The Full Domain Partition Approach to Distributing Adaptive Grids

William F. Mitchell [1]

Applied and Computational Mathematics Division, National Institute of Standards and Technology, Gaithersburg, MD 20899 USA

Abstract

Adaptive refinement has been shown to be an effective means of speeding up the solution of partial differential equations. Adaptive multilevel methods combine multigrid iteration with adaptive refinement to produce fast $O(N)$ solutions on sequential computers. However, many problems of interest require parallel computers. While adaptive refinement, multigrid and parallel computers are individually achieving wide spread use in applications, the combination of all three is still a research topic. In this paper we describe the full domain partition (FuDoP) distribution of adaptively refined grids over a distributed memory parallel computer. In the FuDoP approach, each processor has a compatible grid that covers the full domain, but with refinement confined to the processor's partition of the total grid. Outside the partition area, refinement is limited to that required for compatibility and leads to a small number of extra grid elements. With the FuDoP distribution, a parallel multigrid method can maintain a multigrid rate of convergence while sending only two messages per V-cycle. FuDoP also accommodates parallel implementation of adaptive refinement and partitioning.

1 Introduction

The numerical solution of partial differential equations (PDEs) is the most computationally intense part of solving mathematical models with 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 [1,2,5,8,10,11]. 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 a parallel multilevel adaptive method is in the early research phase.

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

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.

The crux of this paper is to present a new approach to determining how much shadow information is provided to each processor. This is motivated by the difficulty in designing a parallel multigrid algorithm that maintains the optimal order of convergence without requiring so much communication that the communication time dominates the computation time. 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 approach presented in this paper, 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. We refer to this as the *full domain partition* (FuDoP) because the partition is extended to cover the full domain.

In Section 2 we define the full domain partition. Section 3 analyzes the additional work on each processor, and potential parallel speedup. In Section 4 we examine the modifications that FuDoP imposes on the refinement, multigrid and partitioning algorithms.

2 Full Domain Partition

Multilevel adaptive methods begin with a very coarse grid (possibly just enough elements to define the domain) and alternately perform phases of adaptive grid refinement and multigrid cycles resulting in a full multigrid method that solves the PDE in time proportional to the number of vertices in the final grid. In the parallel setting we must be concerned with balancing the work load of the processors. Since the adaptivity is likely to cause more refinement to occur in some partitions than others, the partitions must be modified after each refinement phase, and data redistributed among the processors to reflect the new partitions. (In actual implementations, one would not blindly perform this repartitioning every time, but would check the status of the load balance and repartition if imbalance is above some threshold.) The repartitioning can be done by a local improvement approach [4,12] or by applying a global partitioning algorithm. If the later approach is taken, it is important that the partitioning algorithm be very fast and that the new partition is similar to the old partition [9]. The high level form of the parallel multilevel adaptive method is thus:

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

The problem that motivates FuDoP is that of developing an effective parallel multigrid algorithm in the computational environment previously described. In parallel environments without low-latency/high-bandwidth communication channels, it might not be possible to simultaneously achieve (1) optimal order convergence rates, and (2) non-dominating communication costs, if the only data communicated between processors is from the boundaries of the partitions. Consider the following two approaches, which represent the extremes in frequency of communication.

Approach 1: Communicate whenever necessary. In this approach, the boundary data is passed whenever it has changed and the receiving processor needs it for computations. In this case the parallel multigrid is mathematically equivalent to the sequential multigrid, so the optimal convergence rate is guaranteed. But this requires many communication steps, at least once on each level during the multigrid cycle, which is typically $O(\log N)$. The amount of computation on each level (and hence between messages) is quite small, maybe on the order of $10N$ additions and multiplications, which leads to a dominance of the communication time, in particular the latency of starting a new message.

Approach 2: Communicate after each multigrid cycle. In this approach, each processor performs a multigrid cycle on its partition without any communication, and receives the updated shadow values after each cycle. Here the communication is kept to a minimum, just one message per multigrid cycle. However, the multigrid algorithm is no longer intact. Now we are effectively holding the boundary data fixed while solving the problem on each subdomain, exchanging boundary data, and repeating until convergence. This is the classical Schwarz Domain Decomposition method which is known to require many iterations to converge.

The FuDoP philosophy is to pass more information than just the boundary data. With this approach it is possible to design a multigrid algorithm that varies only slightly from the sequential algorithm, keeps optimal order convergence rates, and passes only two messages per multigrid cycle [6].

