PARALLEL ALGORITHMS FOR UNSTRUCTURED MESH COMPUTATION *

LORI A. FREITAG, MARK T. JONES, AND PAUL E. PLASSMANN†

Abstract. The efficient solution of many large-scale scientific calculations depends on unstructured mesh strategies. For example, problems where the solution changes rapidly in small regions of the domain require an adaptive mesh strategy. In this paper we discuss the main algorithmic issues to be addressed with an integrated approach to solving these problems on massively parallel architectures. We review new parallel algorithms to solve two significant problems that arise in this context: the generation of the adaptive mesh and the mesh partitioning. The gist of our refinement algorithm is the identification of independent sets of elements that can be refined in parallel. The objective of our partitioning heuristic is to construct partitions with good aspect ratios.

We present running time bounds and computational results obtained on the Intel DELTA for these algorithms used in solving an optimization problem to determine the vortex structure in a high-temperature superconductor. These results demonstrate that the algorithms exhibit scalable performance and have runtimes small in comparison with other aspects of the computation.

1. Introduction. Unstructured mesh strategies have proven to be very successful in reducing the computation and storage requirements for many scientific and engineering calculations [14]. Massively parallel computers offer a cost-effective tool for solving such problems. However, many difficult algorithmic and implementation issues must be addressed to make effective use of this resource. In this paper, we review the major aspects of an unstructured mesh strategy and present an integrated approach to deal with these aspects on distributed memory machines. We also present computational results from a preliminary implementation of this approach.

It has often been observed that the dominant computational cost in unstructured mesh calculations is the solution of the sparse linear systems derived from this mesh. For this reason, much effort has been invested in developing the parallel algorithms and software for general, sparse linear systems. For example, the BlockSolve package has been developed for the iterative solution of symmetric systems [11], the CAPSS project has developed software for direct methods [10], and PETSc contains parallel iterative methods for nonsymmetric systems [9].

However, the nonnumeric phases of generating, refining, and partitioning unstructured meshes must be addressed to use massively parallel machines in a coherent manner. We have identified the following basic problems that are fundamental to computation on unstructured domains:

- Mesh generation: constructing meshes that satisfy user-specified properties over irregular domains;
- Mesh refinement: adaptive refinement and de-refinement of an initial mesh to accurately model rapidly changing solutions;
- Domain partitioning: partitioning graphs/geometries into equally sized, well-separated regions; and
- Linear system solution: the assembly and solution of the linear systems generated by general, unstructured mesh problems.

In the following sections we give an overview of the current state-of-the-art in these tasks. We examine the performance on the Intel DELTA of the new methods we have developed for adaptive mesh calculations. The problem we consider is the determination of the minimum energy configuration of vortices in a high-temperature superconductor model — a large, unconstrained optimization problem.

2. Mesh generation. The first point given above, mesh generation, can be an extremely involved and application dependent process. In spite of the fact that there is a tremendous amount of interest in this area, very little work has been done in the development of parallel algorithms to solve this problem.

Possible approaches to parallel unstructured mesh generation include conformal mappings of regular meshes to more complicated domains [4] and the use of coarse background grids to partition the domain across processors [13]. However, this work fails to address the manner by which many geometric models (upon which the finite element mesh is defined) are specified. Most geometric models are defined implicitly by parametric descriptions, and only very preliminary work with parallel implementations have been done in this area. The most promising general-purpose approach for parallel implementation

* This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

† The address of the first and third authors is Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439. The address of the second author is Computer Science Department, University of Tennessee, Knoxville, TN 37996.

seems to be quadtree/octree mesh generation methods [18], but this topic remains an open area for future research.

3. Adaptive mesh refinement. Rather than using a structured mesh with grid points evenly spaced on a domain, adaptive mesh refinement techniques place more grid points in areas where the solution is changing rapidly. The mesh is adaptively refined and de-refined during the computation according to local error estimates. This technique is much more efficient than the use of structured meshes when the solution is changing much more rapidly in some areas than in others.

Many researchers have examined the adaptive construction of these nonuniform meshes. Typically, one begins with an initial mesh and selectively refines that mesh based on local error estimates until a final mesh is constructed that satisfies an error tolerance. Most research has focused on simplicial meshes: meshes composed of line segments in one dimension, triangles in two dimensions, or tetrahedra in three dimensions. We consider two-dimensional simplicial meshes; however, the algorithms and analyses we present are applicable to other dimensions and to nonsimplicial meshes.

In this paper we consider adaptive refinement of triangular meshes by simple bisection. Other possible approaches, and more detail of the following algorithms, are given in [6]. Simple bisection has excellent properties; it generates conforming, graded meshes that preserve the element quality of the initial mesh.

