A Comparison of Three Fast Repartition Methods for Adaptive Grids*

William F. Mitchell†

Abstract

We examine three parallel algorithms for repartitioning adaptive finite element grids in the context of dynamic load balancing for adaptive multigrid methods. The first is a recursive bisection method based on the refinement tree of the adaptive refinement process. The second is a k-way version of the refinement tree partitioning method. The third is a multilevel repartitioning algorithm from ParMETIS. The methods are compared numerically via execution time of the partitioner and quality metrics of the partition.

1 Introduction

An adaptive multigrid method solves an elliptic partial differential equation (PDE) by beginning with a very coarse grid and cycling through phases of adaptive refinement of the grid and multigrid solution of the linear system of equations resulting from discretization of the PDE on the adaptive grid. In a parallel adaptive multigrid method, the adaptive refinement phase causes the load balance over the processors to become unequal. If the load is too unbalanced, the grid must be repartitioned and redistributed before continuing with the solution phase.

An important part of a parallel adaptive multigrid method is the method for determining this partition. In this context, it must not only produce equal sized sets to balance the load and minimize cut edges to reduce communication, but must also be very fast to not dominate the computation time of fast multigrid, and must produce similar partitions for the refinement of a grid to reduce redistribution costs. In this paper we examine three mesh partition methods in the context of repartitioning finite element grids after the refinement phases of an adaptive multigrid method.

In the next section, the three partitioning algorithms are briefly described. Further details of the methods can be found in the references. In the subsequent section, the numerical comparison is presented.

2 Partitioning Algorithms

The first partitioning algorithm we consider is the Refinement Tree Recursive Bisection (RTRB) method [3], [4]. This method uses the refinement tree created by the adaptive refinement method. The leaves, which correspond to triangles in the grid, and interior nodes, which correspond to triangles that have been refined, may be weighted to reflect the work associated with each triangle. A weight of 1 at the leaves and 0 at the interior nodes

†National Institute of Standards and Technology, Gaithersburg, MD.

will create partitions with equal numbers of triangles. The weights are summed to give a weight to every subtree. A tree corresponding to bisection refinement is a binary tree. The tree is bisected into two parts of equal weight by constructing a path from the root to a leaf. At each node along the path, it is decided which child will be the next node in the path, and which partition will receive all the nodes in the subtree rooted at the other child. The algorithm is recursively applied to the two subtrees to form four partitions, etc. In the parallel version of the algorithm, the tree is distributed over the processors and the summation of the weights is performed in parallel. Once all processors have the summed weights for their subtree, all processors can determine the bisection path independently with no further communication.

The second partitioning algorithm is the Refinement Tree K-way (RTK) method [6]. RTK also begins with the weighted and summed refinement tree. But instead of bisecting the tree and recursively applying the algorithm, the tree is split into k equal parts all at once. The tree is traversed in a truncated depth first manor. At each visited node, if the weight of that subtree will fit into the current partition, then all nodes in that subtree are assigned to the current partition. If they will not fit, then the children of the node are visited so the subtree is split among two or more partitions. When the current partition is filled, the next partition becomes current. By visiting the children in a carefully chosen order, the partitions are guaranteed to be contiguous. This method is very similar to the Octree Partition method [1], and can be related to space filling curve methods [7].

The final partitioning algorithm is the multilevel diffusion algorithm [8] with local balancing criteria implemented in the popular ParMETIS library [2]. Other methods in ParMETIS were also considered, but preliminary results and papers on ParMETIS indicated that this algorithm performs best in the given context. This algorithm consists of three phases: coarsening, multilevel diffusion and multilevel refinement. The coarsening phase merges nodes in the graph to create a multilevel hierarchy. In the multilevel diffusive phase, nodes are moved to neighboring partitions to balance the load. The multilevel refinement phase improves the quality of the partitions by reducing the edgecut.

3 Numerical Results

Numerical computations were performed to compare the three methods in terms of execution time and partition quality. These computations were performed on a cluster of eight 200 MHz Pentium Pro [1] based PCs connected by 100 BaseT ethernet and operating under Linux 2.1.117. Programs were compiled with Absoft Fortran 90 1.01 or gcc 2.7.2.1. Message passing was performed with the LAM 6.0 implementation of MPI.

The methods were compared in the context of an adaptive refinement multigrid finite element program with triangular elements in two dimensions. The Full Domain Partition [5] is used to extend the partitions to cover the whole domain. Laplace's equation was solved on the unit square with a singularity in the boundary condition at (0.7,1.0) which focuses the refinement to be near that point. The program begins with a grid containing eight triangles and cycles through four phases: 1) refine to approximately double the number of triangles, 2) repartition the grid, 3) redistribute the data, 4) solve the linear system using two multigrid V-cycles. The program terminates when there are approximately 64,000

[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.

TABLE 1
Time for partition (sec.).

Processors	RTRB	RTK	Metis
2	1.02	0.55	4.11
3	-	0.43	6.28
4	1.87	0.43	9.24
5	-	0.44	9.97
6	-	0.44	10.70
7	-	0.55	11.29
8	8.68	0.58	12.45

triangles. Runs were made using from two to eight processors, with the number of partitions equal to the number of processors. RTRB is applicable only when the number of processors is a power of two.

