Electronic Transactions on Numerical Analysis.
Volume 6, pp. 224-233, December 1997.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A PARALLEL MULTIGRID METHOD USING THE FULL DOMAIN PARTITION*

WILLIAM F. MITCHELL†

Abstract. The combination of adaptive refinement, multigrid and parallel computing for solving partial differential equations is considered. In the full domain partition approach, each processor contains a partition of the grid plus the minimum number of additional coarse elements required to cover the whole domain. A parallel multigrid algorithm using the full domain partition is presented. Multigrid rates of convergence have been observed while communicating between processors only twice per V-cycle. Numerical computations on a network of up to 32 workstations show that parallel efficiency rates of 50% to 90% can be obtained.

Key words. grid partitioning, multigrid, parallel algorithms.

AMS subject classifications. 65N30, 65N55, 65Y05, 65N50.

1. Introduction. The numerical solution of partial differential equations (PDEs) is the most computationally intensive part of mathematical modeling in many important applications. For this reason, much research has been performed to find faster methods to solve PDEs at high resolution. On sequential computers multilevel adaptive methods, i.e., methods that combine adaptive grid refinement and full multigrid, have been shown to have optimal efficiency for many classes of PDEs [1, 2, 4, 6, 8, 9]. However, effective implementation of these techniques on parallel computers is still not understood. Considerable research has been done to parallelize the individual components, but combining these results to form an effective parallel multilevel adaptive method remains a difficult challenge.

In this paper we consider a parallel multigrid algorithm for adaptive multilevel methods in the context of a Single Program Multiple Data (SPMD) programming model in a high-latency/low-bandwidth parallel architecture such as a cluster of workstations. In this environment the communication cost is high compared to the computation cost, so the frequency and size of messages between the processors must be kept small. We will focus on the finite element discretization of self-adjoint elliptic PDEs on bounded two dimensional domains using triangular elements, but the concepts presented here should be extendible to other solution techniques and problems.

The outer loop of the parallel adaptive multilevel method is:

```
start with very coarse grid
repeat
   adaptively refine grid
   partition new grid and redistribute
   apply multigrid cycle(s)
until solution is accurate enough
```

In the usual SPMD approach to parallelizing a PDE solver, the domain is partitioned into a number of sets equal to the number of processors. (In this paper the term "processor" should be interpreted as "virtual processor," since it is often advantageous to have each physical processor assume the role of more than one "processor.") Each processor is responsible for the data and computations of one set from the partition, and usually also contains shadow copies of grid entities that are outside but near its partition. The processors periodically update the shadow information through messages sent from the owner of the data to the processors that have shadow copies.

* Received May 15, 1997. Accepted for publication September 30, 1997. Communicated by C. Douglas.
† National Institute of Standards and Technology, Gaithersburg, MD. Contribution of NIST, not subject to copyright in the United States.

ETNA
Kent State University
etna@mcs.kent.edu

In the typical approach, shadow vertices are created for the vertices that are immediate neighbors of the vertices on the boundary of the partition. This provides a local copy of the data required by computations on this processor. In the full domain partition (FuDoP) approach [7] the shadow vertices extend over the full domain with the number of them decreasing exponentially as the distance from the boundary of the partition increases.

In Section 2, FuDoP and FuDoP with extended overlap are defined. Section 3 presents a multigrid algorithm that uses FuDoP to define a parallel algorithm that is nearly the same as the sequential algorithm while sending messages only twice per V-cycle. Numerical computations examining the multigrid rate of convergence and speedup are presented in Section 4.

2. FuDoP Distribution. Full Domain Partition (FuDoP) is a method of distributing a partitioned adaptive grid over the processors of a parallel computer. In particular, it determines how much additional (shadow) data should be placed on each processor. The approach is briefly described here. A more detailed description and analysis of the approach can be found in [7]. The FuDoP distribution provides a grid on each processor that covers the entire domain. This allows the definition of a parallel multigrid algorithm that is nearly the same as the sequential multigrid algorithm with only two communication phases per V-cycle. Moreover, since the FuDoP grid on each processor is just a particular adaptively refined grid, the sequential forms of the components of the adaptive multilevel algorithm (including adaptive refinement) are easily modified for parallel execution.

