Refinement Tree Based Partitioning for Adaptive Grids

William F. Mitchell*

Abstract

We present a new partitioning algorithm for grids obtained by adaptive refinement. The method uses the adaptive refinement tree to obtain information unavailable to other partitioning methods which use only the final grid and/or some geometric data. The algorithm requires (typically) $O(\log(N))$ operations after an $O(N)$ preprocessing step. The method is guaranteed to produce perfectly balanced connected partitions. Numerical examples indicate that the partitions have only about 10% more crossings than methods that require many times the execution time, and unlike other methods, there is reason to expect similarity of the partitions of nested grids.

1 Introduction

The numerical solution of partial differential equations (PDEs) is often the most computationally intensive part of solving mathematical models of physical phenomena. For this reason, much research has been performed to find faster methods to solve PDEs at higher resolution. A recent development involves the combination of solution-adapted unstructured grids, to provide higher resolution only where it is needed, with multi-level (multigrid) linear system solution techniques, which have the smallest possible asymptotic operation counts. Computer programs based on these methods, such as MGGHAT [5] and PLTMG [1], have been developed for sequential computers and released in the public domain. However, efficient implementation of such methods on parallel computers has not yet been achieved, primarily because of the difficulties associated with the dynamically changing nonuniform grid.

Most of the prior research in partitioning nonuniform grids has been focused on static unstructured grids (see, for example, the methods described in [3]) where the grid need only be partitioned once, or time-evolution grids [2] where there is only a small change in the grid to be repartitioned. In this paper we consider the problem of partitioning an adaptively refined and dynamically changing grid over the processors of a parallel computer in the context of the multilevel adaptive method for solving partial differential equations described in [4]. In this context there are three program phases that are cycled through several times: 1) adaptively refine the grid; 2) partition the new grid and redistribute over the processors; 3) multigrid solution.

The desirable properties of a partitioning algorithm in this context are:

Balance. The algorithm should produce equal sized partitions to balance the computational load on the processors.

Crossings. The number of edges crossing from one partition to another should be minimized to reduce the amount of interprocessor communication.

Speed. The algorithm should be very fast so the partitioning algorithm doesn't dominate the execution time of the PDE solver.

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

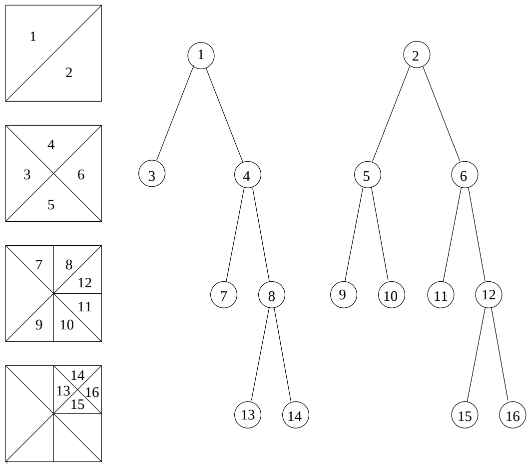


FIG. 1. *Refinement trees.*

Nestedness. The algorithm should produce similar partitions for two grids when one is a refinement of the other to minimize the amount of data migration during redistribution.

Connectedness. Each partition should be a connected set to provide locality compactness of the subproblem assigned to a processor, and reduce interprocessor communication.

Multilevel. For the sequence of grids used by a multigrid method, the balance and crossings properties should hold for all the grids.

2 Partitioning algorithm

A new method for partitioning grids has been developed based on the refinement tree. This method is presented in the context of partitioning the triangles formed by bisection refinement, but, with appropriate modifications, it can be applied to other refinements such as rectangles refined by quadrisection, and to the vertices (rather than triangles) of a grid formed by bisection of triangles.

The *refinement tree* of an adaptive triangular grid is a binary tree containing one node for each triangle that appears during the refinement process. The two children of a node correspond to the two triangles that are formed by bisecting the triangle corresponding to the parent node. In figure 1, the numbering of the triangles in the grid and the nodes in the tree indicates the relationship.

The algorithm begins by counting the number of leaves below each node in the refinement tree. This can be performed in $O(N)$ operations for N triangles, using a depth first traversal of the tree.

