7332

P.8

PARALLEL ADAPTIVE MESH REFINEMENT WITHIN THE PUMAA3D PROJECT[1]

Lori Freitag
Argonne National Laboratory
Argonne, IL

Mark Jones
The University of Tennessee
Knoxville, TN

Paul Plassmann
Argonne National Laboratory
Argonne, IL

SUMMARY

To enable the solution of large-scale applications on distributed memory architectures, we are designing and implementing parallel algorithms for the fundamental tasks of unstructured mesh computation. In this paper, we discuss efficient algorithms developed for two of these tasks: parallel adaptive mesh refinement and mesh partitioning. The algorithms are discussed in the context of two-dimensional finite element solution on triangular meshes, but are suitable for use with a variety of element types and with $h-$ or $p-$refinement. Results demonstrating the scalability and efficiency of the refinement algorithm and the quality of the mesh partitioning are presented for several test problems on the Intel DELTA.

INTRODUCTION

Unstructured meshes have been used successfully in conjunction with finite element techniques to solve problems in a large number of application areas. Unfortunately, many of these applications are unable to take advantage of the power of parallel computing because of a lack of algorithms and portable software tools for distributed memory architectures. The PUMAA3D (Parallel Unstructured Mesh Algorithms and Applications) project will address this need by providing a publicly available, integrated software package for many important aspects of unstructured mesh computation. In particular, we are designing and implementing provably good, parallel algorithms in the following areas:

- **Mesh generation**: construction of meshes that satisfy user-specified properties over complex, irregular domains;
- **Mesh smoothing**: local adjustment of grid point position to improve the overall quality of the mesh;

[1]This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

- **Mesh refinement**: adaptive refinement and de-refinement of an initial mesh to accurately model rapidly changing solutions;
- **Domain partitioning**: decomposition of the mesh into equally sized, well-separated regions for distribution on multiple processor architectures; and
- **Linear system solution**: the assembly and solution of the linear systems generated by general, unstructured mesh problems.

We have made considerable progress in developing parallel, portable software in each of these areas and have already released the `BlockSolve` [5] software tool for solving large sparse linear systems. In this paper, we concentrate on the algorithms and software developed for the third and fourth components listed above: adaptive mesh refinement and domain partitioning.

Adaptive mesh refinement techniques are known to be successful in reducing the computational and storage requirements for solving a number of partial differential equations [7]. Much research has been done in this area, particularly in the development of sequential algorithms for refining simplicial meshes in two and three dimensions (see, for example, [1], [8], and [9]). Research on the corresponding parallel algorithms has just begun. We describe here an algorithm that uses independent sets to efficiently refine elements in parallel. This algorithm is suitable for use in two and three dimensions, with $h-$ or $p-$refinement, and with a variety of mesh element types.

Because adaptive mesh refinement is a dynamic process, it is often necessary to repartition the mesh after each modification to maintain load balance and good communication characteristics on parallel computers. We have developed a geometric partitioning algorithm that strives to minimize latency and transmission communication costs while evenly distributing the unknowns to the processors for load balance. Because the algorithm is geometric, the partitions are inexpensive to compute, and the algorithm requires only a small fraction of the total solution time. In addition, we have found that this algorithm is particularly effective for the smoothly varying meshes that typically arise in the solution of partial differential equations.

The remainder of the paper is organized as follows. We first describe the parallel refinement and partitioning algorithms. Then we present experimental results obtained on the Intel DELTA that demonstrate the effectiveness and efficiency of our algorithms for several test cases.