A nonuniform grid is generated by an adaptive refinement algorithm (see, for example, [5]) and partitioned into a number of sets equal to the number of processors. Each processor contains the triangles in one partition of the grid. The FuDoP distribution adds to each partition the minimum number of additional triangles required to cover the whole domain. The additional triangles are obtained from coarser refinement levels. The resulting grids are as if the processor had performed adaptive refinement with zero as the error indicator outside the region of its partition, so triangles outside that region are refined only as required for compatibility. (A triangulation is said to be compatible if the intersection of two triangles is either empty, a common vertex or a common side.) In fact, this is a reasonable way to implement the generation of FuDoP grids. Some triangles and vertices may be replicated over several processors, but each triangle and vertex has a unique processor, called the *owner*, which contains the definitive data should there be any discrepancies. Figure 2.1 illustrates an example of FuDoP grids. The top part of the figure shows an adaptive grid that has been partitioned into four sets, with the colors (or shades of grey on black and white displays) indicating the owners of the vertices. The remaining four parts of the figure show the FuDoP grid that is contained on each of the four processors. For example, the lower left part of the figure shows the grid contained on the “dark blue” (darkest shade of grey) processor with the vertices that the dark blue processor owns, and connecting line segments, colored in dark blue. The points that are not colored dark blue are shadow copies of points owned by other processors.

Another interpretation of FuDoP is obtained from domain decomposition with overlapping subdomains. In this context, domain decomposition refers to the process of partitioning the domain into regions, each of which will be assigned to a processor. The term “overlapping subdomains” means that the regions are not disjoint, and that grid elements in the overlap of two regions are replicated on both processors. Figure 2.2 illustrates a FuDoP grid on the “red” processor (the second darkest shade of grey) separated into refinement levels. The non-red triangles around the boundary are triangles owned by other processors. The grids illustrated in this figure are the grids of the multigrid solution method. When viewed level-by-level it is clear that, on each level, the additional triangles of the FuDoP grid form a thin boundary on the sides that are adjacent to other partitions. This is similar to domain decomposition with overlapping subdomains *on a level-by-level basis*, and is similar to a method proposed

