THE FULL DOMAIN PARTITION APPROACH TO PARALLEL ADAPTIVE REFINEMENT

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 adaptive refinement algorithm using the full domain partition is presented. The method is a small modification of a sequential adaptive refinement algorithm, and uses no interprocessor communication during the refinement process. The only communication is one global reduction before refinement and three all-to-all communication steps for synchronization after the refinement is completed. Numerical computations on a network of up to 4 workstations show that parallel efficiency rates of 85% to near 100% can be obtained.

Key words. adaptive refinement, finite elements, parallel algorithms, partial differential equations.

AMS(MOS) subject classifications. 35J25, 65N30, 65N50, 65N55, 65Y05.

1. Introduction. The numerical solution of partial differential equations (PDEs) is the most computationally intensive part of solving mathematical models in many important applications. For this reason, much research has been performed to find faster methods to solve PDEs at higher 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, 7, 9, 15, 16]. 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 adaptive refinement 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 a two dimensional PDE using triangular elements, but the concepts presented here should be extendible to other solution techniques.

The high level form of the parallel adaptive multilevel method is:

---

*National Institute of Standards and Technology, Gaithersburg, MD 20899.

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

This paper focuses on the adaptive refinement part of the algorithm. An approach to grid partitioning is presented in [12] and multigrid in [11].

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.

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 [10] 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 an adaptive refinement algorithm that uses FuDoP to avoid communication during the adaptive refinement process. In this algorithm the only communication is one global reduction before refinement and three all-to-all communication steps for synchronization after the refinement is completed. Previous parallel adaptive refinement algorithms [5, 13, 17] have required many communication steps during the refinement process, creating a large ratio of number of messages to computation. Section 4 presents numerical results demonstrating the speedup that can be achieved by the FuDoP parallel adaptive refinement algorithm.

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 [10].

A nonuniform grid is generated by an adaptive refinement algorithm (see, for example, [8]) 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, and will be the basis of the parallel adaptive refinement algorithm in Section 3. 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 1 illustrates an example of FuDoP grids. The top part of the figure shows an adaptive grid that has been partitioned into four sets. The remaining four parts of the figure show the FuDoP grid that is contained on each of the four processors.

Another interpretation of FuDoP is obtained from domain decomposition with overlapping subdomains. Figure 2 illustrates a FuDoP grid from one processor separated into refinement levels. The grey triangles are owned by the processor and the white triangles around the boundary are triangles owned by other processors. 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 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, contain the traditional overlap which is defined by including all vertices that are neighbors of vertices in the subdomain. Higher levels of overlap are also often used and are defined by including all vertices that are neighbors of neighbors for two overlap levels, or defined similarly for more levels of overlap. 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.

3. Adaptive refinement with FuDoP. The Full Domain Partition was originally developed as a means of parallelizing a multigrid method. In this section the advantages of FuDoP for parallelizing adaptive refinement algorithms are introduced.

