Software for Parallel Adaptive Computation

J. E. Flaherty* J. D. Teresco†

June 13, 2000

Abstract

Reusable software tools for the solution of partial differential equations using parallel adaptive finite element methods have been developed. We describe the design and implementation of the object-oriented parallel mesh structures within the Rensselaer Partition Model (RPM). This hierarchical model is used to distribute finite element meshes and associated data on a parallel computer. It represents heterogeneous processor and network speeds, and may be used to represent processes in any parallel computing environment, including an SMP, a distributed-memory computer, a network of workstations, or some combination of these. The information about different processor speeds, memory sizes, and the corresponding interconnection network can be useful in a dynamic load balancing algorithm which seeks to achieve a good balance with minimal interprocessor communication penalties when a slow interconnection network is involved. An example of a partitioning scheme which takes advantage of this information is given.

Key words: adaptivity, finite element methods, dynamic load balancing, heterogeneous and hierarchical computers.

AMS subject classifications: 68 - Computer Science.

1 Introduction

Parallel, adaptive finite element methods (FEMs) are often used to achieve efficient numerical solutions to problems involving partial differential equations [6]. Portions of the finite element mesh may be refined or coarsened (h-refinement), be moved to follow evolving phenomena (r-refinement), or use methods of different order (p-refinement) to enhance resolution and efficiency. In addition to adaptivity, parallel computation is essential for solving large three-dimensional problems in reasonable times.

Reusable software libraries allow finite element problems to be solved without concern for complex underlying mesh structures, adaptive procedures, or parallelization, allowing an application programmer to concentrate on specific issues in the problem at hand. Parallel computation introduces complications such as the need to balance processor loading, coordinate interprocessor communication, and manage distributed data. The standard methodology for optimizing parallel FEM programs relies on a static partitioning of the mesh across the cooperating processors. This is insufficient in an adaptive computation, since load imbalance will be introduced by adaptive enrichment, necessitating dynamic partitioning and redistribution of data. Data structures must support this dynamic mesh migration.

We are developing such libraries to support parallel adaptive finite element computation [12, 13, 24]. The conventional array-based data representations used for fixed-mesh computation are not well suited for adaptivity using h- or p-refinement [1]. Alternative structures complicate the automatic (compiler) detection of parallelism. We opt

*Scientific Computation Research Center (SCOREC) and Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, flaherje@cs.rpi.edu

†Scientific Computation Research Center (SCOREC) and Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, terescoj@cs.rpi.edu

for the latter and describe (§2) software to manage distributed mesh data and to provide information about the computational environment by explicit parallelism achieved by message passing using the Message Passing Interface (MPI) [15]. Partitioning and dynamic load balancing algorithms distribute the computation across the processors by a domain decomposition of the (spatial or space-time) mesh. Traversal of the data must be efficient in all cases, but when adaptivity is introduced, modification of the mesh structures and corresponding solution data must also be efficient.

While parallel adaptive methods have been used successfully in the solutions of many problems [3, 4, 7, 8, 10, 13, 16], it is difficult to take full advantage of a particular parallel computing environment. A partitioning and dynamic load balancing strategy which is efficient in a shared-memory environment may not be in a distributed-memory environment (§3). This is especially the case with heterogeneous or hierarchical computers which combine multiprocessing shared memory nodes connected by networks of various speeds. A partitioner must have some knowledge of the parallel environment to achieve a good distribution.

2 Rensselaer Partition Model

The *Rensselaer Partition Model* (RPM) [23, 24] provides distributed mesh data structures and information about the parallel computational environment in which a program is executing. RPM is central to the *Meshing Environment for Geometry-based Analysis* (MEGA) [19] and to the *Trellis* framework [2]. The basic mesh data structures in RPM are provided by the *SCOREC Mesh Database* (MDB) [1]; however, many of the ideas may be applied to other systems. MDB includes operators to query and update a mesh data structure consisting of a full mesh entity hierarchy: three-dimensional *regions*, and their bounding *faces*, *edges*, and *vertices*, with bidirectional links between mesh entities of consecutive order. Regions serve as finite elements in three dimensions while faces are finite elements in two dimensions, or interface elements in three dimensions. The full entity hierarchy allows efficient mesh modification during h-refinement [21] and facilitates p-refinement [20] by allowing attachment of degrees of freedom to the mesh entities and by providing necessary geometric information. Mesh entities have an explicit *geometric classification* relative to a geometric (CAD) model of the problem domain. This allows the mesh to remain correct with respect to the geometry during h- or p-refinement. Mesh entities are stored with the geometry, so inverse classification information (retrieval of all mesh entities classified on a given model entity) is readily available. This is useful, for example, when applying a boundary condition on a model face. Rather than visiting all faces in the mesh and querying each to check if it is on the desired boundary, a list of the needed entities is traversed directly.