ETNA
Kent State University
etna@mcs.kent.edu

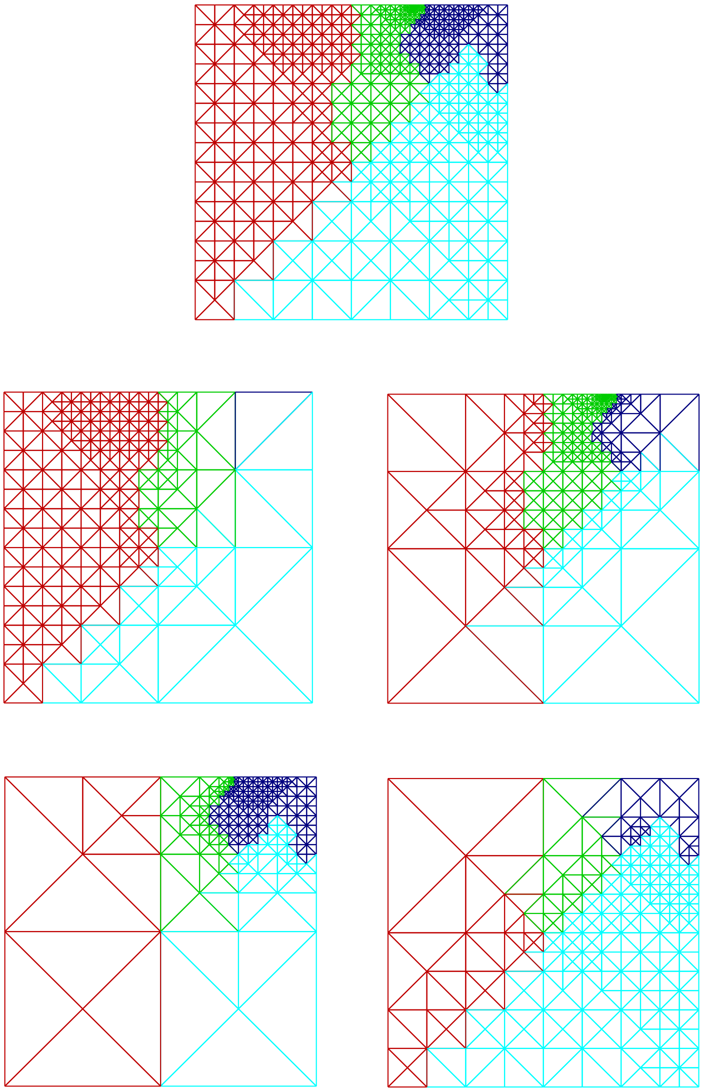

FIG. 2.1. *An adaptive grid partitioned for four processors and the corresponding FuDoP grids.*

by Brandt and Diskin [3] for parallel multigrid with finite differences on a uniform grid.

In its simplest form the FuDoP approach does not, however, provide all of the overlap of traditional overlapping Schwartz domain decomposition methods [10]. The overlap is traditionally defined by including all vertices that are neighbors of vertices in the subdomain. Two levels of overlap are achieved by including all vertices that are neighbors of neighbors, and higher levels of overlap are defined similarly. To achieve higher levels of overlap, the FuDoP approach can be extended by enforcing the requirement that on each level all vertices within a distance k of the partition be included. We refer to this as FuDoP with k levels of extended overlap, FuDoP(k). The nonextended FuDoP is denoted FuDoP(0) even though a considerable amount of overlap exists. The extended overlap can be generated by refining triangles to generate the vertices within k steps of the boundary, and any triangles required for compatibility.

