A Refinement-Tree Based Partitioning Method for Adaptively Refined Grids*

William F. Mitchell†

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 can cause 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 present the K-way Refinement Tree (RTK) partitioning method, a new method for partitioning grids that were created by adaptive refinement. The method uses a weighted tree representation of the refinement process that created the grid. A traversal of the tree sums the weights of the nodes. A second traversal of the tree places subtrees into sets to quickly determine equally-weighted connected partitions.

The RTK method was developed as a k-way version of the Refinement Tree Recursive Bisection (RTRB) method [6]. RTRB is the same as RTK except that

*Contribution of NIST, not subject to copyright.

†Mathematical and Computational Sciences Division, National Institute of Standards and Technology, Gaithersburg, MD 20899-8910

the partitioning phase partitions the grid into only two parts, and is recursively applied until the desired number of partitions have been generated. The motivation for developing RTK was the amount of communication involved in RTRB. The weight summation phase must be performed on each subtree partitioned during the recursive bisection, and each of these involves one communication step. This is a total of $p-1$ all-to-all communication steps for RTRB, which quickly becomes a bottleneck as p increases. In contrast, RTK uses only one communication step independent of p.

Although derived from RTRB, the RTK method is very closely related to the Octree Partition method (OCTPART) of Flaherty et al. [2, 3]. The primary difference between RTK and OCTPART is the generation of the tree. Usually in OCTPART, the tree represents a geometric refinement of a region covering the domain through local subdivision of octants, rather than the refinement of some initial set of coarse grid elements. Many of the nodes in the octree do not represent coarser grid elements as in RTK. In most cases this is of little consequence, but there may be situations where having the correspondence to coarse grid elements is useful. The remainder of the OCTPART method is very similar to RTK, with only a few minor differences.

Another class of related methods are the space filling curve methods [4, 7]. These methods produce a curve that passes through every element once, and cut the curve into segments to define the partitions. In RTK, if the children in the refinement tree are properly ordered then the path passing through the elements ordered by a traversal of the refinement tree is a space filling curve. This can be viewed as a topological generation of a space filling curve, in contrast to the traditional algebraic generation.

The multilevel methods, like the multilevel diffusion algorithm of Schloegel et al. [8], also use a tree-like approach to partition grids quickly. These algorithms consist of three phases: coarsening, multilevel diffusion and multilevel refinement. The coarsening phase merges nodes in the graph to create a multilevel hierarchy, analogous to the refinement tree. 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. These methods are more generally applicable by using an artificial tree, but require more work than a simple method like RTK.

The rest of the paper is organized as follows. Section 2 describes the K-way Refinement Tree algorithm (RTK). In Section 3, a method is presented for determined the order of the children such that a space filling curve is generated within each initial element. Section 4 contains numerical results to compare RTK with two other fast partitioning methods, recursive coordinate bisection and a multilevel diffusive method from ParMETIS.

2 K-way Refinement Tree Partitioning Method

This section describes the K-way Refinement Tree (RTK) algorithm. It begins with a definition of the refinement tree, describes the RTK algorithm, and presents the

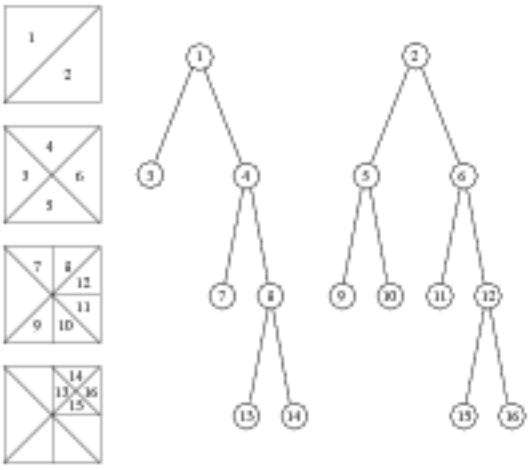

Figure 1. *Correspondence between grid refinement and the refinement tree.*

parallel form of the RTK algorithm.

The refinement tree is a representation of the refinement processes that produced an adaptive grid from an initial grid, as illustrated for bisected triangles in Figure 1. 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. Leaf nodes correspond to the elements in the final grid. Nodes on the first level correspond to the elements of the initial coarse grid. The root is an artificial node with the first level nodes as children. Alternatively, an artificial tree can be constructed above the first level nodes, ultimately terminating with a single root.