Each entity in a distributed finite element mesh is uniquely assigned to a *partition*. Each partition is assigned to a specific *process*, with the possibility that multiple partitions may be assigned to a single process. "Process" in this context refers to an address space. The model is hierarchical, with partitions assigned to a *process model* and processes assigned to a *machine model*.

The machine model represents the computational environment: the processing nodes and their network interconnections. The process model maps processes to the computer, and maps interprocess communication to inter-computer networks or, perhaps, to a shared-memory interface. Partitions know the mesh entities that they contain, and mesh entities know their partition assignments (*partition model classifications*).

In Figure 1, we show a sample two-dimensional mesh (a) and a target parallel environment consisting of two 2-way SMP workstations connected by a network (b). Figure 1(c) shows a partitioning of this mesh and the assignment of those partitions to the processes and machines of the target environment. Six partitions are created and assigned to four processes, since four processors are available. Two processes are assigned two partitions, while the other two are only assigned a single partition. The four processes are further assigned to the available machines: two to each.

Mesh entities are replicated only when on a partition boundary that is also a process boundary. Figure 2 shows two-dimensional examples of mesh faces which share a common edge across a partition boundary. The shared mesh edge is classified on the partition boundary in each case. On the left, the partition boundary is local to the process, so the mesh entity need not be replicated and is stored only with the partition boundary mesh. On the right, the partition boundary is also on the process boundary. This partition boundary is replicated in each process, so any mesh entity classified on this boundary must also be replicated.

3 Load Balancing Within RPM

Communication costs associated with interprocess boundary entities are useful in a partitioner. The availability of the information in the process and machine models allows a partitioner to avoid large communication volume on interprocess boundaries which communicate across a slow network. Many partitioners attempt to minimize the total

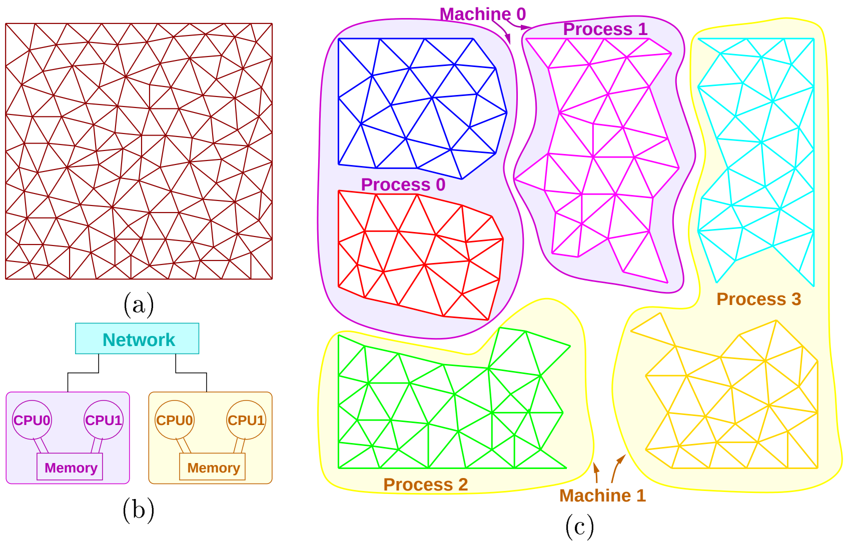


Figure 1: A sample two-dimensional mesh (a), target parallel environment in which the mesh is to be partitioned (b), and partitioning of the mesh and assignment to processes and machines for the parallel environment (c).