PARALLEL ADAPTIVE MESH REFINEMENT

One of the most attractive features of unstructured, simplicial meshes is the ease with which they may be adaptively refined to capture rapidly changing solutions in the numerical modeling of partial differential equations. One popular refinement technique is mesh enrichment, in which grid points are added or deleted from the mesh to increase or decrease accuracy in the numerical solution. Typically, one begins with an initial mesh and selectively adds grid points to regions in that mesh according to local error estimates. In this way, grid points are concentrated in areas where a high resolution is necessary to reduce error and placed more sparsely in other areas of the domain. In addition to appropriate placement of grid points, the mesh must meet several criteria if it is to be used in conjunction with the finite element discretization technique. Let T_0 be an initial triangulation conforming to some geometric domain and T_k be the triangulation corresponding to the *k-th* refinement iteration. Then for $k = 0, 1, 2, ...$, T_k must be conforming, T_k must be graded or smooth, and the angles in T_k must be bounded away from 0 and π.

Several techniques for adaptive refinement on simplicial meshes meet the requirements given above to produce valid finite element meshes (see [7] for an overview). The technique that we focus on in this paper is Rivara's two-dimensional bisection algorithm [9]. In this algorithm, a triangle marked for refinement is divided by connecting the midpoint of the longest side to the opposite vertex. This process creates a nonconforming point in a neighboring triangle. The refinement is then propagated until all nonconforming points are removed from the mesh (see Figure 1 for an illustration). Rivara has shown that this propagation will terminate in a finite number of iterations, L_P. In addition, the algorithm guarantees that the angles of T_k are bounded away from 0 and π if the angles in T_0 are. In particular, $\theta^k_{min} \geq \frac{1}{2}\theta^0_{min}$ [10].

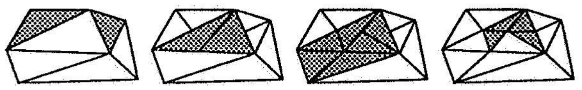

Figure 1: Propagation within the bisection algorithm

To implement Rivara's bisection algorithm on medium grain parallel architectures such as the Intel DELTA, we partition the vertices of the initial mesh and distribute them across the processors. For example, in Figure 2 partition boundaries are indicated by dashed lines. The processor assigned the center partition is responsible for the storage and computation relating to the vertices and triangles indicated by the black dots and shaded regions, respectively. In addition, this processor stores the nearest neighbor information in the finite element mesh, indicated by the clear dots and clear triangles in the figure.

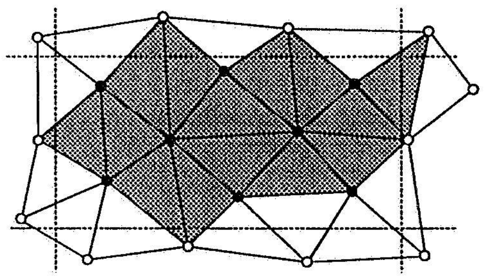

Figure 2: Partitioning of vertices and triangles across processors

One critical aspect of adaptive mesh refinement on distributed memory computers is the management of synchronization points so that global mesh information is correct. There are two ways in which this information can be incorrect: (1) two different processors can create vertices in the same location so that the global vertex list is not unique, and (2) outdated element neighbor information can be used if neighboring processors are not notified of refined elements on processor boundaries. To ensure that data is not corrupted in this manner, we select a near-maximal subset of triangles that can be refined simultaneously on different processors. These subsets are known as independent sets and are defined in the context of the dual graph of the mesh. The dual graph is defined to be $D = (T, F)$, where T is the set of triangles in the mesh and F is the set of edges that connect two triangles if they share a common edge. We say that a triangle, t_i, is in the independent set, I, if for every neighboring triangle $t_j \in D$: t_j is not marked for refinement, t_j is owned by the same processor as t_i, and $\rho(t_j) < \rho(t_i)$, where $\rho(t)$ is a random number assigned to the triangle at its creation. We note that finding I requires no communication, since each processor stores the triangle neighbor information.

