1

THE QUALITY OF PARTITIONS PRODUCED BY AN ITERATIVE LOAD BALANCER

Carlo L. Bottasso, Joseph E. Flaherty*, Can Özturan*, Mark S. Shephard, Boleslaw K. Szymanski*, James D. Teresco*, Louis H. Ziantz*

Scientific Computation Research Center (SCOREC)
** and Department of Computer Science,*
Rensselaer Polytechnic Institute,
Troy, NY 12180 USA

ABSTRACT

We examine the quality of partitions produced by an iterative load balancer in parallel adaptive finite element calculations. We present several metrics which we use to evaluate the quality of a mesh partitioning, and report statistics generated from our analysis of adaptively refined meshes produced during the solution of computational fluid dynamics problems. Timings from the finite element solution phase for runs involving these meshes on 16 and 32 processors of an IBM SP2 are also presented.

1 INTRODUCTION

Adaptive finite element methods (FEMs) have gained importance based on their ability to offer reliable solutions to partial differential equations. In such an analysis, the computational domain is first discretized to create a mesh. During the solution process, portions of this mesh are recursively refined or coarsened (h-refinement) and/or the finite element approximation is varied in order (p-refinement) to improve the convergence rate and concentrate the computational effort in parts of the domain where the solution resolution is inadequate [3].

In order to solve large problems within a reasonable period of time, these methods have been implemented on parallel computers. The current methodology for optimizing parallel FEM programs relies on the initial partitioning of the meshes involved in the computation. However, in parallel adaptive codes, a good initial partitioning is not sufficient to assure high performance throughout the computation. Load imbalance caused by adaptive enrichment necessitates a dynamic redistribution of data. Since performing a global repartitioning of the entire mesh can be costly relative to the time spent in actual computation, a number of iterative dynamic load balancing techniques that incrementally migrate elements from heavily to lightly loaded processors have been proposed [2, 3, 5, 15]. While these methods have proven to be inexpensive and effective ways to achieve a balanced load, they can lead to degradation in the partition quality of the resulting subdomains. On a distributed-memory parallel computer, this generally results in a larger communication penalty during the finite element solution phase. The two pieces of information readily available to aid in the element-migration process are coordinates and local topological entity connectivity. Vidwans, *et al.* [15] experiment with using both types of information. Others [3, 5] use prioritized connectivity information, whereas our system [2] utilizes local connectivity information among various entities.

The primary goal of most mesh-partitioning algorithms is the minimization of the number of "cuts" that the subdomains create when the partitioning is viewed as a communication graph [7, 8, 14]. With many finite volume and finite element schemes this would closely correspond to the task of minimizing the number of element faces on interprocessor boundaries. Thus, the number of faces on the boundary or "surface" of each partition is related to the amount of non-local data that must be communicated to perform the computation. But, both iterative load-balancing and partitioning methods require other important measures of partition quality. In particular, we focus on three complementary measures: (*i*) different estimates of the number of faces on the boundary (normalized by the total number of faces), (*ii*) the spatial connectivity within partitions, and (*iii*) the degree of interconnection between partitions.

Our parallel adaptive FEM system is described in Section 2. In Section 3, we briefly discuss the characteristics of a "good" partition. Sections 4–6 provide a more detailed description of the various measures of partition quality. We present statistics and timings for meshes generated during an analysis of a compressible, inviscid flow about an Onera M6 wing in Section 7. In Section 8, we discuss future modifications to the load-balancing scheme that may improve partition quality.

2 EXISTING SYSTEM

The system we are using is built upon the *SCOREC Mesh Database* [1], which provides a hierarchical representation of a finite element mesh. It also includes a set of operators to query and update the mesh data structure. The basic hierarchy consists of three-dimensional *regions*, and their bounding *faces*, *edges*, and *vertices*. In our case, each region is a tetrahedral finite element.

A *Parallel Mesh Database* [2, 10] (PMDB) which provides operators to create and manipulate distributed meshes is built on top of the SCOREC Mesh Database. The package includes routines for partitioning using the Orthogonal Recursive Bisection (ORB) and the Inertial Recursive Bisection (IRB) algorithms [9] and for load balancing through iterative element migration. Once an adaptive FEM calculation begins, it consists of alternating phases of computation, mesh-refinement, and load balancing.