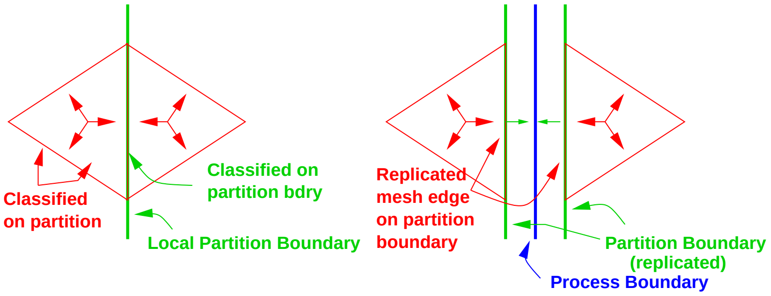


Figure 2: Partition classification of mesh entities at a same-process partition (left) and at a process-boundary partition (right).

size of the interprocessor boundaries, but that is not always equivalent to minimizing the actual communication cost. Like the computational costs of elements, the communication costs associated with boundary entities are not necessarily uniform.

Figure 4 shows four 16-way partitionings of a mesh used in a simulation of blood flow in human arteries [22]. The mesh contains 1,103,018 regions. The partitioning algorithms used are *parallel sort inertial recursive bisection* (PSIRB) [21] and the multilevel k-way parallel partitioner ParMetis [18]. We consider only two factors in partition quality for this example, although other factors [5] must be considered for a more thorough analysis. We consider computational balance, and two *surface index* measures. The *global surface index* (GSI) measures the overall percentage of mesh faces which are on interpartition boundaries, and the *maximum local surface index* (MLSI) measures the maximum percentage of interpartition boundary faces on any one partition. Partitioning using PSIRB (Figure 4a) achieves an excellent computational balance, with each partition assigned 68,938 or 68,939 regions, and reasonable surface indices with MLSI=1.77 and GSI=0.61. A partition such as this would be useful when the primary concern is a good computational balance, as in a shared-memory environment (Figure 3a). The partition using only ParMetis (Figure 4b) achieves excellent surface index values, MLSI=0.40 and GSI=0.20, but at the expense of a large computational imbalance. Here, the partition sizes range from 49,389 to 89,302 regions. For a computation running on a network of workstations (NOW) (Figure 3b), it may be worth accepting the significant load imbalance to achieve the smaller communication volume.

When executing on hierarchical systems, partition such as those shown in Figure 4c and Figure 4d may be desirable. Here, different combinations of ParMetis and PSIRB are used. Consider computational environments that consist of collections of SMP workstations connected by a relatively slow network. Two possible configurations are two 8-way

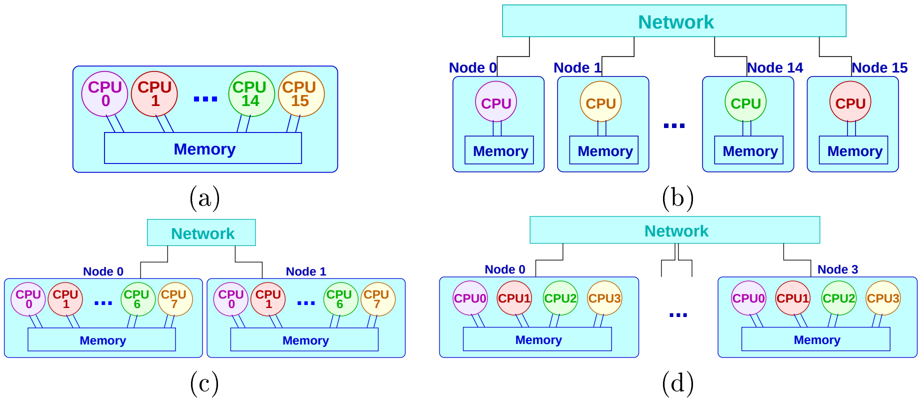


Figure 3: Examples of parallel computing environments. (a) A 16-way SMP workstation. (b) A 16-node computer with all uniprocessor nodes, connected by a network. (c) Two 8-way SMP workstations connected by a network. (d) Four 4-way SMP workstations connected by a network.