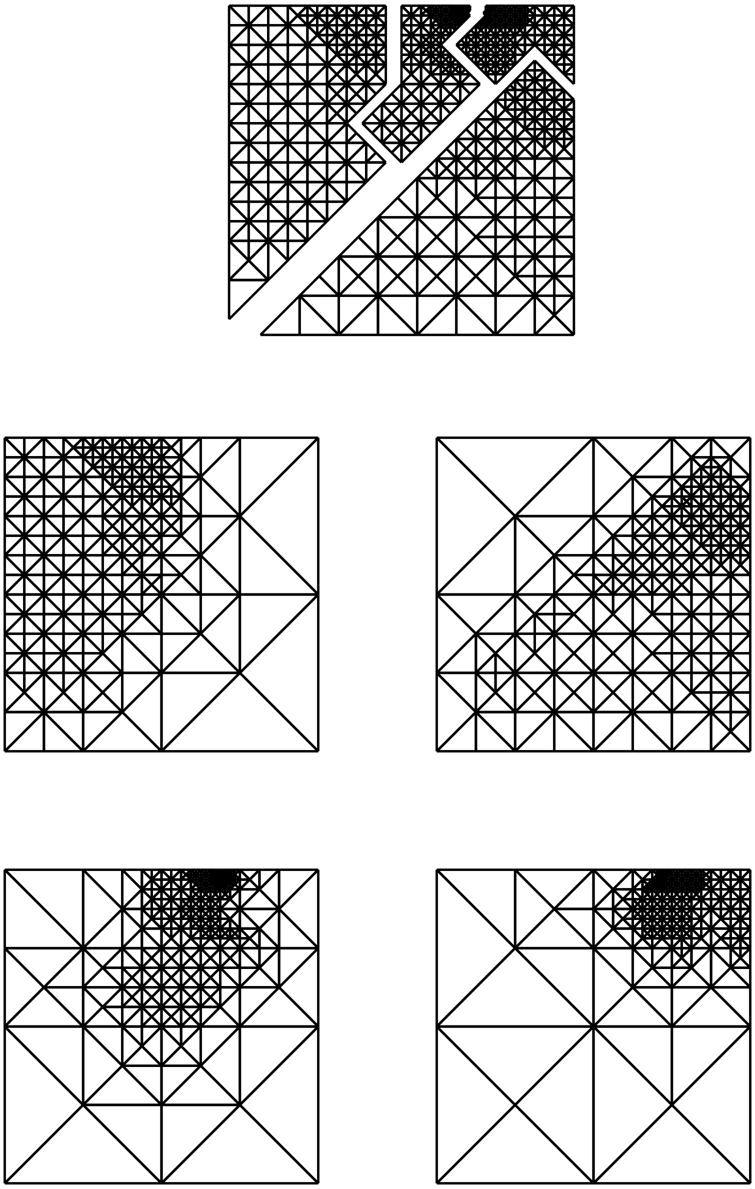

FIG. 1. *An adaptive grid partitioned for four processors and the corresponding FuDoP grids.*

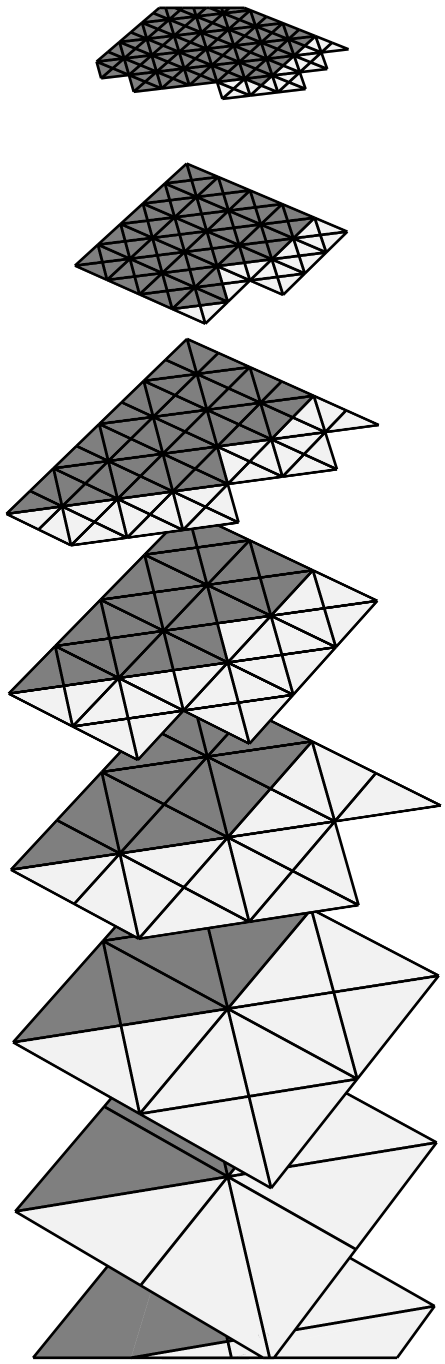

Fig. 2. *The individual levels of a FuDoP grid.*

Since with the FuDoP distribution each processor contains a compatible grid covering the entire domain, each processor can perform a refinement of that grid independent of the other processors without concern for conflict along the partition boundaries. To create the FuDoP distribution of the grid, each processor examines only the triangles in its partition when selecting triangles that have large error indicators. Outside its partition, triangles are only refined for compatibility and overlap constraints. Communication to synchronize the grids occurs after the refinement is complete.

For illustration it is assumed that the adaptive refinement strategy is the newest node bisection algorithm described in [9], however the general approach should be applicable to almost any adaptive refinement strategy. It is also loosely assumed that the grid partitioning algorithm is the refinement tree recursive bisection method described in [12]. One property of this method is used, namely that the owner of a triangle will also be the owner of the children of the triangle, i.e., that ownership is assigned to subtrees of the refinement tree. The parallel adaptive refinement algorithm should be applicable under any partitioning algorithm that satisfies this property or provides a communication-free means of immediately determining the owners of the children of a triangle when it is refined.