PMDB data structures and operators are used to perform an incremental load balancing. Balance is achieved through mesh migration, which occurs between processors that control neighboring spatial regions. Each processor requests load from its most heavily loaded neighboring processor. These requests are viewed as a tree, and a logarithmic time scan operation is used to compute load flows on this tree. The amount of data to be migrated is determined by these flows. Single layers of regions on interprocessor boundaries are moved from heavily loaded to lightly loaded processors based on the load to be migrated. This process is repeated until the load becomes balanced to within a tolerance, or until the maximum number of iterations has been performed [2, 10]. The iterative migration can also be run periodically with only a few iterations to help maintain balance between substages of an operation like refinement (cf. [2]) without a large run-time penalty.

The results that we present were generated using a parallel adaptive FEM program called *Fluid Analysis in Space-Time* (FAST) [2] which uses a Time Discontinuous Galerkin/Least-Squares method [12] and the Generalized Minimum Residual (GM-Res) [11] algorithm for linear systems to analyze compressible, inviscid fluid flows. It utilizes PMDB to perform parallel mesh operations. The code can do any number of solution steps on a mesh and can apply refinement and load balancing based on user-supplied parameters.

3 PARTITION QUALITY

We are interested in quantifying the notion of a "high quality" partition. The basic requirement of a good partition is that the computational load be balanced. If some processors have more work than others, then the lightly loaded processors will sit idle while the heavily loaded processors finish their computation. Another important measure is the interprocessor communication volume. We use metrics based on ratios of faces on interprocessor boundaries to total faces to indicate of the amount of interprocessor communication relative to on-processor data access. We also examine the number of geometrically disjoint components within each processor. Our final measure of partition quality is the number of processors that need to communicate with each other. This is a lower bound on number of message startups that will be required during the finite element computation, which can be significant when message startups are expensive relative to sustained communication. Each is detailed in the following three sections. Note that we consider only fixed-order elements and assume a homogeneous processing environment in our descriptions.

Our goal is to evaluate the influence of these measures on the performance of our parallel adaptive calculations and to use them to estimate the quality of partitions generated by our current load-balancing scheme, as well as to help design and appraise new selection and load-balancing techniques.

4 METRICS

On most distributed-memory parallel computers, interprocessor communication is expensive relative to computation and on-processor data access. We estimate the cost of interprocessor communication by computing two *surface index* metrics of partitions. We use the number of boundary faces as an indication of the partition "surface area," since a face on the boundary will also have its edges and vertices on the interprocessor boundary. A face is called a *boundary face* if it is shared by two regions that are assigned to different processors. Intuitively, the surface index metrics can be thought of as surface-to-volume ratios, although the concepts of surface and volume do not fit conventional notions.

Given a set of n partitions $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, let b_i denote the number of faces on the boundary of P_i, f_i the number of faces in the partition P_i, b_t the total number of faces on all partition boundaries, and f_t, the total number of faces in all partitions, then the *maximum local ratio* r_M and the *global ratio* r_G are defined as

$$r_\mathrm{M} = \max_{i=1,\ldots,n} \left(\frac{b_i}{f_i} \right) \quad \text{and} \quad r_\mathrm{G} = \frac{b_\mathrm{t}}{f_\mathrm{t}} \ .$$

The surface index metrics are inexpensive to compute. The PMDB maintains structures representing interpartition boundaries, so computing r_M involves traversing the boundary structure on each processor in parallel, followed by a global reduction operation and a local division. The total number of faces on a boundary can be computed by first moving through the boundary structure of a processor and counting a face only if the current processor's number is less than that of the processor which shares the face. This is done in parallel on all processors, and a global summation is then carried out to produce b_t. Alternately, all of the boundary faces can be counted on each processor in parallel, and the result of a subsequent global summation is divided by two. In either instance, the computation assures that a particular face on the boundary between two partitions will be counted only once. After b_t has been determined,

$$f_\mathrm{t} = \left(\sum_{i=1}^{n} f_i \right) - b_\mathrm{t} \ .$$

An estimate of the worst case communication relative to the amount of on-processor data is provided by r_M. On some systems, the number of the processor that produced r_M may identify the processor that will be the last to complete its communication. The global metric r_G represents the total volume of communication normalized by computation and is the number of cuts [7, 8, 14] divided by the total number of faces in the mesh.

