 4
Two-dimensional FMV-cycle residual reduction factors.

ratio	$n1 = n2 = 10$	$n1 = n2 = 20$	$n1 = n2 = 40$	$n1 = n2 = 80$
1000.	.0055	.0080	.010	.014
100.	.0037	.0051	.0068	.0091
64.	.0034	.0044	.0059	.0085
32.	.0029	.0041	.0057	.0078
16.	.0048	.0056	.0065	.0077
8.	.0090	.010	.012	.014
4.	.014	.016	.019	.021
2.	.017	.020	.023	.026
1.	.017	.019	.022	.025
.5	.014	.017	.019	.022
.25	.012	.015	.017	.019
.125	.011	.013	.015	.017
.0625	.010	.011	.013	.015
.03125	.0072	.011	.013	.015
.015625	.0037	.012	.014	.015
.01	.0034	.011	.012	.015
.001	.0052	.0068	.0092	.015

Table 5
3-d residual reduction factors.

problem	Dendy V	V	IFMV	FMV
1	.27	.044	.027	.012
2	.27	.059	.025	.0022
3	.27	.11	.011	.0070
4	.25	.058	.028	.0083
5a	.09	.21	.16	.054
5b	.02	.27	.22	.091
6	.29	.24	.20	.10
7	.27	.37	.29	.19

Table 6
3-d residual reduction factors.

ordering :	123	132	213	231	312	321
problem 6	.43	.13	.43	.31	.10	.31
problem 7	.19	>1	>1	>1	>1	.16

TABLE 7
2-d V-cycle hypercube timing results.

$n1 = n2$	*nnodes*	*nnode*1	*nnode*2	CPU	efficiency
32	1	1	1	3.8	1.0
	2	2	1	2.8	.7
	4	2	2	2.1	.5 *
	8	4	2	1.9	.3
	16	4	4	1.6	.1
	32	8	4	1.9	.1
	64	8	8	2.0	.
64	1	1	1	14.9	1.0
	2	2	1	9.7	.8
	4	2	2	5.9	.6
	8	4	2	4.8	.4 *
	16	4	4	3.8	.2
	32	8	4	3.8	.1
	64	8	8	4.8	.
128	1	.	.	59.6S	.
	2	.	.	.	.
	4	2	2	20.7	.7
	8	4	2	15.0	.5 *
	16	4	4	10.8	.3
	32	8	4	9.7	.2
	64	8	8	10.3	.1

TABLE 8
2-d FMV-cycle hypercube timing results.

$n1 = n2$	*nnodes*	*nnode*1	*nnode*2	CPU	efficiency
32	1	1	1	7.9	1.0
	2	2	1	5.4	.7
	4	2	2	4.7	.4 *
	8	4	2	4.4	.2
	16	4	4	4.0	.1
	32	8	4	4.0	.
	64	8	8	4.2	.
64	1	1	1	30.9	1.0
	2	2	1	18.1	.9
	4	2	2	12.5	.6
	8	4	2	9.9	.4 *
	16	8	2	7.7	.3
	32	8	4	6.8	.1
	64	16	4	7.3	.
128	1	.	.	123.6S	.
	2	.	.	.	.
	4	2	2	41.2	.8
	8	4	2	25.9	.6
	16	8	2	17.8	.4 *
	32	8	4	14.6	.3
	64	16	4	13.4	.1

TABLE 9
3-d hypercube V-cycle CPU seconds for given node configuration.

$n1 = n2$	$n3$	8x8x1	4x4x2	4x4x1	2x2x2	2x2x1	1x1x1
32	1	4.6	.	4.9	.	5.0	5.3
32	2	15.8	16.6	17.1	18.4	18.9	29.6S
32	4	27.8	28.4	31.3	32.4	38.7	70.0S
64	1	6.5	.	7.5	.	9.6	21.2S
64	2	22.6	26.5	27.6	.	.	118.4S
64	4	41.0	46.3	54.8	.	.	280.0S
128	1	9.3	.	13.2	.	.	84.8S

TABLE 10
3-d hypercube V-cycle parallel efficiencies.

$n1 = n2$	$n3$	8x8x1	4x4x2	4x4x1	2x2x2	2x2x1	1x1x1
32	1	<.1	.	.1	.	.3	1.0
32	2	<.1	.1	.1	.2	.4	.
32	4	<.1	.1	.1	.3	.5	.
64	1	.1	.	.2	.	.6	.
64	2	.1	.1	.3	.	.	.
64	4	.1	.2	.3	.	.	.
128	1	.1	.	.4	.	.	.

TABLE 11
3-d hypercube FMV-cycle CPU seconds for given node configuration.