Table 1 shows the execution time of the partitioning algorithms. This is "wall clock" time and includes all of the time spent by the partitioner during the run. Since ParMETIS uses different data structures than the finite element code, the data was copied into ParMETIS's data structures. The time for this copying is not included in the measurements. It is easily seen that RTK is by far the fastest and most scalable of the three methods, running ten to twenty times faster than ParMETIS.

Table 2 compares the time spent partitioning to the time spent in refinement and solution. It is desirable to have the partitioner be much faster than the "real" work of refining and solving, so that the speed ups obtained from parallelization are not lost to the time spent partitioning. For this example, we see that RTK uses about 5-7% of the refinement and solution time, RTRB starts fast but approaches the refinement and solution time with more processors, and ParMETIS quickly dominates the time spent in refinement and solution.

The remaining tables examine the quality of the partitions generated by the three methods. Table 3 shows the number of cut edges in the final grid. A cut edge is a triangle side for which the end points belong to different processors. The number of cut edges reflects the volume of data that must be communicated during the solution process. Table 4 presents the ratio of the number of triangles in a partition to the number of triangles in the Full Domain Partition extension of the partition, averaged over the partitions of the final grid. The reflects the amount of extra work that is performed because of the extended partition. Values close to 1.0 are desirable. Finally, Table 5 presents the number of triangles that were moved between processors during data redistribution after repartitioning, summed over all processors and redistributions. This indicates the volume of data communicated during data redistribution. In general the results in these three tables show the three methods to be very close to each other. In Table 5 where we do see some significant differences, we find that sometimes RTK is the best and sometimes ParMETIS is the best.

4 Conclusions

The paper used a numerical example to compare three partitioning algorithms in the context of dynamic load balancing for an adaptive multigrid program. The results showed the k-way refinement tree method (RTK) to be the fastest method, running ten to twenty times faster

TABLE 2
Ratio of time for partition to time for refinement and solution.

Processors	RTRB	RTK	Metis
2	0.083	0.045	0.337
3	-	0.050	0.640
4	0.179	0.051	0.950
5	-	0.056	1.120
6	-	0.064	1.198
7	-	0.062	1.440
8	0.774	0.070	1.195

TABLE 3
Number of cut edges.

Processors	RTRB	RTK	Metis
2	1824	1824	2082
3	-	2635	2304
4	3719	3478	3979
5	-	4273	4775
6	-	4794	5639
7	-	5663	6199
8	6695	6461	7258

TABLE 4
Ratio of owned triangles to total triangles.

Processors	RTRB	RTK	Metis
2	0.94	0.94	0.94
3	-	0.92	0.93
4	0.87	0.89	0.88
5	-	0.87	0.86
6	-	0.86	0.84
7	-	0.84	0.83
8	0.81	0.83	0.81

TABLE 5
Number of triangles transfered.

Processors	RTRB	RTK	Metis
2	4643	4643	5378
3	-	7123	9153
4	44685	33009	43915
5	-	43540	44451
6	-	26032	50604
7	-	108396	42618
8	117256	94035	86913

than the multilevel diffusion algorithm as implemented in ParMETIS. The execution time of RTK is also small compared to the execution time of adaptive refinement and multigrid solution. The results show very little difference in the quality of partitions generated by the three methods. While broad sweeping conclusions cannot be made on the basis of a single example, it appears that simple partitioning algorithms like RTK may be useful when a fast partitioner is required, as in dynamic load balancing for adaptive multigrid methods.

References

[1] J. E. Flaherty, R. M. Loy, C. Ozturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, *Parallel Structures and Dynamic Load Balancing for Adaptive Finite Element Computation*, Appl. Numer. Maths., 26 (1998), pp. 241–263.

[2] G. Karypis, K. Schloegel, and V. Kumar, *ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 2.0*, http://www.cs.umn.edu/ karypis/metis/metis.html.

[3] W. F. Mitchell, *Refinement Tree Based Partitioning for Adaptive Grids*, Proceedings of the 7^{th} SIAM Conference on Parallel Processing for Scientific Computing, SIAM (1995), pp. 587–592.

[4] W. F. Mitchell, *The Refinement-Tree Partition for Parallel Solution of Partial Differential Equations*, NIST Journal of Research, 103 (1998), pp. 405–414.

[5] W. F. Mitchell, *The Full Domain Partition Approach to Distributing Adaptive Grids*, Applied Numerical Mathematics, 26 (1998), pp. 265–275.

[6] W. F. Mitchell, *The k-way Refinement Tree Partitioning Method for Adaptive Grids*, submitted.

[7] A. Patra and J. T. Oden, *Problem Decomposition for Adaptive hp Finite Element Methods*, Comp. Sys. Engng., 6 (1995).

[8] K. Schloegel, G. Karypis, and V. Kumar, *Parallel Multilevel Diffusion Algorithms for Repartitioning of Adaptive Meshes*, Technical Report #97-014, Dept. Computer Science, U. of Minnesota, 1997.