5 CONNECTIVITY OF PARTITIONS

A second quality metric involves statistics that quantify the geometric connectivity of the mesh assigned to each processor. Having multiple disjoint connected components within a processor's mesh, called *subdomain splitting* [9], is undesirable. The interprocessor boundary may be larger than necessary if a processor's mesh is not a single geometrically connected entity, and domain-decomposition methods for solving linear systems may converge slowly in this case [4]. If a relatively small disjoint part of one subdomain can be merged into a neighboring partition, the surface area on the interprocessor boundary will decrease to improve the surface index metrics.

In a three-dimensional mesh, the definition of a "connected partition" depends on the degree of the geometric entity that must be shared by two regions for them to be considered "connected." We consider *face connectivity*, *edge connectivity*, and *vertex connectivity*. As the names suggest, two regions are face-connected if they share a common face, edge-connected if they share a common edge, and vertex-connected if they share a common vertex. Face-connected partitions are the most desirable, since two regions that share a face will also share the edges of that face, and the vertices of those edges. A higher degree of face connectivity means less interprocessor communication during the computation phase, as a face on the boundary will also have its edges and vertices on the boundary as well.

The algorithm for computing connectivity starts with a region that has not been visited, and successively marks all adjacent elements as visited until all the regions that make up a connected component have been marked. This process continues until all regions in the partition have been visited. The cost of this computation increases significantly as the type of connectivity being considered moves from face to edge to vertex. There are at most four face-adjacent regions to consider at each step, but there can be many more when edge and vertex adjacency are considered.

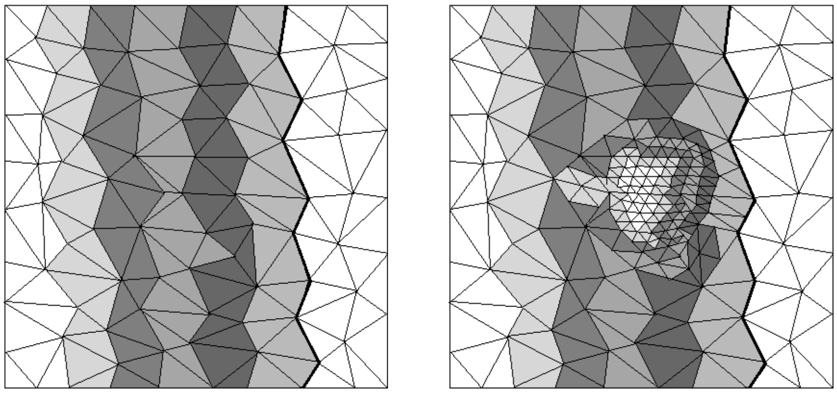

Figure 1 Slice migration of elements from right to left with all elements nearly the same size (left). Disconnected partition resulting when the same load-balancing selection is applied following refinement (right).

None of the partitioning algorithms in our system guarantee that each resulting partition consists of a single connected component. Others do ensure this [4], but they may not produce a balanced computational load. Also, a connected initial partition is no guarantee that meshes on the processors will not become disjoint after repeated applications of adaptivity and load balancing. In Figure 1, we illustrate how a connected initial partition may be split into more than one connected component in a two-dimensional mesh after refinement and load balancing are applied. Elements to the left of the heavy black line belong to Processor 0, while those to the right belong to Processor 1. The different shadings represent the "slices" that are migrated along the interprocessor boundary from Processor 0 to 1. On the left of Figure 1, we show that when the element sizes are close to uniform, this algorithm preserves the shape of the interprocessor boundary. However, on the right of Figure 1, we demonstrate that the domain becomes disconnected if refinement is applied to elements on Processor 0 that are near the boundary of Processor 1 before migration is done. The interprocessor boundary moves across areas with coarser elements more quickly than it does through the refined area. After several iterations of migration have been applied, a cluster of elements still assigned to Processor 0 has been completely surrounded by elements of Processor 1. Similar situations can occur in three dimensions.

6 INTERPROCESSOR CONNECTION DENSITY

A final measure of partition quality is the degree of interconnection between partitions. During the solution phase, a processor needs to exchange information with all of its vertex-adjacent neighbors. On a computer with a specific communication topology, this could impact performance significantly since direct neighbor communications can be significantly cheaper than more general communication. Having a large number of adjacent processors can also degrade performance on a computer like the IBM SP2 which has a high software latency for interprocessor communication compared to the speed of its processors.