With FuDoP each processor contains its partition of the grid, plus the minimum number of extra triangles from the global refinement process to cover the full domain. The resulting local grid is as if the processor performed adaptive refinement with the error indicators set to zero outside its partition of the domain. Triangles outside the partition are refined only to maintain compatibility.

Some data elements (triangles, vertices or equations) will be replicated on several processors, but each has one processor designated as the owner. This processor has the definitive values for the element when there is a discrepancy between the values on different processors.

Figure 1 illustrates FuDoP for an adaptive grid partitioned over four processors. Part (a) shows the complete grid and the partition of the physical domain indicated by grey levels. Parts (b)–(e) show the grids contained on each processor, with the "owned" elements drawn darker.

3 Analysis

It is reasonable to wonder what the cost of the full domain partition is, i.e., how much extra work is involved in having the extra grid elements. To examine this, it suffices to determine the number of extra triangles in the grid. This is because the amount of computational work depends on the number of triangles and vertices in the grid, the amount of communication depends on the number of extra triangles and vertices in the grid, and the number of vertices is proportional to the number of triangles.

Let

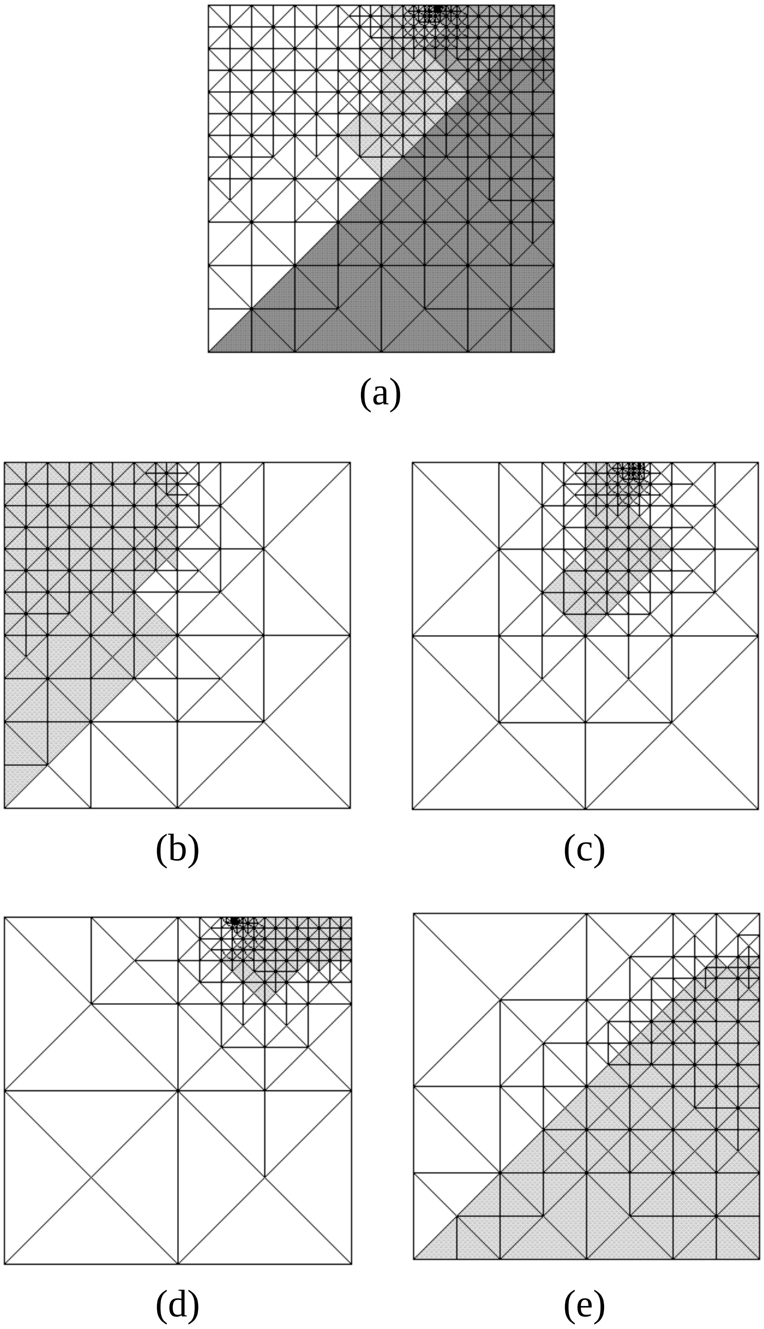

Fig. 1. (a) a partitioned adaptive grid, (b)–(e) the corresponding FuDoP grids.