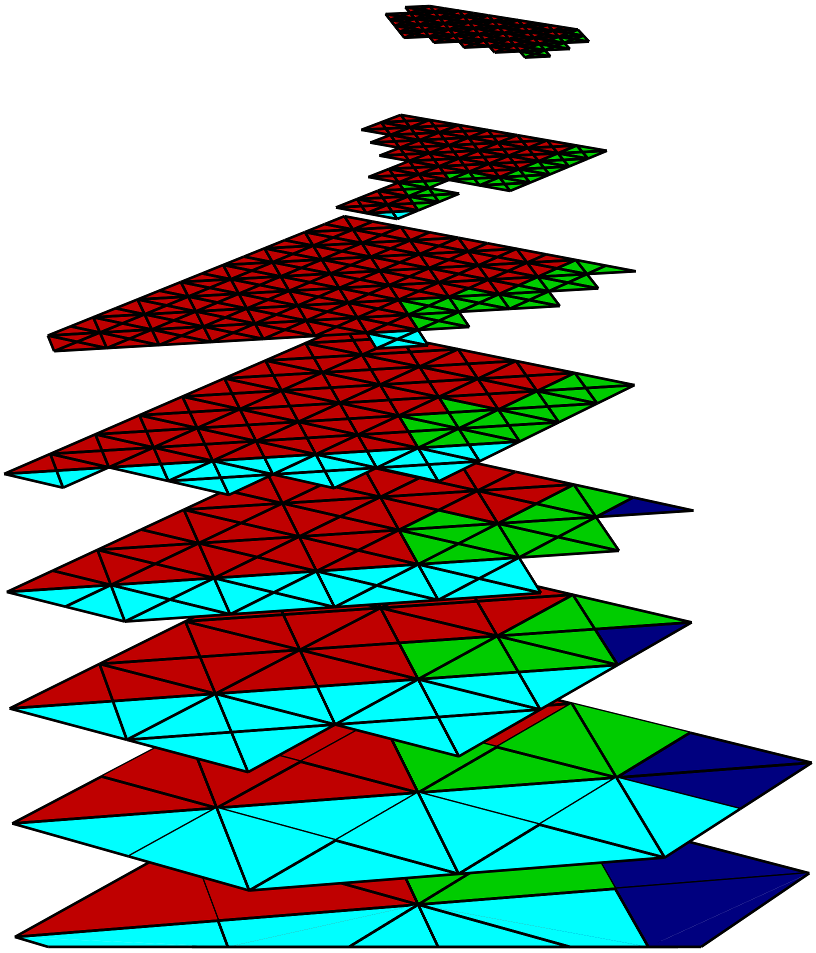

FIG. 2.2. *The individual levels of a FuDoP grid.*

3. Multigrid with FuDoP. In this section the use of FuDoP for parallelizing the multigrid method in [6] is described. This is a FAS (full approximation scheme) multigrid method [2] based on the hierarchical basis [11] over grids created by adaptive refinement using the newest node bisection of triangles [5].

The sequential form of one V-cycle of the algorithm can be briefly described as follows. The l^{th} grid in the sequence of L grids is obtained by using the first l levels of refinement in the adaptive grid. When working on level l, the points that are in grid $l-1$ are referred to as black points, and those that are in grid l but not in grid $l-1$ are referred to as red points. With the rows and columns ordered such that the black points come before the red points, the equations formed by using a two level hierarchical basis can be written

$$\begin{bmatrix} A_{bb} & A_{br}^T \\ A_{br} & A_{rr} \end{bmatrix} \begin{bmatrix} x_b \\ x_r \end{bmatrix} = \begin{bmatrix} f_b \\ f_r \end{bmatrix}.$$

(For simplicity of notation, the designation of the associated grid level is omitted.) Note that the coarse-coarse block, A_{bb}, of the two level hierarchical matrix is the nodal matrix of the coarse grid. This suggests the use of

$$A_{bb}x_b \;=\; f_b \;-\; A_{br}^T x_r$$

as the coarse grid problem, and such a formulation can be shown to be equivalent to standard multigrid [6]. With this formulation, the V-cycle can be described as:

Algorithm Sequential V-Cycle
for l = L downto 2
    perform red relaxation
    convert to two level hierarchical basis of level l
    $f_b \;\leftarrow\; f_b \;-\; A_{br}^T x_r$
end for
solve coarse grid problem on level 1
for l = 2 to L
    $f_b \;\leftarrow\; f_b \;+\; A_{br}^T x_r$
    convert to nodal basis of level l
    perform red relaxation
    perform local black relaxation
end for

Here, red relaxation is a Jacobi or Gauss-Seidel sweep over the red points, i.e., the equations corresponding to x_r. Local black relaxation is a Jacobi or Gauss-Seidel sweep over the black points that are immediate neighbors of the red points. The coarsest grid problem is solved exactly or to high accuracy. The conversion between nodal and two level hierarchical bases is detailed in [6] and requires a small number of operations per red point.

The FuDoP grids provide a means of adapting this algorithm for parallel computation. Since the FuDoP approach provides each processor with a compatible, hierarchical adaptive grid that covers the whole domain, the multigrid algorithm can be run on each processor in parallel. Obviously, some level of communication will be necessary, since the individual FuDoP grids do not contain sufficient resolution over the entire domain. Numerical studies with Laplace's equation and a moderate number of processors (see Section 4) indicate that it is sufficient to send messages only twice during a V-cycle to maintain nearly the same rate of convergence as the sequential algorithm. These messages occur at the top and bottom of the V-cycle.

The key difference between the sequential and parallel versions is the computation of the $A_{br}^T x_r$ term that appears in the restriction from the fine grid to the coarse grid. None of the processors contain all of the data to compute this term, so the computation is distributed over the processors. For each processor, the result can be decomposed into three parts:

1. the part due to elements of x_r that are owned by this processor,
2. the part due to elements of x_r that this processor has a copy of, but does not own, and
3. the part due to elements of x_r for which this processor does not have a copy.