We compute both the average and maximum number of processors that each processor must communicate with. These are expressed as a percentage of the largest possible number of adjacent processors (*i.e.*, $n - 1$, when using n processors).

7 EXAMPLES

The test problem that we are using involves transonic flow about an Onera M6 wing. On the left of Figure 2, we show the initial partitioning of the mesh onto 16 processors using IRB. This picture shows the mesh elements at the surface of the wing along with some of the regions that are part of the surrounding flow. The different shadings indicate elements that are assigned to different processors. This is only a small portion of the entire mesh, but it contains the interesting part at the wing's surface.

On the right of Figure 2, we show the part of the mesh that is assigned to processor number 12 of the initial partition. Notice that there is a large cluster of small elements with several larger elements outside of the main cluster. Some elements are disjoint from the main component of this partition. This looks like a poor partition, but quantitative measures are needed for an objective determination.

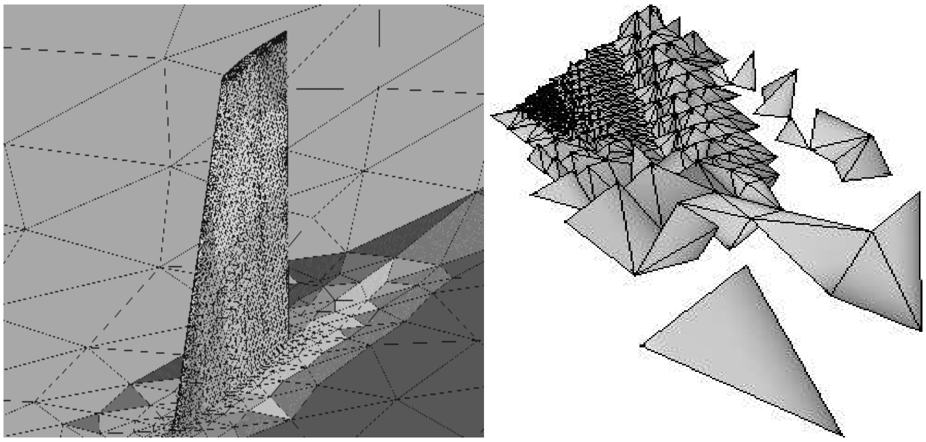

Figure 2 Initial partitioning of a portion of the mesh about an Onera M6 wing on 16 processors (left). The mesh associated with Processor 12 of this initial partition (right).

Refinements Performed on 16 Processors				
	Number of	Maximum Load		
Test Case	Tetrahedra	Unbalanced	Migration	Partitioned
1. **onera0**	85567	—	—	5348
2. **onera1**	130454	12701	8156	8154
3. **onera2**	258045	30698	16303	16128
4. **onera3**	317756	26399	19865	19860

Refinements Performed on 32 Processors				
	Number of	Maximum Load		
Test Case	Tetrahedra	Unbalanced	Migration	Partitioned
1. **onera0**	85567	—	—	2674
2. **onera1**	131006	7588	4095	4095
3. **onera2**	223501	12222	6992	6985
4. **onera3**	388837	19639	12162	12152

Table 1 Data for three h-refinements of an Onera M6 wing mesh performed on 16 (top) and 32 (bottom) processors.

Table 1 shows the mesh sizes for a series of three refinements of this initial Onera wing mesh. The initial mesh consisted of 85,567 tetrahedra which were distributed by IRB onto 16 or 32 processors. The 16- and 32-processor runs were done on IBM SP2s at Rensselaer and the NASA Ames Research Center, respectively. The other meshes were generated by performing an adaptive *h*-refinement of the prior mesh. The "Maximum Load" columns contain the maximum number of elements on any processor. The first column shows the maximum processor load before load-balancing was applied and the second shows the maximum processor load after our migration-based iterative load balancing algorithm was applied (for at most ten iterations) [2, 10]. The third column shows the maximum number of elements on any processor after a parallel global IRB was performed [6, 13].

7.1 Surface Index Statistics

In Figure 3, we show r_{G} as a function of mesh size for the four Onera meshes from 16- and 32-processor runs. The scale for mesh size is normalized by the number of elements of the initial mesh (**onera0**). The surface index metrics are expressed as percentages by multiplication by 100. In both cases, the global IRB repartitioning does better than the migration-based load balancing for all mesh sizes. With 32 processors, the percentages of faces on the interprocessor boundaries increases as expected for problems having the same mesh sizes.