$n1 = n2$	$n3$	8x8x1	4x4x2	4x4x1	2x2x2	2x2x1	1x1x1
32	1	14.3	.	14.2	.	13.6	11.5
32	2	63.7	64.1	66.5	64.0	65.7	74.4S
32	4	149.0	148.1	150.4	150.4	161.7	211.6S
64	1	22.9	.	23.8	.	27.3	46.0S
64	2	102.2	110.4	114.5	.	.	297.6S
64	4	242.3	257.5	284.3	.	.	846.4S
128	1	35.6	.	45.0	.	.	184.0S

TABLE 12
3-d hypercube FMV-cycle parallel efficiencies.

$n1 = n2$	$n3$	8x8x1	4x4x2	4x4x1	2x2x2	2x2x1	1x1x1
32	1	<.1	.	.1	.	.2	1.0
32	2	<.1	<.1	.1	.1	.3	.
32	4	<.1	<.1	.1	.2	.3	.
64	1	<.1	.	.1	.	.4	.
64	2	<.1	.1	.2	.	.	.
64	4	.1	.1	.2	.	.	.
128	1	.1	.	.3	.	.	.

$$\begin{pmatrix} a & b & & & & & \\ c & d & e & & & & \\ & f & a & b & & & \\ & & c & d & e & & \\ & & & f & a & b & \\ & & & & c & d & e \\ & & & & & f & a \end{pmatrix}$$

FIG. 1. *1-d matrix nonzero structure (n1=7).*

$$\begin{pmatrix} a & & & & b & & \\ & a & & & f & b & \\ & & a & & & f & b \\ & & & a & & & f \\ c & e & & & d & & \\ & c & e & & & d & \\ & & c & e & & & d \end{pmatrix}$$

FIG. 2. *1-d red-black matrix nonzero structure.*

REFERENCES

[1] Alcouffe, R. E., Brandt, A., Dendy, J. E., Jr., and Painter, J. W., *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Stat. Comp., 2 (1981), 430-454.

[2] Appleyard, J. R., and Cheshire, I. M., *Nested factorization*, paper SPE 12264 presented at the Seventh Society of Petroleum Engineers Symposium on Reservoir Simulation, Nov. 16-18, 1983.

[3] Behie, A., and Forsyth, P. A., Jr., *Multi-grid solution of the pressure equation in reservoir simulation*, Soc. Pet. Eng. J., 23 (1983), 623-632.

[4] Briggs, B., Hart, L., McCormick, S., and Quinlan, D., *Multigrid methods on a hypercube*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., Marcel Dekker, Inc., 1988.

[5] Dendy, J. E., Jr., *Black box multigrid*, J. Comp. Phys., 48 (1982) 366-386.

[6] Dendy, J. E., Jr., *Two multigrid methods for three-dimensional problems with discontinuous and anisotropic coefficients*, SIAM J. Sci. Stat. Comp., 8 (1987) 673-685.

[7] Dendy, J. E., Jr., private communication.

[8] Dendy, J. E., Jr., McCormick, S. F., Ruge, J. W., Russell, T. F., and Schaffer, S., *Multigrid methods for three-dimensional petroleum reservoir simulation*, paper SPE 18409 presented at the Society of Petroleum Engineers Symposium on Reservoir Simulation in Houston, TX, Feb. 6-8, 1989.

[9] Frederickson, P. O., and McBryan, O. A., *Parallel superconvergent multigrid*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., Marcel Dekker, Inc., 1988.

[10] Hempel, R., and Schuller, A., *Experiments with parallel multigrid algorithms using the SUPRENUM communications subroutine library*, GMD Studie 141, St. Augustin, April 1988.

[11] iPSC/2 Programmer's Reference Manual, Intel Corporation, 1988.

[12] Johnsson, S. L., *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Stat. Comp., 8 (1987), 354-392.

$$\begin{pmatrix}
a & b & & c & d & & & & & & & & & & \\
e & a & b & f & c & d & & & & & & & & & \\
& e & a & & f & c & & & & & & & & & \\
g & h & & i & j & & k & l & & & & & & & \\
m & g & h & n & i & j & o & k & l & & & & & & \\
& m & g & & n & i & & o & k & & & & & & \\
& & & p & q & & a & b & & c & d & & & & \\
& & & r & p & q & e & a & b & f & c & d & & & \\
& & & & r & p & & e & a & & f & c & & & \\
& & & & & & g & h & & i & j & & k & l & \\
& & & & & & m & g & h & n & i & j & o & k & l \\
& & & & & & & m & g & & n & i & & o & k \\
& & & & & & & & & p & q & & a & & \\
& & & & & & & & & r & p & q & e & a & b \\
& & & & & & & & & & r & p & & e & a
\end{pmatrix}$$

