Adaptive Grid Refinement and Multigrid on Cluster Computers*

William F. Mitchell
Mathematical and Computational Sciences Division
National Institute of Standards and Technology
Gaithersburg, MD.
william.mitchell@nist.gov

Abstract

It has been shown that the combination of adaptive grid refinement and multigrid solution, known as adaptive multilevel methods, provide effective methods for solving partial differential equations on sequential computers. Recently, research has been performed on parallelizing these procedures. Effective parallelization is difficult because of the irregular nature of both adaptively refined grids and the multigrid process. This is particularly true on cluster computers which have slow communication channels that require algorithms with infrequent communication. An approach to parallelizing adaptive multilevel methods with few communication steps is presented. Numerical results on an 8-processor PC cluster demonstrate 60-90% efficiency.

1. Introduction

The numerical solution of partial differential equations (PDEs) is the most compute-intensive part of most scientific computer simulations, with applications in all fields of science and engineering. Consequently, improving the methods for solving PDEs has been the focus of much research in numerical methods for several decades. In the last two decades it has been shown that the combination of adaptive grid refinement and multigrid solution, known as adaptive multilevel methods, provide effective methods on sequential computers [2, 4, 6]. Adaptive grid refinement reduces the problem size by focusing the effort to the regions where higher resolution is needed, and multigrid is the fastest solution method for problems that can be formulated with a grid hierarchy. In the last decade, research has been performed on parallelizing these procedures. Effective parallelization is difficult because of the irregular nature of both adaptively refined grids and the multigrid process.

In the last few years, cluster computers have begun to emerge as a viable platform for scientific computing. While traditional expensive massively parallel processors (MPPs) continue to thrive in the large laboratories, organizations with a more modest budget are finding that a cluster of PCs can provide a small-to-medium scale parallel supercomputer at the same price as a traditional workstation. Indeed, clusters of PCs are no longer just a curiosity for computer scientists, but are now being used for practical computations by scientists and engineers.

For algorithm design, the primary difference between an MPP and a cluster computer is the speed of communication. A large portion of the expense of an MPP is the specially designed high-performance communication network between the processors. A cluster computer uses off-the-shelf networking technology which is much slower, both in bandwidth and, more importantly, start-up latency. Consequently, algorithms for cluster computers should be designed to perform interprocessor communication less often than algorithms that were designed for MPPs. So-called "embarrassingly parallel" algorithms are a good match for cluster computers, but are not available for all applications. For applications such as solving a PDE, we need coarse-grain algorithms that communicate infrequently and perform much computation between communication steps.

In this paper we present a coarse-grain algorithm for the solution of elliptic PDEs by adaptive multilevel methods. The algorithm uses only two communication steps during an adaptive refinement phase, two during load balancing, and two during each multigrid cycle. This is in contrast to traditional parallel algorithms that communication much more frequently. For example, a traditional parallel multigrid algorithm would communicate at least twice on each grid level during a cycle.

The high-level parallel adaptive multilevel algorithm is given in Fig. 1. It is a full multigrid method consisting of alternately refining the grid and solving the equations on the new grid until it is determined that the grid is fine enough to give a sufficiently accurate solution. In the parallel version, one must also perform load balancing since different

```
start with very coarse mesh
repeat
   adaptive grid refinement
   load balance
   multigrid cycles
until termination crite-
rion is met
```

Figure 1. Parallel adaptive multilevel algorithm.

processors are likely to perform different amounts of refinement, which would result in an unbalanced load during the multigrid cycles. A variant of the algorithm in Fig. 1 is to use predictive load balancing in which load balancing occurs before adaptive refinement.

2. The full domain partition

The key to the reduced communication in the parallel adaptive multilevel algorithm is a new form of data distribution known as the full domain partition, first introduced in [7]. In this distribution, the grid is partitioned and distributed to the processors as usual. But in addition, each processor contains shadow copies of enough additional elements to cover the full domain. These elements come from coarser levels of the adaptive grid hierarchy. Figure 2 illustrates an adaptive grid partitioned for two processors and the full domain partition that each processor contains. The colors indicate the assignment of elements to processors. Shadow elements are shown in grey.