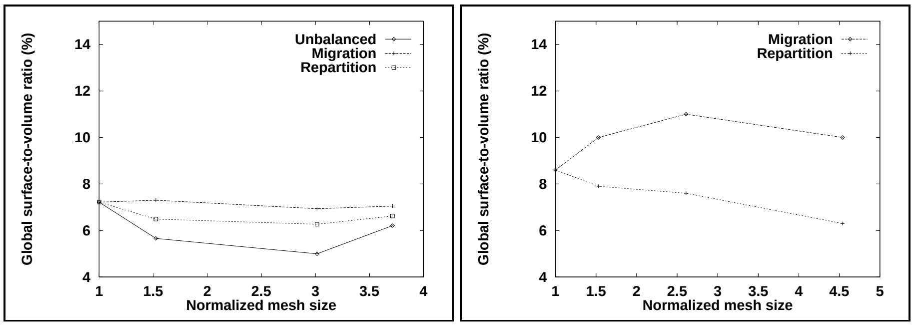

Figure 3 Global surface index r_G vs. mesh size for the meshes of Table 1 on 16 (left) and 32 (right) processors.

In Figure 4, we show data similar to Figure 3 for the maximum local ratio r_{M} on 16 and 32 processors. The vertical line in the left graph will be discussed in Section 7.4. In general, both the partitions produced by repartitioning and migration show an increase in this ratio relative to the unbalanced mesh. While the partitions generated by migration are slightly superior to those produced by repartitioning for the smaller meshes, this trend disappears as the mesh size increases.

In particular, there are two instances in which the ratio computed on the subdomains balanced by migration is significantly higher than for the repartitioned ones (*i.e.*, from **onera1** to **onera2** for 16 processors and from **onera2** to **onera3** for 32 processors). In both cases, the meshes nearly double from one refinement to the next and the resulting meshes showed a large increase in the maximum load on a processor relative to the previous mesh. Global repartitioning has a better chance of reducing the worst surface index after such a massive change in the mesh than local migration. Indeed, with 16

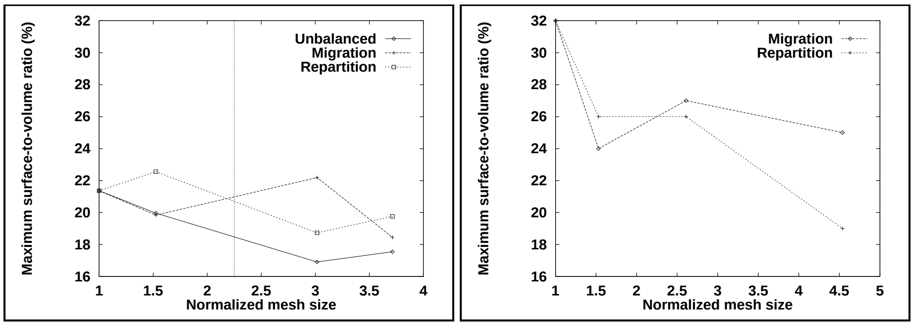

Figure 4 Maximum local surface index r_M vs. mesh size for the meshes of Table 1 on 16 (left) and 32 (right) processors.

processors, it is clear that migration would need more than the ten iterations it was allowed in order to fully balance the load (cf. Table 1). Thus, for large mesh changes, full repartitioning should be a better choice than iterative migration.

7.2 Intraprocessor Connectivity

In Figure 5, we examine the intrapartition connectivity of the meshes in Table 1 that have been balanced using iterative migration. The figure shows plots of the average and maximum number of disjoint connected components *vs.* mesh size.

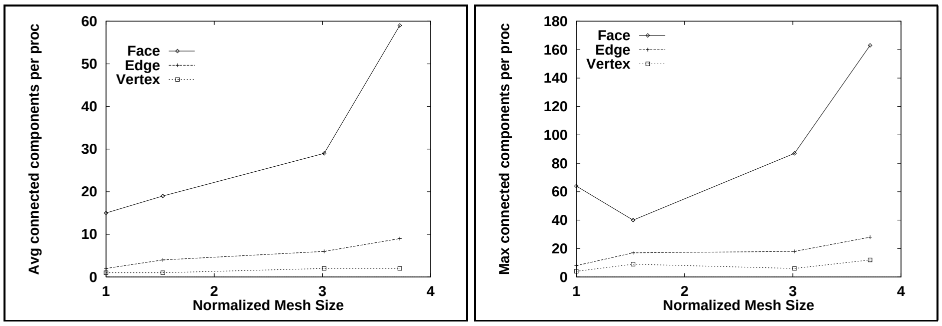

Figure 5 Average (left) and maximum (right) number of connected components per processor vs. mesh size for the meshes of Table 1 on 16 processors.