These three components are handled individually in the parallel algorithm.

The contribution of $A_{br}^T x_r$ to the right hand side is kept in two vectors, r_m and r_o, rather than being incorporated into f as in the sequential algorithm. The vector r_m ("m" for "my contribution") contains part 1, while the vector r_o ("o" for "other's contributions") contains parts 2 and 3. With this notation, the linear system approximately solved on each level has the form

$$Ax \;=\; f - r_m - r_o.$$

Each processor can compute its r_m since it has all the data for that part of $A_{br}^T x_r$. If x_r' denotes x_r with the part 2 and part 3 entries replaced by zero, then the part 1 contribution can be written as $A_{br}^T x_r'$. Each processor can also approximately compute the part 2 contribution to r_o, since it has copies of all the data required for this computation. The resulting values will not be exact since the owner of the data will have updated x_r since the last communication step, which occurred immediately before the V-cycle began. If $\bar{x}_r'$ denotes x_r with the part 1 and part 3 entries replaced by zero, then the part 2 contribution can be written as $A_{br}^T \bar{x}_r'$. The remainder of r_o, due to part 3, cannot be computed by the processor since it has none of the required data. However, as will be seen, r_o can contain an approximation to this part based on values that lag by one iteration.

The first half of the V-cycle can proceed without communication in this manner. When the first half is complete (before the solution of the coarsest grid problem), the processors exchange information. To each shadow copy of the vertices the processor owns, it sends the current hierarchical solution value and the value of r_m, which replaces the approximation in r_o on the receiving processor. After this communication all processors have the same values for x and $r_m + r_o$.

In the second half of the V-cycle, the part 2 contributions to r_o are eliminated exactly by adding $A_{br}^T \bar{x}_r'$ because this operation occurs before x_r is changed by red relaxation. The part 3 contribution to $A_{br}^T x_r$ remains in r_o and can be used in the first half of the next V-cycle, with a one iteration lag in the solution values.

The difference between the results obtained by the second half of the sequential and parallel versions is due to the local black relaxation. First, the values in r_o for black points with neighbors owned by other processors will be inaccurate due to the part 3 contributions remaining from coarser levels. Second, because some of the neighbors of the local black points are shadow copies and not current, they will cause slight inaccuracies in the solution values at the black points, which will infect the red points at the next level. This can be eliminated by using a sufficiently large overlap (k in FuDoP(k)), however numerical experiments (not shown) indicate that this has a much smaller effect on the convergence rate than the inaccuracies in r_o, so this alone probably does not warrant the extra overlap.

In summary, the parallel version of the V-Cycle algorithm can be described as:

Algorithm Parallel V-Cycle

for l = L downto 2
    perform red relaxation
    convert to two level hierarchical basis of level l
    $r_m \leftarrow r_m - A_{br}^T x_r'$
    $r_o \leftarrow r_o - A_{br}^T \bar{x}_r'$
end for
send r_m and x to shadow copies on other processors
solve coarse grid problem
for l = 2 to L
    $r_m \leftarrow r_m + A_{br}^T x_r'$
    $r_o \leftarrow r_o + A_{br}^T \bar{x}_r'$
    convert to nodal basis of level l
    perform red relaxation
    perform local black relaxation
end for
send x to shadow copies on other processors

TABLE 4.1
Contraction factors for a uniform grid with 1089 vertices.

	Processors						
Overlap	1	2	4	8	16	32	64
FuDoP(0)	.090	.178	.251	.214	.236	.214	.268
FuDoP(1)	.090	.092	.093	.095	.097	.097	.098
FuDoP(2)	.090	.092	.093	.095	.097	.096	.098

TABLE 4.2
Contraction factors for a uniform grid with 66049 vertices.