The number of additional elements decreases exponentially as one moves away from the partition boundary, resulting in the total number of additional elements being proportional to the square root of the number of elements in the partition (in 2D), the same as the traditional approach of adding shadow copies of adjacent elements. The consequence of the full domain partition is that each processor contains a grid that looks like an adaptively refined grid for the whole problem, and can perform essentially the sequential adaptive multilevel algorithm with occasional communication to keep the solutions synchronized. A modification of this technique to eliminate all communication until the final grid is being developed by Bank and Holst [1], but in this paper we consider an approach that attempts to produce nearly the same results as the sequential algorithm.

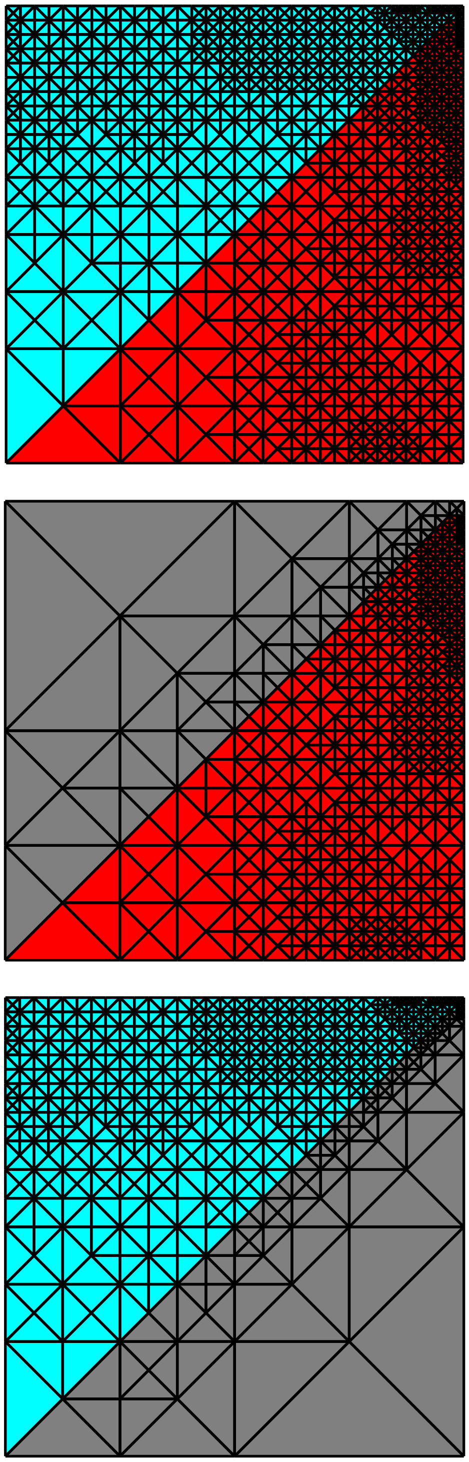

Figure 2. An adaptive grid partitioned for two processors, and the two full domain partitions.

3. Adaptive refinement

The full domain partition provides an easy means of parallelizing an adaptive refinement algorithm. Additional details of this approach can be found in [8]. Adaptive refinement can be performed by each processor in parallel without communication until the very end. The algorithm is identical to the sequential algorithm except that only the elements in the partition owned by the processor, and those needed for compatibility of the grid, are candidates for refinement. This can be achieved by setting the refinement error indicator to zero outside the partition. The parallel refinement algorithm actually creates the full domain partition. A communication step at the end informs other processors of any elements that were refined outside the partition to insure that the owner of that element also refines it. A second communication step enforces an overlap requirement needed for fast convergence of the parallel multigrid algorithm (see Section 5).