The parallel adaptive refinement algorithm executed on each processor is as follows:

```
determine global maximum error indicator over all processors
repeat
  mark all triangles that I own and have a sufficiently
    large error indicator
  mark triangles needed for compatibility
  refine marked triangles
until all error indicators are less than some factor times
  the prior global maximum error indicator
synchronize the grid with other processors
```

In this algorithm the grid is refined until the maximum error indicator has been reduced by some given factor, for example 0.5. This is one of several possibilities for determining how much refinement to perform before continuing with the next step of the adaptive multilevel algorithm of section 1. This choice is not fundamental for the algorithm, but the method chosen must be such that it does not require communication inside the loop. For example, a common choice is to increase the number of vertices by some factor. This must be the total number of vertices, not the number of vertices on each processor. Checking this condition would require a global sum each time through the loop, which requires communication. With the maximum error estimate condition, a global maximum occurs once before the loop; after that the condition is local to each processor.

The first main difference between the sequential algorithm and the parallel algorithm is the selection of triangles to be refined. In the sequential algorithm, all triangles with a sufficiently large error indicator are refined. In the parallel algorithm, this is restricted to triangles that are owned by the processor. Triangles that are owned by other processors are refined only if needed to maintain compatibility of the grid. This generates the FuDoP distribution of the grid. Other than this modification, the refinement loop is essentially the same as in the sequential algorithm.

The other main difference is the addition of the synchronization step after refinement. Some refinement can occur outside the processor's partition, creating triangles in its grid that are owned by other processors. The processor will have to inform the owners of those triangles that it now has a copy of the triangles and associated vertices and equations. Depending on the implementation of the linear system solver, it may also need to have a list of other processors that have a copy. In addition, the generation of the overlap for FuDoP with extended overlap occurs during this synchronization.

Each processor sends a message to each other processor containing three parts:

1. a list of triangles that this processor refined and the other processor owns,
2. a list of triangles that this processor owns and the other processor needs for overlap, and
3. the "who has a copy" information for triangles that this processor owns and the other processor has a copy, if it has changed.

Messages of this form will have to be sent three times to fully synchronize the processors. The first message will only contain the first two parts, since a processor does not yet know of any changes in the copies of the triangles it owns. When a processor processes the first part of the list, it is possible that it will find one or more triangles that it has not yet created, and it will perform the refinements necessary to create those. These refinements may cause additional compatibility refinements in partitions owned by other processors causing the need for the first part of the message a second time, and may create new triangles on the partition boundaries that will be needed by other processors for overlap causing the need for the second part of the message a second time. This need for additional refinements can occur only once, since any triangles in the first part of the second message will have already been in the first part of the first message from a different processor. Thus the second message can contain all three parts, and the third message will contain only the third part and will complete the synchronization.

Three additions to the data structures are required to maintain additional information for the parallel algorithm, but most of the data structure is identical to that used in a sequential algorithm.

First, each processor needs to know who the owner of each triangle is. This can be stored in an integer.

Second, the linear system solver may require that each processor has a list of all processors that have a copy of a triangle, vertex and/or equation. The least memory intensive data structure for this information is to use a bit field, where the i^{th} bit is "1" if the i^{th} processor has a copy and "0" otherwise.

Finally, a global numbering scheme is needed for the processors to communicate about particular elements. A hash table [6] approach is used for this; hash tables have been used in this context by Stals [17] and Griebel[4]. Each triangle is given a unique identifier, or hash key, based on its position in the refinement tree. In particular, when a triangle with hash key k is bisected, the two children are given hash keys $2k$ and $2k+1$, with the child containing the oldest vertex receiving the smaller key. This assigns a unique hash key to every triangle even before the triangles exist. Processors can now communicate about triangles whether or not they have a copy. When a vertex is created it is associated in a deterministic way with one of the triangles created at the same time, and is given the hash key of that triangle. For linear finite elements, equations are uniquely associated with vertices and receive the same hash key as the vertex. A hash function and hash table is then used to relate the hash key to the local index on the processor.

4. Numerical results. Numerical computations were performed to examine the speedup obtained by parallel computing. Each processor contained exactly one "virtual processor" in these computations. Tables 1 and 2 present these results.

These results were obtained with a cluster of four 200MHz Pentium Pro-based computers with 128 Mbytes of memory, connected by fast ethernet (100 BaseT). NASoftware 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.

TABLE 1
Speedup results for problem 1.