The leaves of the tree (triangles of the final grid) can be partitioned into two equal sized (plus or minus 1) sets by the following algorithm:

```
   current node <-- root
while (current node has children)
   determine which child shares a triangle side with the sibling of
     the current node (arbitrary choice if both do)
   tentatively assign that child to the same set as the sibling
     and the other child to the other set
   for each child of the current node, compute the sum of the number
```

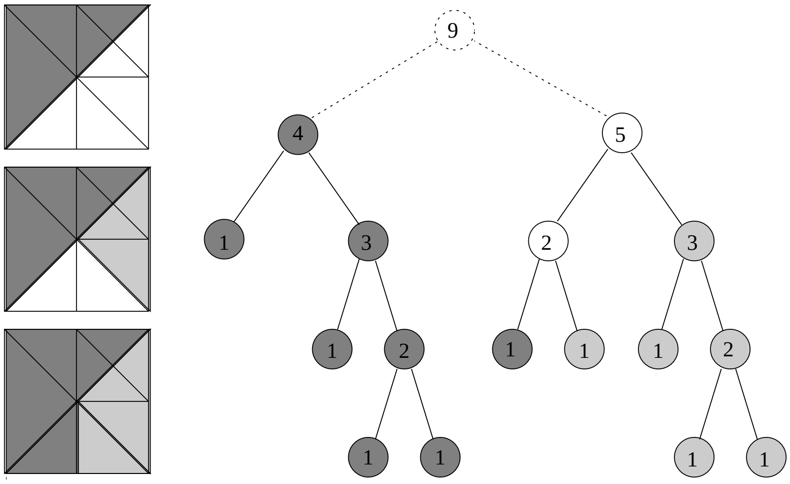


Fig. 2. *Partitioning the triangles into two sets.*

```
      of leaves below it with the number of triangles already in its
      tentative set
    assign the child with the smaller sum, and the subtree below it,
      to its set (the other child is not assigned)
    current node <-- other child
end while
```

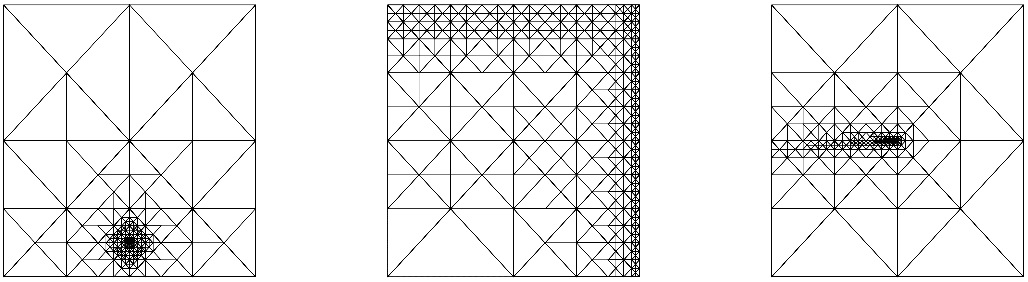
Note that the first time through the loop there is no sibling of the current node, and tentative assignment to the two sets is arbitrary. Also, during the last time through the loop both children are assigned to their sets.

This process requires $O(number\ of\ levels)$ operations, which is typically $O(log\ N)$. The algorithm can be applied to the two subtrees to form 4 sets, and repeated iteratively for 8, 16, ... The algorithm is modified slightly to account for the fact that in a subtree a parent may have only one child.

Figure 2 illustrates the algorithm for a simple triangulation. Light and dark grey are used to represent the two sets. The nodes are labeled with the number of leaf nodes below them. The assignment of the triangles to sets after each step is also illustrated.

It can be proven that if the child with the smaller sum is the one that is assigned to a set, then the number of remaining triangles to be assigned is never less than the difference between the number of triangles that have been assigned to each of the two sets. This guarantees that, in the end, the size of the two sets will be equal (or differ by one if N is odd), and that after repeated application of the algorithm to the subtrees the size of all the sets will differ by at most one. The algorithm is guaranteed to maintain connected sets by choosing the tentative assignment of the two children in such a way that the child that is connected to the sibling of the parent will go in the same set as the sibling of the parent.

