

A Parallel, Adaptive Refinement Scheme for Tetrahedral and Triangular Grids

Alan Stagg¹, Jackie Hallberg², and Joseph Schmidt³

¹ Los Alamos National Laboratory, Applied Physics Division
P.O. Box 1663, MS P365
Los Alamos, NM 87545

`stagg@lanl.gov`

² U.S. Army Engineer Research and Development Center
Coastal and Hydraulics Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180

`pettway@juanita.wes.army.mil`

³ 2420 Wanda Way

Reston, VA 20191

`roig.and.schmidt@erols.com`

Abstract. A grid refinement scheme has been developed for tetrahedral and triangular grid-based calculations in message-passing environments. The element adaption scheme is based on edge bisection of elements marked for refinement by an appropriate error indicator. Hash table/linked list data structures are used to store nodal and element information. The grid along inter-processor boundaries is refined consistently with the update of these data structures via MPI calls. The parallel adaption scheme has been applied to the solution of an unsteady, three-dimensional, nonlinear, groundwater flow problem. Timings indicate efficiency of the grid refinement process relative to the flow solver calculations.

1 Introduction

Adaptive grid methods based on point insertion and removal have been popular for a number of years for achieving greater solution accuracy with relative cost efficiency. However, issues related to implementing such schemes on parallel systems are just now being addressed, and much work is needed to identify the best approaches.

In this paper we present a new approach for the h-refinement of irregular tetrahedral and triangular grids in message-passing environments. Data structures have been selected to simplify implementation and coding complexity as much as possible for refinement, coarsening, and load balancing components. This software has been implemented in the Department of Defense code ADH (ADaptive Hydrology) under development at the U.S. Army Engineer Research

and Development Center. ADH is a modular, parallel, finite element code designed to support groundwater, surface water, and free-surface Navier-Stokes modeling [1].

1.1 Serial Element Adaption Scheme

Given an initial grid, the model subdivides grid elements to achieve the desired resolution in regions of interest. The parallel grid adaption scheme developed here is based on the geometric splitting algorithm of Liu and Joe [2]. Elements are refined by edge bisection according to an error indicator, and elements can be merged to increase efficiency where high resolution is not required. A grid closure step is utilized to eliminate non-conforming nodes. Element edges are selected for bisection based on a modified longest-edge bisection approach in which the oldest edge in the element is first flagged for bisection followed by the longest edge. Refinement and coarsening of a tetrahedral element are illustrated in Figure 1. Here a new node is added to an edge, creating two new tetrahedra. The new elements can be merged to recover the original element by removing the inserted node.

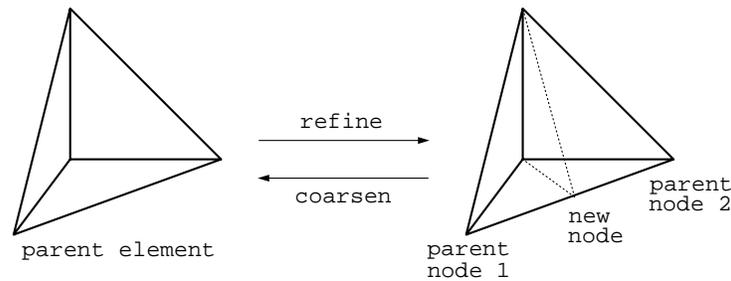


Fig. 1. Tetrahedral Grid Adaption Based on Edge Bisection

The refinement process in ADH begins with determination of elements to be split according to an explicit residual error indicator. The following pseudo-code describes the basic steps in the refinement scheme.

Refinement pseudo-code

```

loop over elements
  refine element via edge bisection if its error > threshold;

conforming_grid = false;
do while conforming_grid == false {
  conforming_grid = true;

```

```

loop over elements
  if element has an edge with a newly inserted node {
    refine element;
    conforming_grid = false;
  }
}

```

2 Parallel Implementation

The parallel implementation of local grid refinement schemes like the edge bisection scheme presents a number of challenges for the message-passing environment. First, in the standard approach where the grid is partitioned and subdomains are assigned to processors, the subdomains must be refined and coarsened consistently along processor boundaries. Also, closure requirements may force element refinement to spread to a processor that has no elements marked for refinement by the error indicator. Finally, the local adaption process will likely lead to load imbalances among the processors, and nodes and elements must be transferred between processors to maintain processing efficiency. In this paper we focus on describing the methodology developed for dealing with such difficulties when refining elements in parallel. Our grid coarsening methodology will be described in another paper.

In our approach the grid is partitioned by assigning element nodes to processors. Elements along processor boundaries are shared by the processors owning the element nodes. Nodal information for these elements is communicated between processors using MPI, and each processor stores complete data for its shared elements [3].