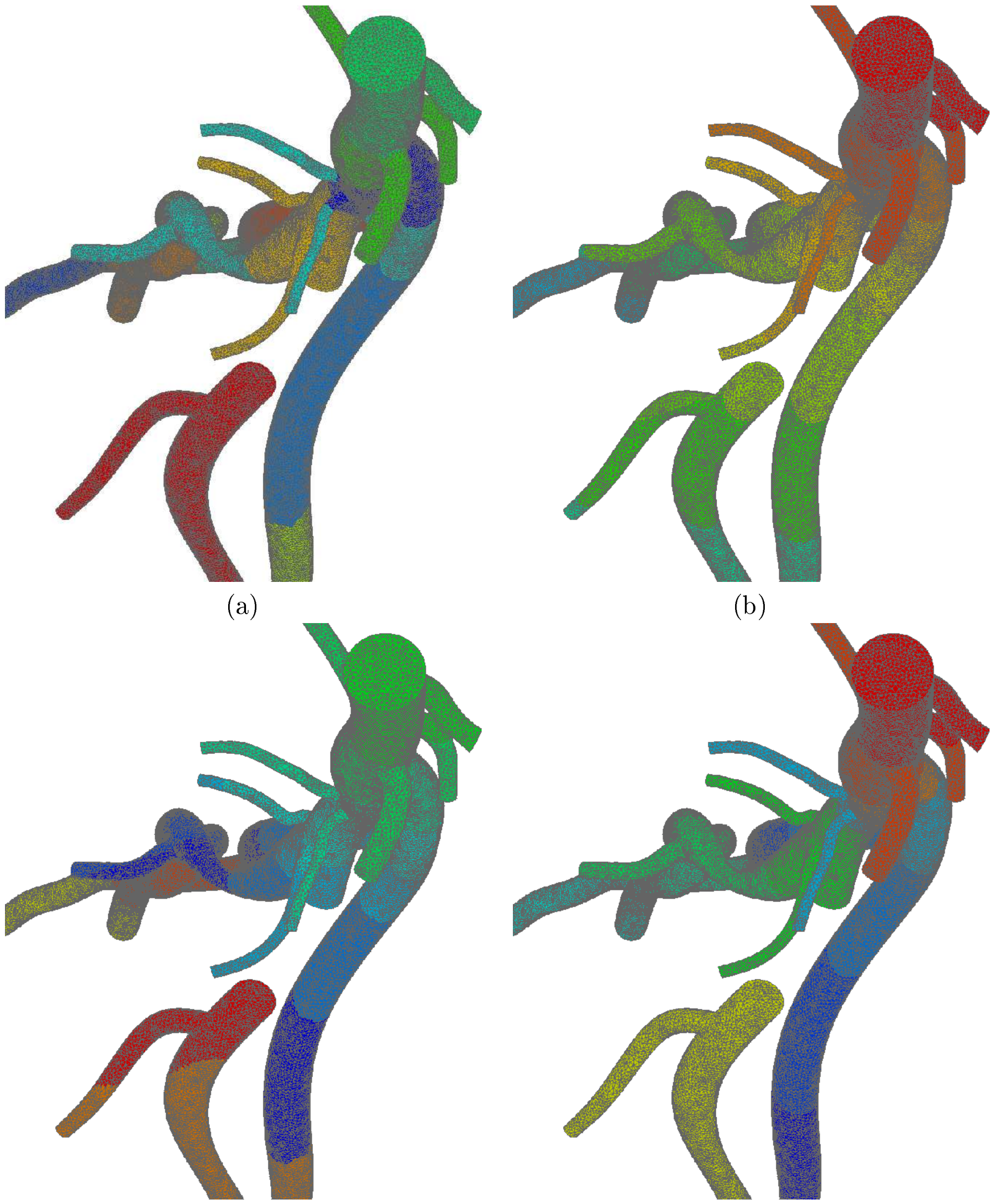
SMPs, or four 4-way SMPs. For the two 8-way node case (Figure 3c), ParMetis is used to divide the computation between the two SMP nodes, resulting in partitions of 532,063 and 570,955 regions, with MLSI=0.06 and GSI=0.03. Within each SMP, the mesh is partitioned eight ways using PSIRB, resulting in the partitioning of Figure 4c, where the partitions within each SMP are balanced to within one region, and the overall MLSI=1.88 and GSI=0.56. However, since the extra effort was made to minimize communication across the slow network, this will be a very effective partitioning for this environment. Similarly for the case of four 4-way SMPs (Figure 3d), ParMetis is used to divide the computation among the four SMP nodes, resulting in partitions of 265,897, 272,976, 291,207 and 272,938 regions with MLSI=0.23 and GSI=0.07. Within each SMP, the mesh is partitioned four ways using PSIRB, producing the partitioning of Figure 4d. Again, the partitions within each SMP are balanced to within one region, and the overall MLSI=1.32 and GSI=0.32.