The algorithm also seems (from numerical examples) to produce very similar partitions for two grids when one grid is a refinement of the other. The following heuristic may explain this. When an adaptive grid is refined, one would expect most of the refinement to occur in the same region that previous refinement occurred. In the refinement tree, this means that nodes that are heavy (i.e. have a lot of leaves below them) will tend to get heavier. Since



(a) (b) (c)

FIG. 3. *Sample grids based on (a) peak, (b) boundary layer, and (c) crack.*

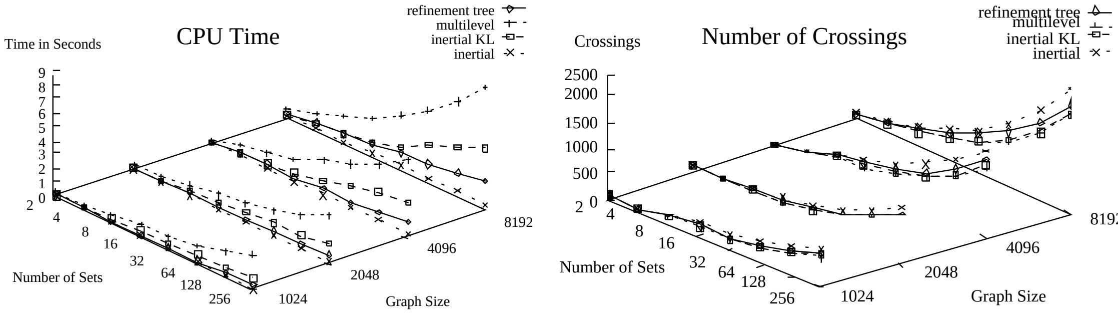


FIG. 4. *CPU time and number of edges crossing between partitions for the "peak" grid.*

the decision of which child is actually assigned to a set depends on which child is lighter, the same decision is likely to be made most of the time in two refinement trees when one represents an adaptive refinement of the other.

3 Numerical results

The refinement tree based partitioning method was compared with three methods from the Chaco package [3]: inertial bisection, inertial bisection with local improvements via Kernighan-Lin, and a multilevel method.

Three families of grids were used, based on a sharp peak, a boundary layer, and a crack in the interior of the domain as illustrated in Figure 3.

Within each family, four grids were used, with approximately 1024, 2048, 4096 and 8192 triangles. Partitions were made for 2, 4, 8, 16, 32, 64, 128 and 256 sets. Figure 4 contains results for the grid based on a sharp peak. Results for the other two problems are similar. The graph on the left side of the figure shows the CPU time required to partition the grids. The graph on the right shows the quality of the partition as measured by the number of edges crossing between sets of the partition.

Figure 5 shows, for three of the methods, the resulting partition of two grids for the sharp peak example. (The other method is similar to inertial bisection.) The first grid has about 1024 triangles. The second grid is a refinement of the first grid with about 2048 triangles.