N = number of triangles in full grid
M = number of triangles owned by a processor
X = number of extra triangles to cover the domain
L = number of refinement levels
P = number of processors (or partitions)

We are interested in the size of X relative to the size of M. It would be very difficult, if at all possible, to prove a relationship between them in the very

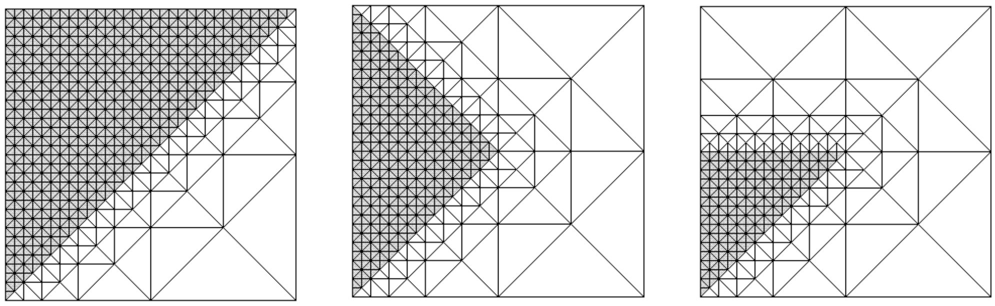

Fig. 2. FuDoP grids for 2, 4 and 8 partitions of a uniform grid.

general setting of arbitrary adaptive grids and arbitrary partitioning schemes, but in the case of "reasonable" partitioning schemes where the boundary of the partitions is not too wild, we put forth the following conjecture.

Conjecture 1 $X = O(\sqrt{M})$

We supply supporting evidence for the conjecture in three forms: a heursitic argument, exact results for a uniform grid, and a numerical example from an adaptive grid.

3.1 Heuristic

One would expect the one dimensional boundary of a two dimensional region containing M points to contain $O(\sqrt{M})$ points, and thus there would be $O(\sqrt{M})$ triangles just outside the boundary. As we move away from the boundary, the number of triangles decreases exponentially (see Figure 1 or 2), being approximately cut in half at each new layer. The total number of extra triangles is thus still $O(\sqrt{M})$.

3.2 Uniform grids

With uniform grids we can determine the exact value of X, given the method for generating the grid and the method of partitioning the grid. Consider the case of the unit square initially triangulated with two triangles, uniform refinement by the newest node bisection method [7], and partitioning by the refinement tree based partition method [9]. Figure 2 shows the FuDoP grid of one processor in the cases where the number of processors is 2, 4 and 8. It is easy to determine the pattern of the extra triangles and count them. For example, in the case of two partitions:

$$N = 2^L$$

Table 1
Number of extra triangles for an example nonuniform grid.

2 processors				4 processors				8 processors			
N	X	M	$X/\sqrt{M}$	N	X	M	$X/\sqrt{M}$	N	X	M	$X/\sqrt{M}$
1538	132	769	4.74	1187	112	297	6.49	946	100	118	9.23
3185	195	1593	4.87	2654	166	664	6.45	2124	161	266	9.87
6373	287	3187	5.08	5866	248	1467	6.47	4599	220	575	9.16
11164	388	5582	5.19	11133	357	2783	6.77	8893	328	1112	9.84
24644	575	12322	5.18	25143	524	6286	6.61	15261	412	1908	9.44
46679	816	23340	5.34	45585	726	11396	6.80	39755	714	4969	10.12

$$
\begin{aligned}
M &= 2^{L-1} \; = \; \frac{1}{P} 2^L \\
X &= 4 \sum_{i=1}^{\lfloor \frac{L}{2} \rfloor - 1} (2^i - 1) \; + \; 2 \\
&= 4 \cdot 2^{\lfloor \frac{L}{2} \rfloor} \; + \; O(L)
\end{aligned}
$$

In general we find

$$
\begin{aligned}
N &= 2^L \\
M &= N/P \; = \; 2^L/P \\
X &= \begin{cases} 4 \cdot 2^{\lfloor \frac{L}{2} \rfloor} \; + \; O(L) & P = 2,\ 4,\ 8 \\ \frac{4}{\sqrt{P}} \cdot 4 \cdot 2^{\lfloor \frac{L}{2} \rfloor} \; + \; O(L) & P = 16,\ 64,\ 256, \ldots \\ \frac{3\sqrt{2}}{\sqrt{P}} \cdot 4 \cdot 2^{\lfloor \frac{L}{2} \rfloor} \; + \; O(L) & P = 32,\ 128,\ 512, \ldots \end{cases}
\end{aligned}
$$