For a mesh to be conforming, we require that the intersection of any two triangles in the mesh is a single vertex, a line segment connecting two vertices, or the empty set. In addition, we require the mesh to be graded, that is, adjacent triangles should not differ dramatically in area. Finally, we require that all angles in the mesh be bounded away from 0 and π. The latter requirement is necessary because the discretization error in a finite element approximation has been shown to grow as the maximum angle approaches π [1]. We would like to avoid small angles because the condition number of the matrices arising from mesh elements has been shown to grow as $O(\frac{1}{\theta_{\min}})$, where $\theta_{\min}$ is the smallest angle in the mesh [7].

3.1. A parallel bisection algorithm. The bisection algorithm bisects triangles across the largest edge (dividing the largest angle) with selective divisions across a smaller edge (termed simple bisection). This has been shown to yield triangulations whose smallest angle is bounded by at worst one half the smallest angle in the inital mesh [17]. The algorithm is given in Figure 1.

$i = 0$
Q_i = the set of triangles marked for refinement
$R_i = \emptyset$
while $(Q_i \cup R_i) \neq \emptyset$ **do**
    bisect each triangle in Q_i across its longest edge
    bisect each triangle in R_i across a nonconforming edge
    all incompatible triangles embedded in Q_i are placed in R_{i+1}
    all other incompatible triangles are placed in Q_{i+1}
    $i = i + 1$
endwhile

FIG. 1. *The bisection algorithm*

Obviously, the refinement could propagate through many initially unmarked triangles before finishing. Rivara, however, has shown that this loop will terminate in a finite number of iterations, say L_P iterations [16]. Rivara also shows that each triangle in the resulting compatible mesh, T_{i+1}, embeds 1, 2, 3, or 4 triangles of T_i. We show the possible 2, 3, or 4 triangle results in Figure 2. During the execution of the algorithm, no side of a triangle will have more than one nonconformity. We give an example of the propagation in Figure 3.

The refinement algorithm is formulated mainly within the context of the dual graph to the mesh, which we define as follows. Let $V = \{v_i \mid i = 1 \ldots n\}$ be the set of vertices in the mesh and $T = \{t_a \mid a = 1 \ldots m\}$ be the set of triangles. Let $G = (V, E)$ be the graph associated with the mesh where $E = \{e_{i,j} = (v_i, v_j) \mid v_i, v_j \in t_a\}$. Let $D = (T, F)$ be the dual graph associated with the mesh where $F = \{(t_a, t_b) \mid e_{i,j} \in t_a, t_b\}$.

For our initial discussion assume that we have as many processors as we have triangles, and that

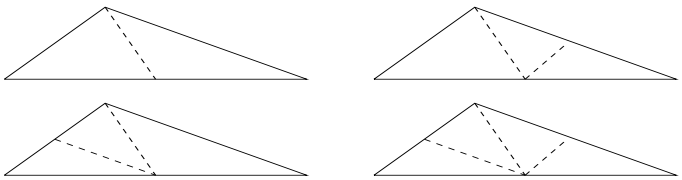

FIG. 2. *The possible divisions of a single triangle in the bisection algorithm*

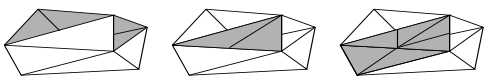

FIG. 3. *From left to right, the process of the bisection algorithm. In the initial mesh the shaded triangles are refined; subsequently the shaded triangles are refined because they are not compatible.*

t_a is assigned to processor p_a. Each processor, p_a, must keep track of the neighbors of t_a in D. Synchronization in the algorithm must be managed so that this neighbor information is correct. The management of the neighbor information in G for the refinement algorithm is straightforward and will not be discussed here. To keep our data structures coherent, we require that two different processors not create vertices at the same location when bisecting a triangle on their processor. For example, in Figure 4 we see two processors creating two copies of the vertex V at the same location. In the same figure, we see a possibility for outdated neighbor information to be propagated; triangle U_1 may believe that triangle W is its neighbor rather than triangle W_1 if triangles U and W are simultaneously refined.

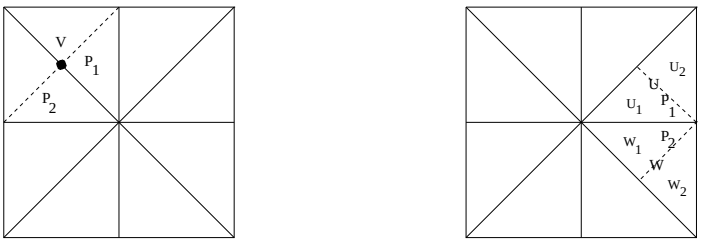

FIG. 4. *On the left, two processors creating a vertex at the same location; on the right, a possible corruption of neighbor information*

To avoid these synchronization problems, we determine a sequence of independent sets of triangles in the dual graph and refine the triangles in these sets in parallel. The complete algorithm, given in [6], takes into account additional triangles to be refined to obtain a compatible mesh. The crux of the algorithm is the Monte Carlo rule used to determine the independent set. An independent set, I, is chosen at step i by the rule: $t_a \in I$ if for each of its neighbors, t_b, in D, if (a) t_b not $\in Q_i \cup R_i$ or (b) $\rho(t_a) > \rho(t_b)$, where the $\rho(t)$ are independent random numbers. The following bound can be obtained for the expected running time of this algorithm on a P-RAM. Our implementation is based on extension of this algorithm to distributed memory computers and is fully described in [6].