FIG. 3. *2-d matrix nonzero structure (n1=3, n2=5).*

$$\begin{pmatrix}
a & b & & & & & & & & c & d & & & & \\
e & a & b & & & & & & & f & c & d & & & \\
& e & a & & & & & & & & f & c & & & \\
& & & a & b & & & & & p & q & & c & d & \\
& & & e & a & b & & & & r & p & q & f & c & d \\
& & & & e & a & & & & & r & p & & f & c \\
& & & & & & a & b & & & & & p & q & \\
& & & & & & e & a & b & & & & r & p & q \\
& & & & & & & e & a & & & & & r & p \\
g & h & & k & l & & & & & i & j & & & & \\
m & g & h & o & k & l & & & & n & i & j & & & \\
& m & g & & o & k & & & & & n & i & & & \\
& & & g & h & & k & l & & & & & i & j & \\
& & & m & g & h & o & k & l & & & & n & i & j \\
& & & & m & g & & o & k & & & & & n & i
\end{pmatrix}$$

FIG. 4. *2-d red-black matrix nonzero structure.*

$$\left(\begin{array}{ccccccccccccccccccccccccccc}
x & x & & x & & & & & & a & b & & b & & & & & & & & & & & & & & \\
x & x & x & & x & & & & & b & a & b & & b & & & & & & & & & & & & & \\
 & x & x & & & x & & & & & b & a & & & b & & & & & & & & & & & & \\
x & & & x & x & & x & & & b & & & a & b & & b & & & & & & & & & & & \\
 & x & & x & x & x & & x & & & b & & b & a & b & & b & & & & & & & & & & \\
 & & x & & x & x & & & x & & & b & & b & a & & & b & & & & & & & & & \\
 & & & x & & & x & x & & & & & b & & & a & b & & & & & & & & & & \\
 & & & & x & & x & x & x & & & & & b & & b & a & b & & & & & & & & & \\
 & & & & & x & & x & x & & & & & & b & & b & a & & & & & & & & & \\
e & f & & f & & & & & & x & x & & x & & & & & & g & h & & h & & & & & \\
f & e & f & & f & & & & & x & x & x & & x & & & & & h & g & h & & h & & & & \\
 & f & e & & & f & & & & & x & x & & & x & & & & & h & g & & & h & & & \\
f & & & e & f & & f & & & x & & & x & x & & x & & & h & & & g & h & & h & & \\
 & f & & f & e & f & & f & & & x & & x & x & x & & x & & & h & & h & g & h & & h & \\
 & & f & & f & e & & & f & & & x & & x & x & & & x & & & h & & h & g & & & h\\
 & & & f & & & e & f & & & & & x & & & x & x & & & & & h & & & g & h & \\
 & & & & f & & f & e & f & & & & & x & & x & x & x & & & & & h & & h & g & h\\
 & & & & & f & & f & e & & & & & & x & & x & x & & & & & & h & & h & g\\
 & & & & & & & & & c & d & & d & & & & & & x & x & & x & & & & & \\
 & & & & & & & & & d & c & d & & d & & & & & x & x & x & & x & & & & \\
 & & & & & & & & & & d & c & & & d & & & & & x & x & & & x & & & \\
 & & & & & & & & & d & & & c & d & & d & & & x & & & x & x & & x & & \\
 & & & & & & & & & & d & & d & c & d & & d & & & x & & x & x & x & & x & \\
 & & & & & & & & & & & d & & d & c & & & d & & & x & & x & x & & & x\\
 & & & & & & & & & & & & d & & & c & d & & & & & x & & & x & x & \\
 & & & & & & & & & & & & & d & & d & c & d & & & & & x & & x & x & x\\
 & & & & & & & & & & & & & & d & & d & c & & & & & & x & & x & x
\end{array}\right)$$