Thus we have

$$
X \; = \; O\left(\frac{2^{\lfloor \frac{L}{2} \rfloor}}{\sqrt{P}}\right) \; = \; O\left(\sqrt{\frac{2^L}{P}}\right) \; = \; O(\sqrt{M})
$$

3.3 Nonuniform Grid

Finally, in Table 1 we present the results for a family of nonuniform grids similar to that in Figure 1. Here we see that, as N increases, the value of $X/\sqrt{M}$ in nearly constant, supporting the conjecture that $X = O(\sqrt{M})$. We also observe the same $\sqrt{P}$ growth that was found in the uniform case for $P = 2,\ 4,$ and 8. We anticipate that, like the uniform case, this dependence on P will disappear for larger values of P.

3.4 Speedup

The amount of speedup that can be obtained by using P processors with the full domain partition depends on the algorithm that is being executed on the partitions. We consider the speedup that can be obtained with the following assumptions on the algorithm (which are met by the adaptive refinement and multigrid algorithms in Section 4):

(i) The amount of data communicated is $O(X)$.
(ii) The number of messages is $O(1)$.
(iii) The amount of computation is proportional to the number of grid points, i.e., $O(N)$ on a uniprocessor and $O(M+X)$ on each parallel processor.

Under these assumptions the communication time is dominated by the computation time for sufficiently large M, so the time is determined by the amount of computation. We then have

$$\begin{aligned}
\text{speedup} &= \frac{\text{time on uniprocessor}}{\text{time on } P \text{ processors}} \\
&= \frac{c_1 N}{c_1(M+X)} \\
&= \frac{N}{M + c_2\sqrt{M}} \\
&= \frac{N/P}{N/P \; + \; c_2\sqrt{N/P}} P \\
&= \frac{\sqrt{N/P}}{\sqrt{N/P} \; + \; c_2} P
\end{aligned}$$

This indicates that

(i) For a fixed number of processors P, as the problem size, N, increases, the speedup approaches P.
(ii) For a fixed problem size on each processor, N/P, the speedup scales with the number of processors.

4 Parallel Algorithms with FuDoP

In this section we present the modifications that are applied to several sequential algorithms to convert them to parallel algorithms with the full domain partition. The algorithms presented are specific forms of adaptive refinement, multigrid and grid partitioning.

4.1 Adaptive Refinement

In this section we examine the modifications that would be made to a sequential adaptive refinement algorithm to convert it to a parallel algorithm with the grid distributed by the full domain partition.

We consider adaptive refinement algorithms of the form:

```
repeat
  select triangles with large error indicators
  refine them and others needed for compatibility
until enough refinement (e.g., double vertices)
```

The modified version of this algorithm (with modifications shown in italics) is:

```
repeat
  select triangles with large error indicators that are owned by
    this processor
  refine them and others needed for compatibility
until enough refinement (e.g., halve error indicator)
reconcile differences between processors
```

The differences are:

(i) Only refine triangles that are owned by the processor or needed for compatibility.
(ii) The method for determining when to quit may need to be modified to one that does not require communication between the processors.
(iii) The processors must communicate after completing refinement (but not before then) to reconcile any differences in the grids they generated.

The following reconciliation process insures that the owner of a triangle refines it if any other processor refined it, and that all processors have current global indexing information for all triangles they have a copy of.

```
repeat
  for any triangle I refined but don't own, inform the owner that
    I refined it
  refine any unrefined triangles that I own and someone else
    refined
```

```
  send new index information
  receive and update index information
until no one refines a triangle
```

The process might repeat because the refinement in the second step may cause additional unowned triangles to be refined, but it is not expected that it would repeat more than two or three times.

The grid that results from this parallel algorithm will differ slightly from that produced by the sequential algorithm only because of the difference in the termination criteria.