In RTK, the nodes of the tree are weighted. The weight assigned to a node should be related to the amount of work associated with the corresponding element. For example, elements containing a Dirichlet boundary may have a smaller weight than elements in the interior of the domain. The weights are not limited to leaf nodes; one may wish to apply weights to interior nodes to, for example, represent the work on a coarse grid of a multigrid solver. In this paper we consider only the simplest weighting scheme in which the leaf nodes have weight 1.0 and the interior nodes have weight 0.0. This results in a partitioning of the elements of the final grid into sets that differ in size by at most one.

RTK consists of two phases. In the first phase, every node is labeled with the sum of the weights in the subtree rooted at that node. This is accomplished by a depth first traversal of the tree, and takes $O(N)$ operations where N is the number of nodes in the tree. In the second phase, a truncated depth first traversal of the tree is performed to create the partitions. The desired size of each partition is determined by dividing the summed weight at the root node by the number of partitions, and the partitions are initialized to be empty. During the traversal, the summed weights at the nodes are examined relative to the size of the partition under

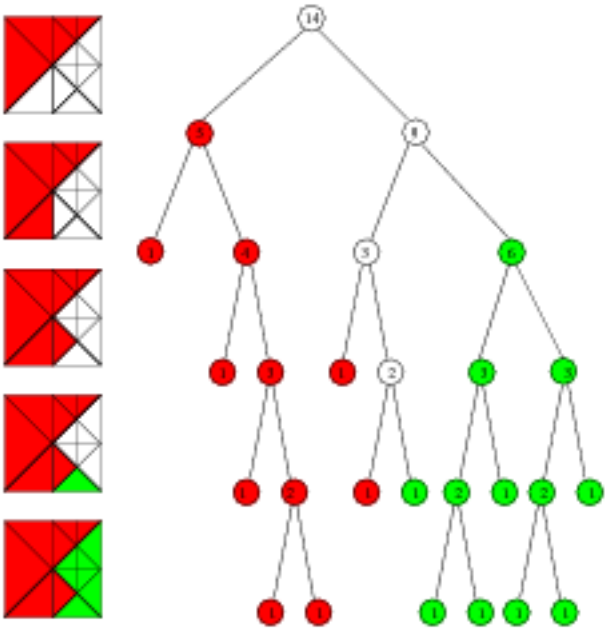

Figure 2. *Partitioning the grid and tree into two sets.*

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. Figure 2 illustrates this process for a simple grid partitioned into two parts. Most of the time the subtree will fit in the current partition, so large portions of the grid are assigned to a partition at the same time. Only when a partition is nearly full will the traversal go deep into the tree to find a small enough subtree to fill out the partition. If there are p partitions and the depth of the tree is $O(\log N)$, one would expect the second phase to require $O(p \log N)$ operations.

In a parallel implementation, the grid is distributed over the p processors. Each processor contains a refinement tree that includes at least the initial grid and the part of the grid belonging to this processor. Parts of the grid belonging to other processors will have been pruned from the tree. Thus the summation of the weights must be done in a distributed manner. This can be accomplished with two tree traversals and one all-to-all communication step. In the first tree traversal, the weights are summed for nodes that belong to this processor. Leaf nodes that are pruning points are given the weight 0.0, but otherwise the summation occurs as usual. The processors then exchange information to provide the summed weights for the pruning points, by each processor sending what it has as the summed weight for each node that is a pruning point on a different processor. Note that a processor may receive contributions for a pruning point from more than one processor, and

the sum of these is the correct summed weight for that node. Now with the summed weight available for the pruning points, a second traversal is performed to obtain the correct summed weights for the entire tree.

Each processor now has sufficient information to perform the second phase of the RTK algorithm independently. Without further communication, all processors will obtain the same partition, except for details within parts of the grid that the processor does not have. These details are not needed since all the processor needs to know is the new assignment for the elements it currently has so that it can send those elements to the new owner during the redistribution of data.

For a grid with N elements, if each processor has $O(N/p)$ elements and $O(\sqrt{N/p})$ "shadow" elements, the expected number of operations on each processor is $O(N/p)$.