Using independent sets, we now describe an algorithm that avoids the synchronization problems mentioned above and has a provably good runtime (for a complete description of this algorithm see [4]).

```
$l = 0$
Based on local error estimates, let $Q_0$ be the
set of triangles initially marked for refinement
**While** $Q_l \neq \emptyset$ **do**
    Choose $I_l \in D$ from $Q_l$
    Simultaneously refine the triangles in $I_l$
    Distribute updated element information
    $l = l + 1$
    $Q_l$ is the set of new nonconforming triangles
    $Q_l = Q_l \cup (Q_{l-1} - I_{l-1})$
**Endwhile**
```

The only communication required in this algorithm is the distribution of updated element information to the processors and the global reduction required to check whether Q_l is empty. Notice that the parallel refinement algorithm is not restricted to Rivara's bisection algorithm. Independent sets may be used successfully with a number of different refinement techniques including techniques for $p-$refinement, nonsimplicial meshes, and higher dimensions.

Jones and Plassmann [4] show theoretically that no two vertices will be created at the same position and that neighbor information in the dual graph is correctly updated. In addition, a P-RAM version of this algorithm is given whose expected runtime is $\mathcal{EO}(\frac{\log Q_{max}}{\log\log Q_{max}}) \times L_P$, where $Q_{max} = \max_l \mid Q_l \mid$ and L_P is the number of levels of propagation. This result implies that the running time is a *very* slowly growing function of the number of vertices, and thus the algorithm can be expected to scale well, as is shown in the Results section.

UNBALANCED RECURSIVE BISECTION

As grid points are dynamically added and deleted in an adaptive mesh, we must recalculate the partitioning of vertices across the processors of a distributed memory architecture. The quality of a partitioning is related to the equity of work assigned to the processors and the cost of communicating data among processors. In particular, for finite element meshes we use the following measures to determine the quality of a partition: the degree of imbalance between the sizes of the partitions, the maximum number of partition neighbors, and the maximum number of edges cut in the finite element mesh. The relative importance of these measures is dependent on the computer architecture and problems considered.

One partitioning algorithm that is effective for the meshes that typically arise in finite element calculations is orthogonal recursive bisection (ORB) [2]. The vertices of the mesh are partitioned according to their physical coordinates in the computational domain. An initial cut is made to divide the grid points in half. Orthogonal cuts are then made recursively in the new subdomains until the grid points are evenly distributed among the processors. This algorithm has the advantages of ease of implementation, inexpensive execution costs, and ease of parallelization. However, it also yields partitionings in which the maximum number of neighbors of any partition is $\mathcal{O}(\sqrt{p})$, where p is the total number of processors [2]. That is, the maximum number of messages that a processor may have to send is dependent on the total number of processors thereby implying a lack of scalability.

To address this problem, we have developed a modification of ORB which we call unbalanced recursive bisection (URB). Let the partition aspect ratio, a_p, be given by $\max(\frac{h}{w}, \frac{w}{h})$, where h is the height of the partition and w is the width. Instead of dividing the unknowns in half, we choose the cut that minimizes a_p and divides the unknowns into $\frac{nk}{p}$ and $\frac{n(p-k)}{p}$ sized groups, where n is the total number of unknowns and $k \in \{1, 2, ..., p-1\}$. Like the ORB algorithm, this algorithm is geometric in nature and hence is easy to implement, inexpensive to execute, and easy to parallelize. Unlike the ORB algorithm, this algorithm does not require that orthogonal cuts be made at each step. That is, we choose the cut that minimizes a_p regardless of the direction of the previous cut. This modification results in improved partitionings for which it can be shown that all of the above criteria can be bounded independently of p for smoothly varying finite element meshes. Hence, the URB algorithm yields scalable partitionings. For a complete description of this work, including proofs of the partition bounds, see [3]. In Figure 3, we show the resulting ORB and URB partitions for a smoothly varying mesh where the densest portion of the mesh is in the lower right i corner. Both algorithms generate partitions with an evenly distributed load. However, the ORB algorithm yields partitions with high aspect ratios which tend to have a large number of partition neighbors. In contrast, the partitions generated by the URB algorithm tend to be square and have fewer partition neighbors.