4.2 Multigrid

In this section we examine the modification to a standard V-cycle sequential multigrid algorithm when it is applied in parallel with the full domain partition grids. In particular, we consider the hierarchical multigrid method of [8] where relaxation between restrictions is performed only on fine grid points, and relaxation between prolongations is performed at fine grid and neighboring coarse grid points. The V-cycle is given by:

```
for level = nlev downto 2
  relaxation
  restriction
end for
solve on coarsest grid
for level = 2 to nlev
  prolongation
  relaxation
end for
```

In parallel this becomes

```
for level = nlev downto 2
  relaxation
  restriction (record modification to right hand side)
end for
send new solution vector and right hand side modifications to shadow copies
solve on coarsest grid
for level = 2 to nlev
  prolongation
  relaxation
end for
```

send new solution vector to shadow copies

The communication of the solution vector and right hand side to the shadow copies on other processors provides all the information required for the multigrid operations. Some of the values will be slightly out of date because the communication only occurs at the top and bottom of the cycle. This slows the convergence rate slightly, but multigrid rates of convergence are still maintained. Further details and analysis of multigrid with the full domain partition are presented in [6]. A similar approach for finite differences on uniform grids is introduced in [3].

4.3 Partitioning

There are many algorithms for partitioning a nonuniform grid for distribution over the processors. We consider here the refinement tree based partitioning algorithm in [9]. In this algorithm, the refinement tree that represents the adaptive refinement process by newest node bisection is used for partitioning the grid. Each node, which corresponds to some triangle formed during the refinement process, is weighted by the number of leaf triangles below it, which correspond to the final triangles that are descendents. A path is found from the root of the tree to a leaf such that the path bisects the tree into two parts containing an equal number of leaves, i.e., divides the triangles into two equally sized connects sets. The algorithm is applied to the subtrees to obtain additional partitions (recursive bisection). The algorithm is:

```
compute weight of each tree vertex
vertex ← root
repeat
  determine potential assignment of children to sets
  compute potential size of sets
  make assignment of child to smaller set
  vertex ← other child
until vertex is a leaf
repeat to get 4, 8, ..., P sets
```

With the full domain partition, each processor applies this algorithm to the refinement tree of its grid, including the extra triangles. The only difference is in the computation of the weights. Since a processor may not have the full subtree below a triangle that it does not own, it computes the weights only for those triangles that it owns. It then obtains the weights for the (locally) leaf triangles from their owners, and finishes computing the weights above them. The resulting partitions are identical to those produced by the sequential version of the algorithm.

5 Conclusion

The full domain partition is a new approach to distributing a grid over parallel processors with distributed memory. The grid is partitioned as usual. The grid on each processor consists of not only the partition, but also the minimum number of additional triangles to cover the full domain. The additional triangles are taken from those that occurred during the refinement processes. The number of additional triangles is $O(\sqrt{M})$ where M is the number of triangles in the partition, which is small enough to allow speedups approaching P as $N \to \infty$, and scalability when N/P is fixed and $N, P \to \infty$. Parallel algorithms for adaptive refinement, multigrid and partitioning with full domain partitions are easily derived as modifications of the sequential versions of the algorithms.

References

[1] Randolph E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*, volume 15 of *Frontiers in Applied Mathematics.* SIAM, Philadelphia, 1994.

[2] Achi Brandt. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.

[3] Achi Brandt and Boris Diskin. Multigrid solvers on decomposed domains. In A. Quarteroni, editor, *Proceedings of the Sixth International Conference on Domain Decomposition Methods*, pages 135–155, Providence, 1994. AMS.

[4] Karen D. Devine and Joseph E. Flaherty. Dynamic load balancing for parallel finite element methods with adaptive h- and p-refinement. In *Proceedings of the 7^{th} SIAM Conference on Parallel Processing for Scientific Computing*, pages 593–598, Philadelphia, 1995. SIAM.

[5] Stephen F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*, volume 6 of *Frontiers in Applied Mathematics.* SIAM, Philadelphia, 1989.

[6] William F. Mitchell. Parallel multigrid with full domain partitions. in preparation.

[7] William F. Mitchell. Adaptive refinement for arbitrary finite element spaces with hierarchical bases. *Journal of Computational and Applied Mathematics*, 36:65–78, 1991.

[8] William F. Mitchell. Optimal multilevel iterative methods for adaptive grids. *SIAM Journal of Scientific and Statistical Computing*, 13:146–167, 1992.

[9] William F. Mitchell. Refinement tree based partitioning for adaptive grids. In *Proceedings of the 7^{th} SIAM Conference on Parallel Processing for Scientific Computing*, pages 587–592, Philadelphia, 1995. SIAM.

[10] Maria-Cecelia Rivara. Design and data structure of fully adaptive, multigrid, finite-element software. *ACM Transactions on Mathematical Software*, 10:242–264, 1984.

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

[12] Chris Walshaw and M. Berzins. Dynamic load-balancing for pde solvers on adaptive unstructured meshes. *Concurrency: Practice & Experience*, 7(1):17–28, 1995.