3 Child Order

RTK will partition the grid into sets that are as equal-sized as possible for the given weights, but one must also consider the quality of the partition. In dynamic repartitioning, one looks for a reasonably good partition quickly, rather than spending a lot of time optimizing the quality of the partition. RTK uses a heuristic approach in which a reasonably good partition is obtained by keeping the partition connected. It is easily seen that large connected blocks of elements are added to a partition at the same time, since the elements corresponding to a subtree are all the descendents of a single element. But this alone does not guarantee connected partitions. It is important to pick the right order in which to traverse the children of a node to guarantee connected partitions. This section presents one method of determining the child order that works with most element types and refinement strategies.

Begin with an ordering of the elements of the initial coarse grid such that each element is connected to the element that follows it. Within each element, designate one vertex as the *in* vertex, and a different vertex as the *out* vertex such that the *out* vertex of an element is the same as the *in* vertex of the next element. (Automatically determining such an ordering and designation is currently a research topic.) Then, for the remainder of the refinement tree, the child order and *in*/*out* designation are determined such that the first child contains the *in* vertex of the parent, the last child contains the *out* vertex of the parent, and the *out* vertex of the k^{th} child is the *in* vertex of the $(k+1)^{st}$ child. Such an order and designation can be found provided that the *in* and *out* vertices do not lie only in the same child. Of the common element types and refinement strategies, the only known examples where they cannot be found are bisection of quadrilaterals in 2D and bisection of hexahedra in 3D. A different approach to finding the child order would be required in these cases.

At worst, the order and designation can be found by exhaustive search. Although this has exponential complexity in the number of children, the number of children is typically bounded and very small, usually two, four or eight. So this is not an infeasible approach. But for a known element type and refinement strategy, one can use that knowledge to determine the order and designation *a priori*,

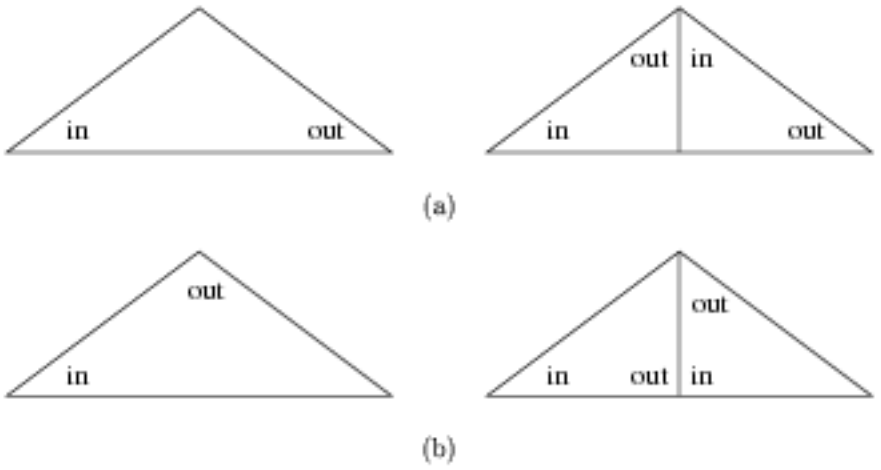

Figure 3. *Template for element order and* in/out *designation for bisected triangles. (a) The peak is not an* in/out *vertex. (b) The peak is one of the* in/out *vertices.*

and define templates to quickly make the assignments. The templates for bisected triangles and quadrisected quadrilaterals are shown in Figures 3 and 4. Templates have also been found for quadrisected triangles (a.k.a. regular or red refinement), bisected tetrahedra, octasected tetrahedra, and octasected hexahedra.

4 Numerical Results

Numerical experiments were performed to compare RTK with Recursive Coordinate Bisection (RCB) and a multilevel diffusive method in terms of execution time and the quality of the partitions. These experiments were performed using PHAML, a parallel adaptive multigrid program for solving elliptic PDEs. PHAML is written in Fortran 90 and uses MPI for message passing. Adaptive refinement is by newest node bisection of triangles.