Notice that even the starting mesh (**onera0**), partitioned with IRB, has an average of 15 face-connected components (cf. Section 5) in each processor. In fact, the submesh assigned to a single processor shown on the right of Figure 2 from the IRB partitioning of the **onera0** mesh has 64 disjoint face-connected components. Further, as refinement and the migration-based load balancing are applied to obtain the later meshes, both the average and maximum number of disjoint connected components consistently increase.

This indicates that the situation illustrated in on the right of Figure 1 is occurring. Our current selection scheme for migration does not take the sizes of the elements into consideration, so the interprocessor boundaries move quickly across coarse areas as compared to highly-refined ones.

7.3 Interprocessor Adjacency

Finally, we consider the interprocessor adjacency values for the Onera meshes. In Figure 6, we show the interprocessor adjacency for the 16-processor Onera meshes. Again, the vertical line in the right plot will be discussed in Section 7.4. Maximum adjacency values are close to or at 100 percent. Even the average adjacency remains in the 75 to 85 percent range, demonstrating a very dense interconnection among the processors. A 16-processor run is too small to see any consistent difference between results for meshes after load balancing through migration and after global repartitioning.

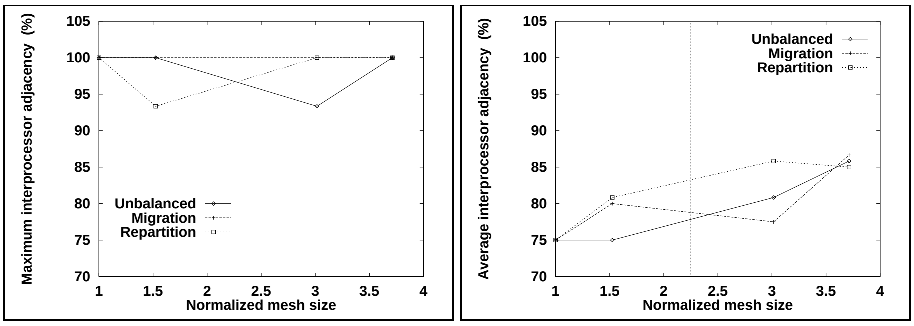

Figure 6 Maximum (left) and average (right) interprocessor adjacency vs. mesh size for the meshes of Table 1 on 16 processors.

With 32 processors, we have more meaningful results as presented in Figure 7. Both maximum and average adjacencies for partitions generated by the migratory load balancing are higher than those produced by repartitioning. This indicates that it may be worthwhile to concentrate more on avoiding increases in interprocessor connectivity when applying load balancing techniques. However, even 32 processors may still be too few to draw definite conclusions. Moving to 64 or 128 processors in future tests will help resolve this issue.

7.4 Timings

The most meaningful statistic is solution speed. Table 2 contains the average times, in seconds, to complete a single solution step of the FAST solver on 16 and 32 processors using some of the meshes we have been discussing. Only data for the last test mesh was available for the 32-processor case (and only for the load-balanced computations). The "Gain" column in the upper portion of the table indicates the percent improvement

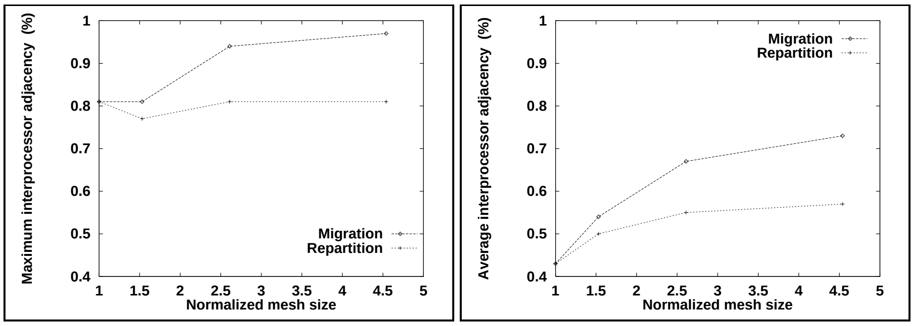

Figure 7 Maximum (left) and average (right) interprocessor adjacency vs. mesh size for the meshes of Table 1 on 32 processors.

in time for partitions produced by each algorithm relative to an unbalanced mesh. The "Difference" column for the 32-processor solution gives the percent decrease in time for the repartitioned mesh *versus* the mesh produced by migration.