A common operation when optimizing partitions is an interprocess boundary smoothing [9, 11, 13, 17] which exchanges elements across interprocess boundaries to decrease communication volume. This operation can provide significant improvement in surface index values, especially when applied to partitions using spatial cuts to introduce a partition boundary (PSIRB or an Octree-based partitioner [14, 16]). However, this improvement is achieved at the cost of the smoothing operation itself and the small load imbalances introduced. Depending on the parallel environment, this may or may not be worthwhile and represents an additional way of taking advantage of information provided by RPM.

The measurement of load imbalance in our existing algorithms is based on a weighted number of elements. Some sources of heterogeneity, such as the extra work on higher-order elements in adaptive p-refinement or the extra time steps taken on smaller elements in a local refinement method [14], can be accounted for using the weighted element scheme. Others, like the run-time imbalance caused by other processes in a nondedicated parallel environment, are not known *a priori* with respect to a given step and do not fit well into this model. Even when it is possible to estimate appropriate weights, there may be considerable effort involved. It will be possible to measure load imbalance dynamically by instrumenting the communication libraries to measure synchronization delays.

4 Discussion

We present the architecure of RPM: a hierarchical partition model to provide a distributed mesh data organization and to quantify information about the parallel computing environment. Within RPM, mesh data is organized to allow efficient traversals of entities of the entire mesh, a partition, or a geometric model entity on which mesh entities are classified. Each traversal is important during phases of a parallel adaptive computation. Support for multiple partitions within a process allows for a natural way to represent entities on partition boundaries and an elegant mesh migration strategy. The process and machine models provide information about the parallel execution environment. A previous study [24] showed execution times for a representative subset of our parallel adaptive finite element software on different computers and interconnection networks, indicating significant differences in performance with, *e.g.* , network bandwidth and latency. Partitioning algorithms that work well for one configuration may not be efficient for



(a) (b) (c) (d)

Figure 4: Examples of partitioning using additional knowledge of parallel environment. Partitioning using PSIRB (a), ParMetis (b), and multilevel combinations of PSIRB and ParMetis (c and d). Shading indicates processor assignments.

another. An example of a mesh partitioned by four different procedures on four different architectures illustrates this point. Further enhancements to existing partitioning algorithms and development of new algorithms becomes possible with the availability of the information from RPM.

RPM is capable of handling the heterogeneities introduced by p-refinement. All of the load balancing procedures include capablilites to weight mesh entitites. While procedures to handle communications hierarchies are in place (§2), these have to be examined more closely and extended to involve memory hierarchies (cache utilization).

Acknowledgments

The authors were supported by the National Science Foundation through grant ASC-9720227, by ARO grant DAAG55-98-1-0200, a grant from DOE Phase 1, ASCI number B341495, and by Simmetrix, Inc. Computer systems used include

the IBM SP systems at the Maui High Performance Computing Center, the IBM SP at Rensselaer, and the IBM SP Blue-Pacific at Lawrence Livermore National Laboratory.

References

[1] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *Int. J. Numer. Meth. Engng.*, 40(9):1573–1596, 1997.

[2] M. W. Beall and M. S. Shephard. A geometry-based analysis framework. In *Advances in Computational Engineering Science*, pages 557–562, Forsyth, GA, 1997. Tech. Science Press.

[3] R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.

[4] C. L. Bottasso, H. L. de Cougny, M. Dindar, J. E. Flaherty, C. Özturan, Z. Rusak, and M. S. Shephard. Compressible aerodynamics using a parallel adaptive time-discontinuous Galerkin least-squares finite element method. In *Proc. 12th AIAA Appl. Aero. Conf.*, number 94-1888, Colorado Springs, 1994.