RTK was implemented as part of PHAML. RCB was implemented in version 1.1 of Zoltan [1], a C language library for dynamic load balancing. The multilevel diffusive method was obtained from version 2.0 of the ParMETIS library [5], a C language library that is currently the most widely used parallel library for graph partitioning. The method used from ParMETIS is RepartLDiffusion, with PartKway for the first partition, and ParMETIS was accessed through Zoltan.

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

[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.

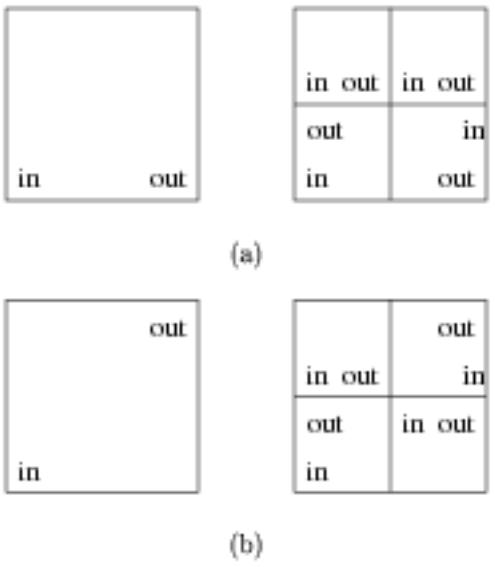

Figure 4. *Template for element order and* in/out *designation for quadrisected quadrilaterals. (a) The* in/out *vertices are adjacent. (b) The* in/out *vertices are opposite.*

2.2.16. Programs were compiled with Lahey/Fujitsu LF95 5.5 or egcs 2.91.66 using "-O" for optimization. Message passing was performed with the LAM 6.3.1 implementation of MPI.

Laplace's equation was solved on the unit square with Dirichlet boundary conditions. On three sides the boundary condition is homogeneous. On the top it is piecewise linear and continuous, going from 0.0 at (0.0,1.0) to 1.0 at (0.7,1.0) to 0.0 at (1.0,0.0). The singularity focuses the refinement to be near (0.7,1.0). The program begins with a grid containing two triangles (with an artificial root added to the refinement tree) 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. To avoid extraneous communication, the program is run on one processor without partitioning until the grid has 1000 vertices. The program terminates when there are approximately 64,000 vertices (128,000 triangles). This gives seven applications of grid partitioning, when the number of vertices is approximately 1000, 2000, 4000, ..., 64000. Runs were made using from two to eight processors, with the number of partitions equal to the number of processors. Figure 5 shows an example grid with approximately 1000 vertices and the four-set partition produced by each of the three methods.

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, including communication. It is easily seen that RTK is slightly faster than RCB and two to three times faster than ParMETIS. It may be noted that the time does not

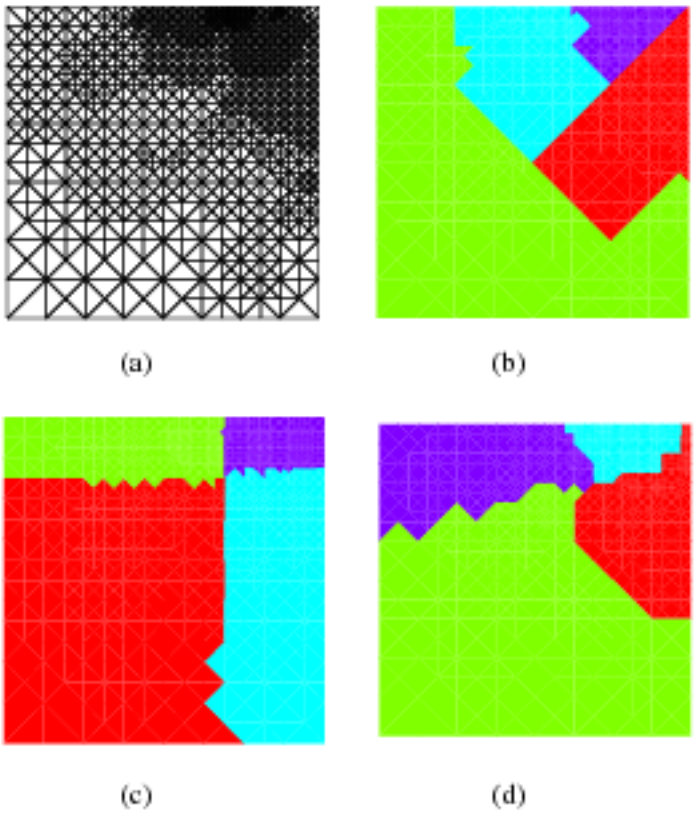

Figure 5. *(a) Sample grid, and partitions for four processors from (b) RTK, (c) RCB, and (d) ParMETIS.*