Refinements Performed on 16 Processors						
Test Case	Number of Tetrahedra	Time Per Step (s)			Gain (%)	
		Unbal.	Mig.	Repart.	Mig.	Repart.
2. **onera1**	130454	434.1	229.6	218.4	47.1	49.7
3. **onera2**	258045	1821.0	692.9	722.4	61.9	60.3
4. **onera3**	317756	1356.6	919.1	1076.8	32.2	20.6

Refinements Performed on 32 Processors				
Test Case	Number of Tetrahedra	Time Per Step (s)		Difference (%)
		Mig.	Repart.	
4. **onera3**	388837	294.2	243.4	17.26

Table 2 Sizes, timings, and relative improvements in run times for example meshes from Table 1 for 16 (top) and 32 (bottom) processors.

Initial results from production runs with 32 processors of an SP2 at the NASA Ames Research Center suggested that a global repartitioning produced a significant reduction in solution time relative to the migration-based load balancing (see the 32-processor portion of Table 2). These runs indicated that interprocessor adjacency is a major factor, since repartitioning clearly beat migration in this area.

However, when similar runs were performed on 16 processors of the SP2 at Rensselaer, we found that migration produced faster solution times than the global repartitioning for the **onera3** mesh. For refinements that had similar numbers of finite elements per processor to the 32-processor **onera3** mesh, the times were comparable for both redistribution methods. Migration-generated partitions yielded slightly better results for the **onera2** mesh whereas the partitions produced by repartitioning were slightly better than those created by migration in the case of the **onera1** mesh. Recall that the plots of interprocessor adjacency for the 16-processor meshes (see Figure 6) failed to show a clear pattern and any differences were reasonably small. In addition, none of the other statistics showed any meaningful trends when compared to the timing results. There are clearly other factors influencing the 16 processor runs, which we are investigating. In particular, cache size and the amount of physical memory may have played a role in the results for the larger Onera meshes on Rensselaer's SP2 because its processors have less memory bandwidth than those of the SP2 at the NASA Ames Research Center.

The 32-processor mesh consisted of approximately 12,000 regions per processor, whereas the 16-processor meshes had about 8,000, 16,000 and 20,000 elements per processor. If we compare the values intersecting the vertical line on the left of Figure 4 to the rightmost values on the right graph of Figure 4 and the values intersecting the vertical line in the right portion of Figure 6 to the rightmost values in the right graph of Figure 7, we see that 16- and 32-processor partitions with a similar number of elements on each processor would still have different characteristics. Based on this observation, even if the refinement were adjusted to get a mesh with about 12,000 regions per processor, it is likely that there would still be a difference between the 16- and 32-processor run times on the same machine.

8 SUMMARY AND FUTURE GOALS

In summary, we have taken measures of partition quality and computed values for them on the meshes used to solve a problem involving a transonic flow about an Onera M6 wing. Our statistics have yet to establish any firm relationship to performance. Interprocessor adjacency, at least in the 32-processor case, appears to be a significant factor, but more test cases on a wider range of processor sets on a single machine are needed to quantify its importance and draw firm conclusions.

Using smaller test meshes and academic problems, we might have found some correlations between the mesh statistics and running times. However, smaller meshes, with artificially induced refinement and load imbalance, may not exhibit important characteristics, such as those which lead to the situation described in Figure 1, and therefore might not point out potential problems with a load-balancing scheme. By concentrating on meshes that arise in computational fluid dynamics problems, we can isolate the properties of a mesh partitioning strategy that lead to a performance improvement of our adaptive software in practical situations.

We would like to measure the effect that the initial partitioning has on subsequent repartitionings necessitated by h-refinement. At present, we have ORB, IRB, and Spectral Recursive Bisection (SRB) available to us. Only ORB and IRB are implemented directly as part of our system. *Chaco* [7], the SRB package made available to us by Sandia National Laboratories, runs serially and does not operate directly on meshes in our framework.