THEOREM 3.1. *This algorithm terminates in a finite number of steps and has an expected runtime on a P-RAM of* $EO(\frac{\log Q_{max}}{\log \log Q_{max}}) \times L_P$ *where* $Q_{max} = \max_i \mid Q_i \mid$ *and* L_P *is the number of levels of propagation.*

Proof: The complete proof is given in [6]. □

4. Mesh partitioning. As grid points are adaptively added to and deleted from the mesh, we must determine good partitionings of these points onto processors. For our purposes a good partition ensures that grid points are evenly distributed to the processors in way that minimizes interprocessor communication costs. We may minimize the latency and transmission communication costs by respectively minimizing the number of partition neighbors and the number of links crossing the partition boundary. For uniform meshes a good partitioning of grid points may be determined *a priori* by simple constructions. However, for unstructured, adaptive meshes the partitioning cannot be predetermined because it changes with each new refinement of the mesh.

Several interesting techniques have been proposed to determine partitionings of unstructured meshes. Spectral methods [15] have the advantage of global access to information about the graph to find good separators at the cost of eigenvalue/eigenvector computations. Although the eigenvectors generally do not need to be found to great accuracy, spectral methods fail to utilize the geometric information inherent to the mesh, which may be used to significant advantage. This geometric information is used in bisection partitioning algorithms such as the orthogonal recursive bisection (ORB) algorithm [2]. This algorithm makes an initial geometric cut 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. Although this algorithm obtains good load balancing, it ignores the communication minimization problem. As a result, long, thin partitions may be created that have a high ratio of links crossing the partition boundaries to the total number of links in the partition. These large ratios lead to a high ratio of communication to computation.

To address this problem, we have developed a modification of ORB that we call the unbalanced recursive bisection (URB) algorithm. Instead of dividing the unknowns in half, we choose the cut that minimizes partition's geometric aspect ratio and divides the unknowns into $\frac{nk}{p}$ and $\frac{n(p-k)}{p}$ size groups, where n is the total number of unknowns, p is the number of processors, and $k \in \{1, 2, ..., p-1\}$. Again, this algorithm is applied recursively. This algorithm leads to an even distribution of grid points with more balanced geometric aspect ratios for the resulting partitions. This fact minimizes the communication costs in two ways. First, partitions with good aspect ratios (close to one) tend to have fewer partition neighbors and hence fewer total messages to send. Second, the percentage of mesh links crossing the partition boundary to the total number of links in the nearly square partitions is small compared with the long, thin partitions generated by the ORB algorithm. Thus, the ratios of computation to communication are increased compared with the ORB algorithm. Finally, the execution time for this approach is significantly less than for the spectral techniques.

5. Computational results. In this section we use the approach described above to solve an unconstrained optimization problem based on an unstructured mesh. The problem is to determine the minimum energy configuration of the Ginzburg-Landau free energy functional, a phenomenological model for high-temperature superconductivity. The problem and methods used are described in [12]. A complete description of the finite-element formulation used and the refinement criteria are given in [5].

We note that this problem is complicated by the invariance of the free-energy functional under gauge transformations; thus a local minimizer is not unique. This degeneracy (the singularity of the Hessian at a solution) significantly complicates the computation of such a minimizer. We have found that an effective approach to computing a minimizer of the free-energy functional is a damped Newton's method [8]. Each iteration of the nonlinear method requires computation of the gradient vector and Hessian matrix. We have used the automated differentiation package ADIFOR [3] to compute these derivatives for the element function. The contributions from the elements are assembled to obtain the total gradient and Hessian.

The computational kernel of this technique is the solution of the damped Newton system — a large, sparse linear system of equations. We do not explicitly invert this system but use the iterative solver from the BlockSolve package [11] to obtain an approximate (inexact) solution.

To demonstrate the efficiency and scalability of the refinement and partitioning algorithms for the superconductivity problem, we have increased the problem sizes in proportion to the number of processors used. The results of four typical runs are shown in Table 1, where P gives the number of processors and E indicates the number of triangular elements in the final solution mesh. The number of vortices in each sample are 32, 48, 64, and 72 for 16, 32, 64, and 128 processors, respectively. We indicate the amount of time required for refinement and partitioning as a percentage of total solution time. We see that these operations require less than one percent of the execution time in all cases.