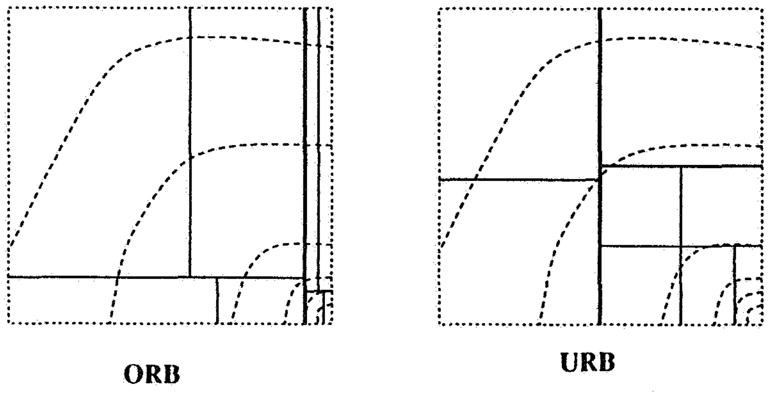

Figure 3: A comparison of ORB partitioning and URB partitioning.

EXPERIMENTAL RESULTS

To demonstrate the effectiveness and efficiency of the parallel refinement and partitioning algorithms, we consider a variety of large scale applications on the Intel DELTA. We use triangular meshes with linear finite elements to solve the following three partial differential equations.

Test Problem 1: (POISSON) Our first test problem arises from Poisson's equation,

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y) \text{ in domain } S \tag{1}$$

$$u = 0 \text{ on boundary} \tag{2}$$

on a square domain, where $f(x,y)$ is a Gaussian charge distribution which forces refinement around a point (S_x, S_y). We move the point (S_x, S_y) several times and find a new solution/mesh from the old solution/mesh. This movement requires mesh refinement around the new position and definement around the old position while the rest of the mesh remains nearly constant.

Test Problem 2: (ELASTIC) We solve the elasticity equations for the plane stress problem given here (without inclusion of the load):

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1+\nu}{2}\left(\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 v}{\partial x \partial y}\right) \tag{3}$$

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = \frac{1+\nu}{2}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y}\right). \tag{4}$$

These equations are solved on an annulus with a load placed on the edges of the domain. In each case above, we selectively refine the mesh until the local error estimate at each triangle is acceptable.

Test Problem 3: (SUPER) Our final test problem arises in the study of the internal structures and configurations of the vortices found in high-temperature superconductors. The model used in these calculations is the nondimensionalized Ginzburg-Landau free energy functional given by

$$F(\mathbf{u}) = F(\psi, \mathbf{A}) = F_{\text{cond}}(\psi) + F_{\text{kin}}(\psi, \mathbf{A}) + F_{\text{fld}}(\mathbf{A}), \tag{5}$$

where $\psi = a + ib$ is the complex-valued order parameter and $\mathbf{A} = [A_x,\ A_y]$ is the vector potential. These three terms are generally known as the condensation, kinetic, and field energy terms and are given by the formulae

$$F_{\text{cond}} = \int_\Omega -|\psi|^2 + \frac{1}{2}|\psi|^4 d\Omega, \tag{6}$$

$$F_{\text{kin}} = \int_\Omega |(\nabla + i\mathbf{A})\psi|^2 d\Omega, \tag{7}$$

$$F_{\text{fld}} = \int_\Omega \kappa^2 |\nabla \times \mathbf{A}|^2 d\Omega. \tag{8}$$

These equations are solved on a rectangular domain, which we assume is far from the boundaries of the physical sample so that magnetic periodic boundary conditions may be used. The mesh is refined by proximity to the vortex core singularities.

The results of four typical runs for each of the test problems are shown in the table below. The number following each test case name gives the number of Intel DELTA processors used in the trial. We have constructed the problem sets so that the final solution mesh for each successive problem has roughly twice as many vertices/triangles as in the previous problem. In all three cases, we solve the linear systems arising from the finite element approximations using the parallel conjugate gradient method preconditioned by an incomplete factorization available in the `BlockSolve` software [5] [6].