	Processors						
Overlap	1	2	4	8	16	32	64
FuDoP(0)	.095	.179	.244	.215	.240	.224	.254
FuDoP(1)	.095	.095	.096	.098	.100	.104	.104
FuDoP(2)	.095	.095	.096	.098	.100	.102	.104

4. Numerical Results. Numerical computations were performed to examine the rate of convergence of the multigrid method with FuDoP. Tables 4.1 and 4.2 present the results for uniform grids of approximately 1000 and 64000 vertices, respectively, using from 1 to 64 processors and with extended overlaps of 0, 1 and 2. Laplace's equation was solved on the unit square with Dirichlet boundary conditions defined such that the solution is $u(x, y) = x + y$. The finite element method with linear elements solves this problem exactly, so convergence of the algebraic error can be examined without interference from the discretization error. The initial guess for the solution is set to 1.0 for all internal vertices and the energy norm of the error is measured. Ten iterations of the V-cycle are performed. After each cycle the energy norm of the error is measured and the contraction factor, i.e. the ratio of the error after the cycle to the error before the cycle, is computed. The reported value is the maximum of the ten computed contraction factors.

Tables 4.3 and 4.4 present similar results for nonuniform grids. Here the boundary conditions for Laplace's equation are piecewise linear across the top boundary, going from 0.0 at $x = 0.0$ to 1.0 at $x = 0.7$ to 0.0 at $x = 1.0$, and zero on the other three sides. This problem generates the adaptive grid of Figure 2.1. Since the exact solution is not known, the algebraic error is computed by comparing the solution after each cycle with the solution obtained by fifteen V-cycles.

The following observations can be made from these tables:

1. There is no significant difference between the small grid and the large grid results, illustrating the independence of the multigrid rate of convergence with respect to grid size.
2. The rate of convergence with FuDoP(0) deteriorates with the number of processors, but may still be bounded away from 1.0 as a function of grid size.
3. FuDoP(1) has nearly the same rate of convergence as the sequential algorithm (processors=1), and shows little degradation with the number of processors.
4. FuDoP(2) does not provide any significant improvement over FuDoP(1).
5. There is little difference between the rate of convergence on nonuniform grids and uniform grids.

Numerical computations were also performed to examine the speedup obtained by parallel computing. Each processor contained exactly one "virtual processor" in these computations. Tables 4.5 and 4.6 present these results.

TABLE 4.3
Contraction factors for an adaptive grid with about 1000 vertices.

	Processors						
Overlap	1	2	4	8	16	32	64
FuDoP(0)	.089	.178	.181	.272	.278	.279	.293
FuDoP(1)	.089	.089	.090	.091	.093	.110	.111
FuDoP(2)	.089	.090	.090	.090	.092	.095	.098

TABLE 4.4
Contraction factors for an adaptive grid with about 60000 vertices.

	Processors						
Overlap	1	2	4	8	16	32	64
FuDoP(0)	.093	.178	.203	.221	.332	.317	.273
FuDoP(1)	.093	.093	.094	.099	.098	.101	.098
FuDoP(2)	.093	.094	.094	.098	.099	.180	.163

TABLE 4.5
Speedup results for fixed problem size.

computer (vertices)	nproc	time (s)	speedup	efficiency
PPro	1	3.16	–	–
(16K)	2	1.53	2.06	1.03
	4	1.07	2.95	.74
	8	.60	5.27	.66
SP2	1	14.09	–	–
(64K)	2	7.75	1.82	.91
	4	4.20	3.35	.84
	8	3.48	4.04	.51
	16	1.76	8.01	.50
	32	4.22	3.34	.21

TABLE 4.6
Speedup results for scaled problem size.

computer (vertices)	nproc	time (s)	scaled speedup	scaled efficiency
PPro	1	3.16	–	–
(16K	2	3.79	1.67	.83
per proc)	4	3.13	4.04	1.01
	8	3.76	6.72	.84
SP2	1	14.09	–	–
(64K	2	15.71	1.79	.90
per proc)	4	16.62	3.39	.85
	8	17.89	6.30	.79
	16	20.83	10.82	.68
	32	25.31	17.81	.56