Statistics on the partitions generated by the new geometric partitioning algorithm, URB, are given in Table 2. The average aspect ratio for the partitions is less than two in all cases, and the maximum aspect ratio is less than 3.6. These result in a partition quotient graph whose average degree is between five and six, which corresponds to an average of five to six messages sent per processor to transfer nearest neighbor information. Finally, to estimate the amount of data that must be transferred between processors, we consider the percentage of edges that cross partition boundaries to the total number of edges in the partition. This number is less than 15 percent in all cases.

TABLE 1
Timing results for the superconductivity problem on 16–128 processors of the Intel DELTA

P	E	Percent Refine Time	Percent Partition Time	Percent Setup Time	Percent Solution Time
16	30484	.229	.193	30.0	69.5
32	48416	.091	.117	13.5	86.3
64	111660	.087	.167	10.8	88.8
128	196494	.181	.452	13.3	86.0

TABLE 2
Partition statistics for the superconductivity problem on 16–128 processors of the Intel DELTA

P	Avg. Graph Degree	Max. Graph Degree	Avg. Aspect Ratio	Max. Aspect Ratio	Percent Cross Edges
16	5.31	7.00	1.47	2.88	6.72
32	5.40	8.00	1.89	3.55	8.32
64	5.64	8.00	1.34	2.49	10.0
128	5.71	9.00	1.81	3.55	13.7

REFERENCES

[1] I. BABUŠKA AND A. K. AZIZ, *On the angle condition in the finite element method*, SIAM Journal of Numerical Analysis, 13 (1976), pp. 214–226.

[2] M. BERGER AND S. BOKHARI, *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Transactions on Computers, C-36 (1987).

[3] C. BISCHOF, A. CARLE, G. CORLISS, A. GRIEWANK, AND P. HOVLAND, *ADIFOR: Generating derivative codes from Fortran programs*, Scientific Programming, 11 (1992), pp. 11–29.

[4] J. E. CASTILLO, *Mathematical Aspects of Grid Generation*, Society for Industrial and Applied Mathematics, Philadelphia, 1991.

[5] L. A. FREITAG, M. T. JONES, AND P. E. PLASSMANN, *New advances in the modeling of high-temperature superconductors*, in 1994 International Simulation Conference – Grand Challenges in Computer Simulation, La Jolla, California, April 11-15, 1994.

[6] ———, *Parallel algorithms for adaptive mesh refinement*, Preprint MCS-P421-0394, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1994.

[7] I. FRIED, *Condition of finite element matrices generated from nonuniform meshes*, AIAA Journal, 10 (1972), pp. 219–221.

[8] J. GARNER, M. SPANBAUER, R. BENEDEK, K. STRANDBURG, S. WRIGHT, AND P. PLASSMANN, *Critical fields of Josephson-coupled superconducting multilayers*, Physical Review B, 45 (1992), pp. 7973–7983.

[9] W. D. GROPP AND B. F. SMITH, *Simplified Linear Equation Solvers Users Manual*, Tech. Rep. ANL-93/8, Argonne National Laboratory, Mar. 1993.

[10] M. T. HEATH AND P. RAGHAVAN, *Distributed solution of sparse linear systems*, Tech. Rep. UIUCDCS-R-93-1793, University of Illinois, Feb. 1993.

[11] M. T. JONES AND P. 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.

[12] ———, *Computation of equilibrium vortex structures for type-II superconductors*, The International Journal of Supercomputer Applications, 7 (1993), pp. 129–143.

[13] R. LÖHNER, J. CAMBEROS, AND M. MERRIAM, *Parallel unstructured grid generation*, Computer Methods in Applied Mechanics and Engineering, 95 (1992), pp. 343–357.

[14] W. F. MITCHELL, *A comparison of adaptive refinement techniques for elliptic problems*, ACM Transactions on Mathematical Software, 15 (1989), pp. 326–347.

[15] A. POTHEN, H. SIMON, AND K.-P. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis, 11 (1990), pp. 430–452.

[16] M.-C. RIVARA, *Mesh refinement processes based on the generalized bisection of simplices*, SIAM Journal of Numerical Analysis, 21 (1984), pp. 604–613.

[17] I. G. ROSENBERG AND F. STENGER, *A lower bound on the agnles of triangles constructed by bisecting the longest side*, Mathematics of Computation, 29 (1975), pp. 390–395.

[18] M. S. SHEPHARD AND M. K. GORGES, *Automatic three-dimensional mesh generation by finite octree technique*, Tech. Rep. SCOREC #1-1991, Scientific Computation Research Center, Rensselaer Polytechnic Institute, 1991.