decrease like $O(N/p)$ as expected from the operation count in Section 2. This is due to the time spent in communication. Table 2 shows the breakdown of the time for RTK into computation and communication. Here it is seen that the computation time decreases like $O(N/p)$ and the communication time increases like $O(\log p)$.

Tables 3 and 4 show measures of the quality of the partitions. Table 3 shows the number of cut edges in the final grid. A cut edge corresponds to a triangle side that is shared by triangles in different partitions. The number of cut edges reflects the volume of data that must be communicated during the solution process. Table 4 shows the wall clock execution time for migration of data after repartitioning. This is a measure of how similar the new partitions are to the old partitions, in the sense of how much data must be moved between processors to create the new partitions. These results show no significant difference between the methods in terms of quality of partition.

Table 1. *Time for partition (sec.).*

Processors	RTK	RCB	Metis
2	1.19	0.90	2.45
3	0.84	0.92	1.99
4	0.74	0.98	1.85
5	0.71	0.72	1.84
6	0.63	0.71	1.84
7	0.60	0.85	1.90
8	0.64	0.97	2.04

Table 2. *Breakdown of time for RTK.*

Processors	Computation	Communication
2	1.16	0.03
3	0.79	0.05
4	0.64	0.10
5	0.61	0.10
6	0.49	0.14
7	0.44	0.16
8	0.45	0.19

5 Summary

This paper introduced the K-way Refinement Tree partitioning method for grids that were created by adaptive refinement. It was originally derived from the Recursive Bisection Refinement Tree partitioning method, but is also closely related to the OCTREE method and space filling curve methods. RTK uses a tree representation of the refinement process with weights representing the amount of work associated with each element. When executed in parallel on p processors, the expected number of operations for partitioning into p sets is $O(N/p)$ with only one communication step. A method was presented for determining the child order for most common element types and refinement strategies. A numerical example was presented in which RTK is two to three times faster than a multilevel diffusive method from ParMETIS, and slightly faster than recursive coordinate bisection, while producing partitions of similar quality.

Table 3. *Number of cut edges.*

Processors	RTK	RCB	Metis
2	639	300	207
3	491	669	414
4	842	572	713
5	847	860	872
6	1120	1098	1019
7	1204	1212	1145
8	1343	1130	1337

Table 4. *Time for data migration. (sec.).*

Processors	RTK	RCB	Metis
2	0.64	0.76	0.58
3	0.50	0.85	0.56
4	0.42	1.39	0.54
5	0.53	0.50	0.48
6	0.39	0.45	0.42
7	0.35	0.61	0.45
8	0.45	0.68	0.37

Bibliography

[1] K. Devine, B. Hendrickson, E. Boman, M. St. John, C. Vaughan and W.F. Mitchell, *Zoltan: A Dynamic Load-Balancing Library for Parallel Applications*, User's Guide, Sandia Technical Report SAND99-1377, 2000.

[2] J.E. Flaherty, R.M. Loy, M.S. Shephard, B.K. Szymanski, J.D. Teresco, and L.H. Ziantz, *Adaptive Local Refinement with Octree Load-Balancing for the Parallel Solution of Three-Dimensional Conservation Laws*, J. Parallel and Dist. Comput. 47 (1997), pp. 139–152

[3] J.E. Flaherty, R.M. Loy, C. Özturan, 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. Num. Math. 26 (1998), pp. 241–263.

[4] M. Griebel and G. Zumbusch, *Hash-Storage Techniques for Adaptive Multilevel Solvers and Their Domain Decomposition Parallelization*, Proc. Domain Decomposition Methods 10, Contemporary Mathematics, Vol. 218, (AMS, Providence, 1998), pp. 271–278.

[5] 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.

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

[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.