Since initial partitions are often used only briefly before adaptivity and rebalancing will be necessary, a good starting partitioning is not as important for parallel adaptive techniques as for other methods. This is especially true when using a repartitioning algorithm because it generates an entirely new partitioning. However, iterative migratory load balancing simply modifies existing partitions to rebalance load, and so undesirable features of a poor initial partitioning may propagate to subsequent partitionings. Thus, it might be worthwhile to investigate partitioning methods that guarantee geometrically connected initial partitions [4], as well as developing others that focus on minimizing the number of interprocessor connections, though the importance of these characteristics is still under study.

A major area of current and future work is to use these measures of partition quality to develop, implement, and analyze the performance of alternate schemes to choose elements for migration during load balancing. We are now experimenting with geometric methods which take advantage of the geometric location of elements that are candidates for migration and their positions in space relative to the centroids of the processors which are sending and receiving load. Vidwans *et al.* [15] have presented divide-and-conquer load balancing methods that take advantage of the geometric information in a similar framework. We have yet to work with inertial heuristics, but such algorithms should give results similar to geometric methods while being potentially less expensive. Methods that select elements to maintain "compactness" of partitions, and those that move elements to improve interprocessor adjacency are also being considered. However, when any type of information is used to improve the partitions, we must also be sure that the cost of doing so does not exceed the savings we wish to realize during the computation phase.

Acknowledgements

We would like to thank Mark Beall for access to the SCOREC Mesh Database, Hugues de Cougny for the use of the parallel mesh refinement and parallel inertial repartitioning packages that he developed, and the NASA Ames Research Center for access to their SP2. The first and fourth authors were supported by ARO grant DAAH04-93-G0003, and the second, fourth, and sixth authors were funded under ARO grant DAAH04-95-1-0091. Support was provided to the second, third, fourth, fifth, and seventh authors via NSF Grant ASC-93-18184. In addition, the second, fifth, and seventh authors were supported under NSF Grant CCR-92-16053, and the fifth and seventh authors were funded by ONR Grant N00014-93-1-0076.

REFERENCES

[1] M. W. Beall. SCOREC mesh database version 2.3 user's guide. Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, 1994.

[2] 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 *Proceedings of the Twelveth AIAA Applied Aerodynamics Conference*, pp. 1–11, Colorado Springs, 1994.

[3] K. Clark, J. E. Flaherty, and M. S. Shephard, Eds. *Applied Numerical Mathematics*, 14, special ed. on Adaptive Methods for Partial Differential Equations, 1994.

[4] L. Dagum. Automatic partitioning of unstructured grids into connected components. In *Proceedings of the Supercomputing Conference 1993*, pp. 94–101, Los Alamitos, 1993.

[5] H. L. de Cougny, K. D. Devine, J. E. Flaherty, R. M. Loy, C. Özturan, and M. S. Shephard. Load balancing for the parallel adaptive solution of partial differential equations. *Applied Numerical Mathematics*, 16:157–182, 1994.

[6] H. L. de Cougny, M. S. Shephard, and C. Özturan. Parallel three-dimensional mesh generation on distributed memory MIMD computers. Submitted to *Engineering with Computers*.

[7] B. Hendrickson and R. Leland. The Chaco user's guide, version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque, 1993.

[8] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. Technical Report SAND93-0074, Sandia National Laboratories, Albuquerque, 1993.

[9] S.-H. Hsieh, G. H. Paulino, and J. F. Abel. Evaluation of automatic domain partitioning algorithms for parallel finite element analysis. Structural Engineering Report 94-2, School of Civil and Environmental Engineering, Cornell University, Ithaca, 1994.

[10] C. Özturan. Distributed environment and load balancing for adaptive unstructured meshes. PhD dissertation, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, 1995.

[11] Y. Saad and M. Schultz. A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[12] F. Shakib, T. J. R. Hugues, and Z. Johan. New finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 89:141-219, 1991.

[13] M. S. Shephard, J. E. Flaherty, H. L. de Cougny, C. Özturan, C. L. Bottasso, and M. W. Beall. Parallel automated adaptive procedures for unstructured meshes. To appear in *Parallel Computing in CFD*, AGARD, Neuilly-Sur-Seine, 1995.

[14] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computer Systems in Engineering*, 2:135–148, 1991.

[15] V. Vidwans, Y. Kallinderis, and V. Venkatakrishnan. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA Journal*, 32:497–505, 1994.