[5] C. L. Bottasso, J. E. Flaherty, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. The quality of partitions produced by an iterative load balancer. In B. K. Szymanski and B. Sinharoy, editors, *Proc. Third Workshop on Languages, Compilers, and Runtime Systems*, pages 265–277, Troy, 1996.

[6] K. Clark, J. E. Flaherty, and M. S. Shephard. *Appl. Numer. Math., special ed. on Adaptive Methods for Partial Differential Equations*, 14, 1994.

[7] K. D. Devine and J. E. Flaherty. Parallel adaptive *hp*-refinement techniques for conservation laws. *Appl. Numer. Math.*, 20:367–386, 1996.

[8] M. Dindar, A. Z. Lemnios, M. S. Shephard, J. E. Flaherty, and K. E. Jansen. Adaptive solution procedures for rotorcraft aerodynamics. In *Proc. 16th Applied Aerodynamics Conference*, number AIAA Paper 98-2417. AIAA, 1998.

[9] P. Diniz, S. Plimpton, B. Hendrickson, and R. Leland. Parallel algorithms for dynamically partitioning unstructured grids. In D. Bailey, editor, *Proc. 7th SIAM Conference on Parallel Processing for Scientific Computing*, pages 615–620. SIAM, February 1995.

[10] H. C. Edwards. *A Parallel Infrastructure for Scalable Adaptive Finite Element Methods and its Application to Least Squares C^∞ Collocation.* PhD thesis, The University of Texas at Austin, May 1997.

[11] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. Numer. Meth. Engng.*, 36:745–764, 1993.

[12] J. E. Flaherty, M. Dindar, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. An adaptive and parallel framework for partial differential equations. In D. F. Griffiths, D. J. Higham, and G. A. Watson, editors, *Numerical Analysis 1997 (Proc. 17th Dundee Biennial Conf.)*, number 380 in Pitman Research Notes in Mathematics Series, pages 74–90. Addison Wesley Longman, 1998.

[13] 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. Numer. Math.*, 26:241–263, 1998.

[14] 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 Distributed Computing*, 47:139–152, 1997.

[15] Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee. *MPI: A Message Passing Interface Standard*, first edition, 1994.

[16] T. Minyard and Y. Kallinderis. Octree partitioning of hybrid grids for parallel adaptive viscous flow simulations. *Int. J. Numer. Meth. Fluids*, 26:57–78, 1998.

[17] T. Minyard, Y. Kallinderis, and K. Schulz. Parallel load balancing for dynamic execution environments. In *Proc. 34th Aerospace Sciences Meeting and Exhibit*, number 96-0295, Reno, 1996.

[18] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel diffusion algorithms for repartitioning of adaptive meshes. Tech. Report 97-014, University of Minnesota, Department of Computer Science and Army HPC Center, Minneapolis, MN, 1997.

[19] M. S. Shephard. Meshing environment for geometry-based analysis. *Int. J. Numer. Meth. Engng.*, 47(1–3):169–190, 2000.

[20] M. S. Shephard, S. Dey, and J. E. Flaherty. A straightforward structure to construct shape functions for variable p-order meshes. *Comp. Meth. in Appl. Mech. and Engng.*, 147:209–233, 1997.

[21] M. S. Shephard, J. E. Flaherty, C. L. Bottasso, H. L. de Cougny, C. Özturan, and M. L. Simone. Parallel automatic adaptive analysis. *Parallel Comput.*, 23:1327–1347, 1997.

[22] C. A. Taylor, T. J. R. Hugues, and C. K. Zairns. Finite element modeling of blood flow in arteries. To appear, *Comp. Meth. in Appl. Mech. and Engng.*, 1999.

[23] J. D. Teresco. *A Hierarchical Partition Model for Parallel Adaptive Finite Element Computation.* PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, 2000.

[24] J. D. Teresco, M. W. Beall, J. E. Flaherty, and M. S. Shephard. A hierarchical partition model for adaptive finite element computation. *Comput. Methods Appl. Mech. Engrg.*, 184:269–285, 2000.