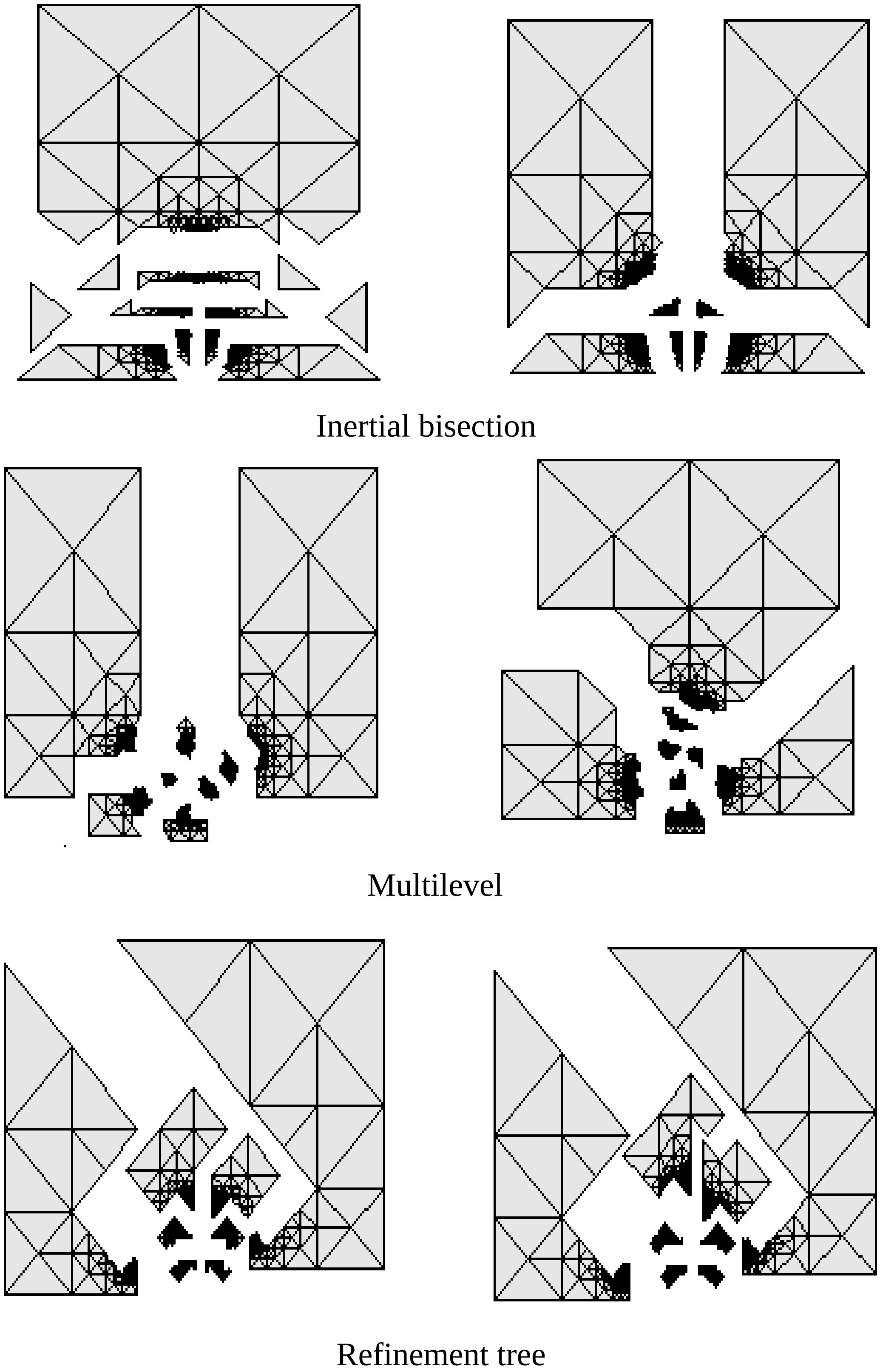


Fig. 5. *Partitions for the "peak" grid with 1K triangles (left) and 2K triangles (right).*

4 Observations

Based on these examples, we make the following observations in terms of the desirable properties listed in Section 1:

Balance. All the methods considered are guaranteed to produce perfectly balanced partitions.

Crossings. There is little difference between the multilevel method and inertial bisection with Kernighan-Lin. With some exceptions, the other methods are not far behind with the refinement tree method producing about 10-20% more crossings and inertial bisection about 20-30% more crossings.

Speed. Inertial bisection is the fastest method. The refinement tree method is second, with inertial bisection with Kernighan-Lin taking 2-3 times longer than the refinement tree method, and the multilevel method taking 5-6 times longer. We also note that the refinement tree method was implemented in Fortran 90 and compiled with the NAG F90 translator, which is not particularly efficient. A much faster implementation should be possible.

Nestedness. The refinement tree method shows very similar partitions for nested grids. Inertial bisection with Kernighan-Lin shows similar partitions, but has more movement than the refinement tree method. Inertial bisection can have radical changes. The multilevel method shows no relationship between partitions of nested grids.

Connectedness. The refinement tree method is guaranteed to produce connected sets. The other three methods are not, and all three produced disconnected sets on some examples.

Multilevel. This property was not addressed in this experiment. All of the methods have the ability to attach weights to the triangles, which should be usable for improving the multilevel properties.

5 Conclusion

The partitioning algorithm based on the refinement tree has potential in the context of adaptive multilevel methods for partial differential equations. It is very fast, produces perfectly balanced sets, and the number of crossings between sets is not much larger than those produced by much slower methods. It also appears that the partition of a grid is closely related to the partition of a refinement of that grid. Together, these observations imply that the partition algorithm will not dominate the PDE solver, the computational load is balanced, the interprocessor communication is near minimum, and the redistribution of data for a new partition is small.

References

[1] R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, User's Guide 7.0.* SIAM, Philadelphia, 1994.

[2] K. D. Devine, J. E. Flaherty, R. M. Loy and S. R. Wheat Parallel Partitioning Strategies for the Adaptive Solution of Conservation Laws. Technical Report 94-1, Department of Computer Science, Rensselaer Polytechnic Institute, 1994.

[3] B. Hendrickson and R. Leland. The Chaco User's Guide Version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, 1993.

[4] W. F. Mitchell. Optimal multilevel iterative methods for adaptive grids. *SIAM J. Sci. Statist. Comput.*, Vol. 13, pp. 146-167, 1992.

[5] W. F. Mitchell. MGGHAT: Elliptic PDE Software with Adaptive Refinement, Multigrid and High Order Finite Elements. *Sixth Copper Mountain Conference on Multigrid Methods*, N.D. Melson, T.A. Manteuffel and S.F. McCormick, Eds., NASA, 1993, 439-448.