2.1 Data Structures

Data structures have been selected to simplify the parallel implementation of the adaption scheme and to facilitate the coupling of the refinement, coarsening, and load balancing components. During early work, we realized that common techniques like the use of tree structures for refinement could adversely impact other adaption components such as load balancing. In this case, the use of graph partitioners and the resulting grid point movement between processors requires splitting refinement trees between processors. To avoid these difficulties we use hash table/linked list structures [4]. Such structures are naturally suited for grid adaption since they are dynamic in nature and facilitate node and element searches. These structures support all grid refinement, coarsening, and load balancing requirements without complicating the implementation of any single component.

Hash tables are used to store nodes and element edges. Each entry in the node hash table consists of a node number local to the processor and a corresponding node identifier in the global grid. Each entry in the edge hash table consists of

the two local node numbers that define the edge, an integer edge rank based on comparative lengths of the edges, and an integer that stores the new node number if a node has been inserted on the edge. These node and edge hash tables are constructed prior to each refinement step, and the memory is freed upon completion of the refinement step.

2.2 Refinement

The grid refinement scheme presented here is primarily a local process and is thus amenable to parallel processing. The principal requirement in the parallel environment is that the grid along inter-processor boundaries be refined consistently. The elements that are shared along these boundaries are duplicated on each of the processors that own the elements' nodes. If one of these duplicated elements is marked for refinement, each processor that stores a copy of the element must bisect that element's edges in the same way.

To insure that shared elements are refined consistently, all element edges are ranked for bisection based on their lengths so that the edges are uniquely and consistently identified throughout the global grid. This ranking also facilitates refinement in elements that are not shared between processors. Integer ranks are utilized rather than computed edge lengths so that processors are easily able to make consistent edge bisection decisions when multiple edges in an element have the same length. The edge hash tables are used to store these ranks.

In our partitioning approach, an edge shared by two processors will appear in each of the processors' hash tables, and a protocol must be established to maintain consistency of the edge hash tables between processors. To support this communication, edge communication lists are constructed which provide a mapping between these duplicated edge storage locations. For each such edge, one of the processors sharing the edge is assigned ownership of it.

The edge ranking follows construction of the edge communication lists. After a parallel odd-even transposition sort, global ranks for edges are returned to the processors owning the edges, and the ranks are stored by these processors in their edge hash tables. These processors then communicate the ranks to the processors sharing the edges using the edge communication lists that have been constructed. Finally, the receiving processors store the ranks in their edge hash tables to complete the edge ranking process.

Elements are selected for refinement based on the error indicator, and edges in these elements are selected for bisection based on their age and rank. The oldest edge is identified first for bisection. However, if all edges in an element have the same age, the longest edge (identified by rank comparisons) is marked for bisection. If the marked edge has not been bisected by an adjacent element, a new node is created and its number is stored in the edge hash table. Edges that have not been bisected have a negative entry in the new node location in the edge data structure. Two new elements are created using the new node on the bisected edge, and the element Jacobians and other data are corrected for these elements. The new node entries for the edges in these new elements are reinitialized to indicate that new nodes are not present.

2.3 Grid Closure

After the initial refinement step, further refinement may be required to obtain a conforming grid. In the serial case each element's edges are checked for the presence of new nodes in the edge hash table. If any element has an edge with a new node, that element is marked for refinement according to the established rules. The refinement process continues iteratively until a conforming grid is obtained.

In the parallel environment this procedure is complicated by the fact that shared edges may be bisected by only one of the processors sharing the edge. To maintain consistency of the edge hash tables, processors owning shared edges must communicate new node information to the other processors sharing the edges. If a message indicates that an edge has a new node, then the receiving processor creates a new node for the edge and updates its hash table.

An example is illustrated in Figure 2 where three elements are distributed over two processors as indicated by the shaded background. The center and right elements are shared by the two processors. In the first step, processor P0 splits the left element because of high error. Next the original center element is split in the closure phase because that element now has a new node on one of its edges. Note that the center element's longest edge is bisected rather than the edge with the new node. Following this second step, the copy of the right element on processor P0 can be refined since the shared edge owned by processor P0 has been updated with the new node number. However, the new node on the shared edge must be communicated to processor P1 using the edge communication lists to inform processor P1 that its copies of the center and right elements must be refined. In this example grid refinement has spread from processor P0 to processor P1 even though processor P1 did not have any elements marked for refinement by the error indicator.