4. Load balancing

Load balancing consists of two phases: partitioning the grid, and redistributing the data among the processors. In the first phase the grid is partitioned into equal sized pieces for assignment to the processors. There are several fast partitioning algorithms that are appropriate to use in an adaptive multilevel setting [3]. The refinement-tree based partitioning algorithm RTK [10] produces contiguous partitions with only one communication step. The refinement tree is a representation of the refinement processes that produced an adaptive grid from an initial grid. The nodes of the tree correspond to the grid elements that existed at some point during the refinement process. The children of a node correspond to the grid elements that were created when the corresponding element was refined. The nodes are weighted to reflect the work load associated with an element. For example, using a weight of 1.0 on the leaf elements and 0.0 on the interior elements will create partitions that have an equal number of elements from the final grid. For predictive load balancing, the error indicator of the corresponding element is a good weight for leaf elements, with 0.0 on the interior elements. In the first phase of RTK, a tree traversal is performed to sum the weights below each node. This requires one communication step to pass partial sums for the parts of the tree that have been pruned on other processors. In the second phase, the partitions are defined by another tree traversal with no communication. The desired size of each partition is determined by dividing the summed weight at the root node by the number of partitions. During the traversal, the summed weight at the nodes are examined relative to the size of the partition under construction. If it is small enough to be added to the partition without exceeding the desired size, then it is added and the subtree is not traversed. Otherwise, the children are visited and the subtree will be split among two or more partitions. Further details can be found in [10].

Table 1. Contraction factors for parallel multigrid.

Partitions	without extra overlap	with extra overlap
1	.095	.095
2	.179	.095
4	.244	.096
8	.215	.098
16	.240	.100
32	.224	.104
64	.254	.104

The second phase is to redistribute the data according to the new partitions. This uses one communication step to pass data from the old owner of an element to the new owner.

5. Multigrid

The parallel multigrid algorithm is defined in detail in [9]. On each processor, it is essentially the same as the sequential multigrid algorithm on each processor, with the addition of two communication steps. Since the full domain partition provides each processor with a compatible 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 grids do not contain sufficient resolution over the entire domain. The key difference between the sequential and parallel versions is the computation of the residual injection in the right hand side. The parts of the residual due to nodes that are not in a particular processor's grid cannot be computed by that processor. Instead, the processors exchange partial residuals when the coarsest grid is reached, i.e., after half of a V-cycle. To each shadow copy of the nodes a processor owns, it sends the current solution value and part of the residual due to the other nodes that it owns. The second half of the V-cycle is then performed, and solution values for shadow nodes are exchanged in a second communication step.

Because data at the shadow nodes lags slightly behind, the computation is not identical to the sequential algorithm. However, numerical experiments with a model Laplace problem indicate that the multigrid rate of convergence is degraded by an insignificant amount if there is sufficient overlap at the boundaries of the partitions. Table 1 shows the convergence rate of the parallel multigrid algorithm for Laplace's equation on the unit square discretized by the finite element method with linear elements and a triangulation of approximately 64,000 vertices. The first column is the number of partitions in the grid (or number of processors). The second column gives the contraction factor (the factor by which a V-cycle reduces the energy norm of the error) for the multigrid algorithm of [9] with the full domain partition without extra overlap. One can see that, while the contraction factors are still small, they degrade significantly as the number of partitions is increased. The third column gives the contraction factors when one also guarantees that all immediate neighbors of the nodes in the partition are included in the shadow nodes. Here the rate of convergence remains nearly constant as the number of partitions is increased.

6. Numerical results

The methods described in this paper were implemented in Fortran 90 in the program PHAML (Parallel Hierarchical Adaptive MultiLevel). This is a finite element program using piecewise linear basis functions over triangular elements. Adaptive refinement is by newest node bisection [5] using the hierarchical coefficient error indicator. The multigrid method is based on a hierarchical basis formulation [6]. Predictive load balancing is used for the results presented in this section.

The computations were performed on a cluster of eight 400 MHz Pentium II based [1] PCs connected by 100 BaseT ethernet switch and operating under Linux 2.2.16. Programs were compiled with Lahey/Fujitsu LF95 5.5 using "-O" for optimization. Message passing was performed with the LAM 6.3.1 implementation of MPI.

To demonstrate the parallel efficiency that can be obtained by the approach described in this paper, Poisson's equation was solved on the unit square with the right hand side and Dirichlet boundary conditions set such that the exact solution is $x^6 + y^6$, which has a mild boundary layer along the top and right side of the domain. An example grid with approximately 2000 vertices is shown in Figure 3. The colors indicate the partitioning of this grid for four processors.