FIG. 5. *3-d matrix nonzero structure ($n1=n2=n3=3$).*

$$\left(\begin{array}{ccccccccccccccccccccccccccc}
x&x&&x&&&&& & &&&&&&&& & a&b&&b&&&&&\\
x&x&x&&x&&&& & &&&&&&&& & b&a&b&&b&&&&\\
&x&x&&&x&&& & &&&&&&&& & &b&a&&&b&&&\\
x&&&x&x&&x&& & &&&&&&&& & b&&&a&b&&b&&\\
&x&&x&x&x&&x& & &&&&&&&& & &b&&b&a&b&&b&\\
&&x&&x&x&&&x & &&&&&&&& & &&b&&b&a&&&b\\
&&&x&&&x&x& & &&&&&&&& & &&&b&&&a&b&\\
&&&&x&&x&x&x & &&&&&&&& & &&&&b&&b&a&b\\
&&&&&x&&x&x & &&&&&&&& & &&&&&b&&b&a\\
&&&&&&&& & x&x&&x&&&&& & c&d&&d&&&&&\\
&&&&&&&& & x&x&x&&x&&&& & d&c&d&&d&&&&\\
&&&&&&&& & &x&x&&&x&&& & &d&c&&&d&&&\\
&&&&&&&& & x&&&x&x&&x&& & d&&&c&d&&d&&\\
&&&&&&&& & &x&&x&x&x&&x& & &d&&d&c&d&&d&\\
&&&&&&&& & &&x&&x&x&&&x & &&d&&d&c&&&d\\
&&&&&&&& & &&&x&&&x&x& & &&&d&&&c&d&\\
&&&&&&&& & &&&&x&&x&x&x & &&&&d&&d&c&d\\
&&&&&&&& & &&&&&x&&x&x & &&&&&d&&d&c\\
e&f&&f&&&&& & g&h&&h&&&&& & x&x&&x&&&&&\\
f&e&f&&f&&&& & h&g&h&&h&&&& & x&x&x&&x&&&&\\
&f&e&&&f&&& & &h&g&&&h&&& & &x&x&&&x&&&\\
f&&&e&f&&f&& & h&&&g&h&&h&& & x&&&x&x&&x&&\\
&f&&f&e&f&&f& & &h&&h&g&h&&h& & &x&&x&x&x&&x&\\
&&f&&f&e&&&f & &&h&&h&g&&&h & &&x&&x&x&&&x\\
&&&f&&&e&f& & &&&h&&&g&h& & &&&x&&&x&x&\\
&&&&f&&f&e&f & &&&&h&&h&g&h & &&&&x&&x&x&x\\
&&&&&f&&f&e & &&&&&h&&h&g & &&&&&x&&x&x
\end{array}\right)$$

FIG. 6. *3-d red-black matrix nonzero structure.*

$$\begin{array}{ccccccc}
4&&&&&&4\\
&3&&&&3&\\
&&2&&2&&\\
&&&1&&&
\end{array}$$

FIG. 7. *V-cycle.*

$$\begin{array}{cccccccccccccccc}
&&&&&&&&&4&&&&&&4\\
&&&&3&&&&3&&3&&&&3&\\
&2&&2&&2&&2&&&&2&&2&&\\
1&&1&&&&1&&&&&&1&&&
\end{array}$$

FIG. 8. *FMV-cycle.*

$$\begin{pmatrix}
n & o & & & f & & & & & & & \\
o & n & o & & f & & & & & & & \\
& o & n & o & f & & & & & & & \\
& & o & N & N & & & & & & & \\
& & & N & N & o & & & F & & & \\
& & & f & o & n & o & & f & & & \\
& & & f & & o & n & o & f & & & \\
& & & F & & & o & N & N & & & \\
& & & & & & & N & N & o & & \\
& & & & & & & f & o & n & o & \\
& & & & & & & f & & o & n & o \\
& & & & & & & f & & & o & n
\end{pmatrix}$$

n : nonzero
o : created zero
f : fill-in
N,F : part of small global system for boundary unknowns

FIG. 9. *Two-level cyclic reduction.*

$$\begin{pmatrix}
n & c & & & & & & \\
a & n & c & & & & & \\
& a & n & c & & & & \\
& & a & N & N & & & \\
& & & N & N & b & & \\
& & & & d & n & b & \\
& & & & & d & n & b \\
& & & & & & d & n
\end{pmatrix}$$

n : nonzero
a : created zero in initial stage for upper node
b : created zero in initial stage for lower node
N : part of small shared system for center unknowns
c : created zero in final stage for upper node
d : created zero in final stage for lower node

FIG. 10. *Burn-at-both-ends elimination.*

	node1	*node2*	*node3*	*node4*	*node5*	*node6*	*node7*	*node8*
ninc = 1	*ab*	*cd*	*ef*	*gh*	*ij*	*kl*	*mn*	*op*
coarsen ninc = 1	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
coarsen copy ninc = 2	*a*	*a*	*b*	*b*	*c*	*c*	*d*	*d*
coarsen copy ninc = 4	*a*	*a*	*a*	*a*	*b*	*b*	*b*	*b*
coarsen copy ninc = 8	*a*	*a*	*a*	*a*	*a*	*a*	*a*	*a*

FIG. 11. *Duplication of neighboring systems.*