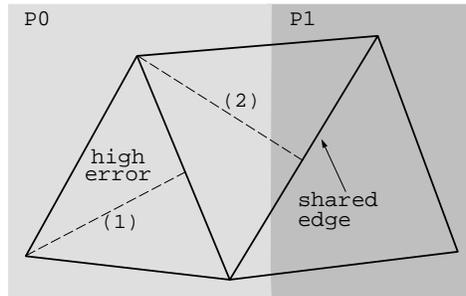


Fig. 2. Grid Closure on Two Processors

Similarly, processors may bisect edges they do not own. To handle this situation, the edge communication lists are utilized in reverse order (the send list becomes a receive list, and vice versa) so that processors owning shared edges can update their hash tables if other processors bisect them. After this communication, the elements with new nodes on edges are refined, and the process is repeated iteratively until a conforming grid is obtained.

3 Groundwater Application

The capabilities of the parallel grid refinement scheme have been investigated for the solution of a draining heterogeneous column. In this problem a column is filled with a mixture of clay, silt, and sand. The column consist primarily of sand with a clay lens near the bottom and silt lenses in several places throughout the column. Initially, the column is completely saturated with water, and then the water is allowed to drain from the bottom of the column. The grid is allowed to refine and coarsen locally as dictated by the explicit error indicator, and dynamic load balancing is utilized to improve processor efficiency.

A snapshot of the adaptively refined grid for the hegerogeneous column is illustrated in Fig. 3. The area shaded black represents the clay material, while

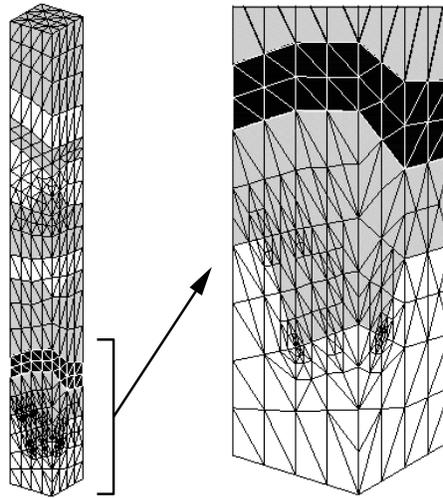


Fig. 3. Adaptively Refined Grid for Heterogeneous Column

the sand and silt are represented by the gray and white regions, respectively. Grid refinement is visible at the sand/silt interface in the lowest points of the sand. This refinement is indicative of the large head gradient from the sand to

the silt. Water travels through the sand at a faster rate than through the silt due to the silt's lower conductivity. As a result, the water ponds, or collects, in the low points of the sand until the pressure is great enough to push the flow across the interface.

The grid adaption software is currently being tested, and preliminary timings are only available for the heterogeneous column described above. Due to the coarse grid size for this problem, timings were only obtained on up to 8 processors of the Origin2000. During the refinement step, the number of nodes was increased by 35%, and the total time spent in grid refinement was an order of magnitude less than the time spent in the flow solver for timings on one through eight processors. The time spent in grid closure, including communication, was less than half of the total time in grid refinement for these cases. Though preliminary, these timings indicate efficiency of the grid refinement process relative to the flow solver.

4 Conclusion

A parallel refinement scheme has been developed for tetrahedral and triangular grids. The refinement scheme and data structures described here have been developed to facilitate the parallel implementation of both grid refinement and coarsening. The refinement and coarsening schemes are based on communicating a minimum set of data and reconstructing information locally without the use of tree structures. The goal with this approach is a balanced design between refinement, coarsening, and load balancing in terms of efficiency and ease of implementation. Preliminary application of the adaptive grid scheme to an unsteady groundwater flow problem has demonstrated the capability and efficiency of the method.

Funding for this project was provided by the Department of Defense High Performance Computing Modernization Office, and computer time was provided by the ERDC Major Shared Resource Center in Vicksburg, MS. Permission to publish this paper was given by the Chief of Engineers and by Los Alamos National Laboratory.

References

1. Jenkins, E.W., Berger, R.C., Hallberg, J.P., Howington, S.E., Kelley, C.T., Schmidt, J.H., Stagg, A.K., and Tocci, M.D., "Newton-Krylov-Schwarz Methods for Richards' Equation," submitted to the *SIAM Journal on Scientific Computing*, October 1999.
2. Liu, A. and Joe, B., "Quality Local Refinement of Tetrahedral Meshes Based on Bisection," *SIAM Journal on Scientific Computing*, vol. 16, no. 6, pp. 1269-1291, November 1995.
3. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI- The Complete Reference, Volume 1, The MPI Core*, The MIT Press, Cambridge, Massachusetts, 1998.
4. Cormen, T., Leiserson, C., and Rivest, R., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.