The equation was solved using 1, 2, 4 and 8 processors. The problem size was scaled with the number of processors so that in all cases the final grid has approximately 65,000 vertices per processor. Ideally, execution time would remain constant as the number of processors and problem size change. The "wall clock" execution times, measured by the Fortran 90 intrinsic SYSTEM_CLOCK subroutine, are given in Table 2. In addition to the total time for the program, the time for the major phases of the program (refinement, load balancing, and solution) are given. The parallel efficiency, defined as the ratio of the time on one processor to the time on multiple processors, is given in Table 3 for the refinement and solution phases, and the total program. It may be observed that the efficiencies run between 56% and 87% for this example.

[1]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.

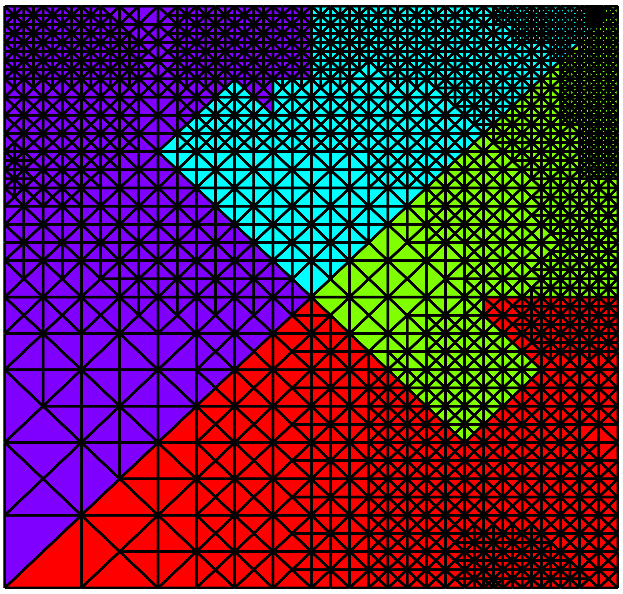

Figure 3. Example grid colored by partition.

Table 2. Execution times (wall clock seconds).

Proc.	Refine	Load Bal.	Solution	Total
1	2.83	0.00	17.72	20.81
2	3.93	3.15	20.39	27.70
4	4.30	3.64	22.37	30.49
8	5.09	4.14	26.19	35.58

Table 3. Parallel efficiency (per cent).

Processors	Refinement	Solution	Total
2	72	87	75
4	66	79	68
8	56	68	58

References

[1] R. E. Bank and M. J. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM J. Sci. Comp.*, to appear.

[2] R. E. Bank and A. H. Sherman. An adaptive multilevel method for elliptic boundary value problems. *Computing*, 26:91–105, 1981.

[3] K. Devine, B. Hendrickson, E. Boman, M. S. John, C. Vaughan, and W. Mitchell. Zoltan: A dynamic load-balancing library for parallel applications, user's guide. Technical Report SAND99-1377, Sandia National Laboratories, 2000.

[4] S. F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations*. Number 6 in Frontiers in Applied Mathematics. SIAM, Philadelphia, PA, 1989.

[5] W. F. Mitchell. Adaptive refinement for arbitrary finite element spaces with hierarchical bases. *J. Comp. Appl. Math.*, 36:65–78, 1991.

[6] W. F. Mitchell. Optimal multilevel iterative methods for adaptive grids. *SIAM J. Sci. Statist. Comput.*, 13:146–167, 1992.

[7] W. F. Mitchell. The full domain partition approach to distributing adaptive grids. *Applied Numerical Mathematics*, 26:265–275, 1998.

[8] W. F. Mitchell. The full domain partition approach to parallel adaptive refinement. In *Grid Generation and Adaptive Algorithms*, number 113 in IMA Volumes in Mathematics and its Applications, pages 151–162. Springer-Verlag, New York, NY, 1998.

[9] W. F. Mitchell. A parallel multigrid method using the full domain partition. *Electronic Trans. Num. Anal.*, 6:224–233, 1998.

[10] W. F. Mitchell. A refinement-tree based partitioning method for adaptively refined grids. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, 2001.