Processors	Time(s.)	Speedup	Efficiency
1	10.49	-	-
2	5.92	1.77	.86
4	2.70	3.89	.97

Table 1 presents the results for the first example problem. Laplace's equation is solved on the unit square with Dirichlet boundary conditions set such that the solution is

$$1.1786 - 0.1801p + .006q$$

where

$$p(x,y) = x^4 - 6x^2y^2 + y^4$$

$$q(x,y) = x^8 - 28x^6y^2 + 70x^4y^4 - 28x^2y^6 + y^8$$

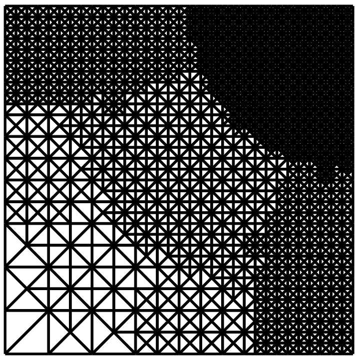

FIG. 3. *An adaptive grid generated by Problem 1.*

This is Problem 35 from the ELLPACK PDE population [14]. An example grid for this problem is shown in Figure 3. To avoid excessive communication on extremely coarse grids, the grid is kept on one processor until the refinement has produced 1000 vertices, at which point the grid is partitioned and distributed over the processors. The program is terminated when the grid contains approximately 20,000 vertices. The wall clock time of the refinement portion of the program is reported in Table 1. This includes all of the time used for refinement including the initial refinement on one processor and communication during synchronization after refinement. 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 2
Speedup results for problem 2.

Processors	Time(s.)	Speedup	Efficiency
1	9.94	-	-
2	4.96	2.00	1.00
4	2.69	3.69	.92

Table 2 presents the results for a second example problem. Here the boundary conditions for Laplace's equation are piecewise linear across the top boundary, with a discontinuity in the derivative at $x = 0.7$, and zero on the other three sides. This problem generates the adaptive grid of Figure 1. For this problem, the final grid contains approximately 16,000 vertices.

These results show that the parallel adaptive refinement method can achieve parallel efficiencies of 85% to near 100% in this environment for moderate sized problems.

5. Conclusion. A new approach to parallel adaptive refinement in the context of adaptive multilevel methods was presented. This approach uses the Full Domain Partition distribution to eliminate communication during the refinement process. The communication consists of a global reduction operation before refinement and three all-to-all communication steps after the refinement is complete. Numerical results illustrate parallel efficiencies in excess of 85% for 2 and 4 processors on moderate sized problems.

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*, volume 15 of *Frontiers in Applied Mathematics*, 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] M. GRIEBEL AND G. ZUMBUSCH, *Hash-Storage Techniques for Parallel Adaptive Multilevel Solvers and Their Domain Decomposition Parallelization*, Proceedings of Domain Decomposition 10, Contemporary Mathematics 218, AMS, 1998, pp. 279–286.

[5] M.T. JONES AND P.E. PLASSMANN, *Parallel Algorithms for Adaptive Mesh Refinement*, SIAM J. Sci. Comp., 18 (1997), pp. 686–708.

[6] D.E. KNUTH, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.

[7] S.F. MCCORMICK, *Multilevel Adaptive Methods for Partial Differential Equations*, volume 6 of *Frontiers in Applied Mathematics*, SIAM, Philadelphia, 1989.

[8] W.F. MITCHELL, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Trans. Math. Soft., 15 (1989), pp. 326–347.

[9] W.F. MITCHELL, *Optimal multilevel iterative methods for adaptive grids*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 146–167.

[10] W.F. MITCHELL, *The full domain partition approach to distributing adaptive grids*, Appl. Numer. Math., 26 (1997), pp. 265–275.

[11] W.F. MITCHELL, *A Parallel Multigrid Method Using the Full Domain Partition*, Electronic Transactions on Numerical Analysis, 6 (1998), pp. 224-233.

[12] W.F. MITCHELL, *The Refinement-Tree Partition for Parallel Solution of Partial Differential Equations*, NIST Journal of Research, 103 (1998).

[13] C. Özturan, H.L. deCougny, M.S. Shephard and J.E. Flaherty, Parallel Adaptive Mesh Refinement and Redistribution on Distributed Memory Computers, Comput. Methods Appl. Mech. Engrg., 119 (1994), pp. 123–137.

[14] J.R. Rice and R.F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[15] M.-C. Rivara, *Design and data structure of fully adaptive, multigrid, finite-element software*, ACM Trans. Math. Soft., 10 (1984), pp. 242–264.

[16] U. Rüde, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, volume 13 of *Frontiers in Applied Mathematics*, SIAM, Philadelphia, 1993.

[17] L. Stals, Parallel Multigrid On Unstructured Grids Using Adaptive Finite Element Methods, PhD thesis, Department of Mathematics, Australian National University, Canberra, 0200, Australia, 1995.