ETNA
Kent State University
etna@mcs.kent.edu

Two parallel environments were examined:

1. The results labeled "PPro" were obtained with 200MHz Pentium Pro-based computers with 128 Mbytes of memory, connected by fast ethernet (100 BaseT). NA-Software FortranPlus Version 1.3.5 and PVM Version 3.3.11 were used under Linux 2.0.28. The cluster was dedicated to a single user.
2. The results labeled "SP2" were obtained from an IBM SP2 with 33 "thin" nodes, which are 66.7 MHz RS6000s with 256 Mbytes of memory. The processors were used in single user (batch) mode. AIX XLF Version 3.2 and PVM Version 3.3.11 were used under AIX 4.1.4.

Table 4.5 presents the results for fixed problem size. A uniform grid with approximately 16,000 vertices (64,000 for the SP2) was generated, and the "wall clock" time for executing four V-cycles was measured. The speedup, defined as the time spent on one processor divided by the time spent on p processors, and the efficiency, defined as the speedup divided by the number of processors, are also presented.

Table 4.6 presents the results for scaled problem size. Here the grid contains approximately 16,000p vertices (64,000p for the SP2), where p is the number of processors, i.e., the number of vertices per processor remains fixed. The scaled efficiency is defined as the time spent on one processor divided by the time spent on p processors, and the scaled speedup is defined as the scaled efficiency times the number of processors.

These results show that the efficiency and scaled efficiency of the parallel multigrid method is typically between 50% and 90% in these environments for moderately large sized problems. As expected with all-to-all communication, the efficiency drops off as the number of processors gets large, and is poor when the number of vertices per processor is small.

5. Conclusion. The Full Domain Partition with extended overlap is an easily generated extension of the partitions of an adaptive grid that, level by level, provides a small overlap with the other partitions, and globally provides an extension that covers the full domain. When used with a multigrid method, it provides a natural domain decomposition with overlapping subdomains on each level. Numerical computations on up to 64 processors demonstrate that, with FuDoP(1), a parallel implementation of a standard multigrid method for Laplace's equation can obtain nearly the same rate of convergence as the sequential method for a modest number of processors, with only two communication steps per V-cycle. Numerical computations with up to 32 processors show that parallel efficiency rates of 50% to 90% can be obtained, with the lower efficiencies coming from the larger number of processors. The FuDoP approach is probably not appropriate for massively parallel computers, but provides high efficiency in workstation cluster environments.

6. Disclaimer. The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology. All trademarks mentioned herein belong to their respective owners.

REFERENCES

[1] R. E. Bank, *PLTMG: a software package for solving elliptic partial differential equations*, Frontiers in Applied Mathematics, vol. 15, SIAM, Philadelphia, 1994.

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

[3] A. Brandt and B. Diskin, *Multigrid solvers on decomposed domains*, in Proceedings of the Sixth International Conference on Domain Decomposition Methods, A. Quarteroni, ed., AMS, Providence, 1994, pp. 135-155.

[4] S. F. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations*,Frontiers in Applied Mathematics, vol. 6, SIAM, Philadelphia, 1989.
[5] W. F. Mitchell, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Trans. Math. Soft., 15 (1989), pp. 326–347.
[6] ———, *Optimal multilevel iterative methods for adaptive grids*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 146–167.
[7] ———, *The full domain partition approach to distributing adaptive grids*, Appl. Numer. Math., 26 (1998), pp. 265–275.
[8] M.-C. Rivara, *Design and data structure of fully adaptive, multigrid, finite-element software*, ACM Trans. Math. Soft., 10 (1984), pp. 242–264.
[9] U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*,Frontiers in Applied Mathematics, vol. 13, SIAM, Philadelphia, 1993.
[10] B. Smith, P. Bjørstad and W. Gropp, *Domain Decomposition*, Cambridge University Press, Cambridge, 1996.
[11] H. Yserentant, *On the multi–level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.