To show the efficiency of the refinement and partitioning algorithms, the maximum time required to perform these operations is given as a percentage of the total solution time. As a comparison, we also include the percentage of total time required to solve the resulting linear systems. The refinement and partitioning operations required five percent or less of the total execution time in all cases and the solution of the linear systems dominates the cost of the calculation. The time not accounted for in these tables is problem initialization and setup, element evaluation, and linear system assembly.

Problem	Number of Elements	Percent Refine Time	Percent Partition Time	Percent Solution Time	Average Adjacent Partitions	Percent Cross Edges
POISSON16	40292	.795	.428	48.1	4.63	2.38
POISSON32	80116	.856	.516	60.8	5.06	2.60
POISSON64	159758	.647	.731	59.0	5.53	2.70
POISSON128	318796	.568	1.03	60.1	5.64	2.78
ELASTIC16	21043	.697	1.15	98.2	4.00	3.56
ELASTIC32	42049	.560	1.29	98.1	4.38	3.99
ELASTIC64	82997	.449	1.30	98.2	4.75	4.26
ELASTIC128	165468	.268	1.23	98.5	5.20	4.27
SUPER16	30484	.229	.193	69.5	5.31	6.72
SUPER32	48416	.091	.117	86.3	5.40	8.32
SUPER64	111660	.087	.167	88.8	5.64	10.0
SUPER128	196494	.181	.452	86.0	5.71	13.7

To show the quality of the partitions generated by the URB heuristic, we have also included statistics on the partitionings for each of the test cases. The average number of adjacent partitions gives the average number of messages that the processors are sending during each step of the solution process. In all cases, the average ranged from 4 to 6 neighbors; and although the results are not included in the table, the maximum number of partition neighbors ranged from 7 to 9. In the final column we show the maximum final percentage of edges of the finite element mesh cut by a partition boundary to the total number of edges in the partition. Recall that this indicates the message volume each processor is required to transmit in the solution of the partial differential equations. These percentages are quite low for the first two test problems and slightly higher for the final test problem. This reflects the fact that the meshes for the first two problems are much more smoothly varying than in the final case.

CONCLUSIONS

We have described scalable, efficient parallel algorithms for the adaptive refinement and partitioning of finite element meshes. Work is currently under way to extend these algorithms to three-dimensional meshes and higher order elements. In addition, we are developing parallel algorithms for mesh generation and mesh smoothing. This software will be integrated with the software described in this paper and with `BlockSolve` to form a complete package for parallel solution of finite element problems on simplicial meshes.

REFERENCES

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

[2] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5), 1987.

[3] Mark Jones and Paul Plassmann. Bounds on partition quality for orthogonal recursive bisection. Technical report, University of Tennessee, *in preparation*, 1994.

[4] Mark T. Jones and Paul E. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In *Scalable High Performance Computing Conference*, Knoxville, Tennessee, May 1994.

[5] Mark T. Jones and Paul E. Plassmann. BlockSolve v1.0: Scalable library software for the parallel solution of sparse linear systems. ANL Report ANL-92/46, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992.

[6] Mark T. Jones and Paul E. Plassmann. Scalable iterative solution of sparse linear systems. *Parallel Computing*, 20:753–773, 1994.

[7] W. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. *ACM Transactions of Mathematic Software*, 15(4):326–347, 1989.

[8] R. V. Nambiar, R. S. Valera, K. L. Lawrence, Robert B. Morgan, and David Amil. An algorithm for adaptive rerfinement of triangular element meshes. *International Journal for Numerical Methods in Engineering*, 36:499–509, 1993.

[9] M. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21:604–613, 1984.

[10] I. Rosenberg and F. Stenger. A lower bound on the angles of triangles constructed by bisecting the longest side. *Mathematics of Computation*, 29:390–395, 1975.