A Comparison of Zoltan Dynamic Load Balancers for Adaptive Computation*

J. D. Teresco† and L. P. Ungar
Department of Computer Science
Williams College
Williamstown, MA 01267

January 28, 2003

Abstract

Dynamic load balancing is an essential tool for parallel adaptive computation. It has been an important research topic for more than a decade. The *Zoltan* library provides applications with a reusable, object-oriented interface to several load balancing algorithms, including coordinate bisection, octree/space filling curve methods, and multilevel graph partitioners. It allows run-time selection among balancing techniques, facilitating comparisons. It also provides a framework in which to modify existing algorithms or to implement new algorithms. We compare the performance of several available methods as mesh partitioners and as dynamic load balancers for an adaptive computation. Partition quality metrics and running times of the balancers and the solution process are measured for several methods. We also examine the effect of octant granularity for the Zoltan octree partitioning algorithm, and describe a modification of the recursive coordinate bisection algorithm that bisects along only one coordinate axis, to achieve a "slice" partitioning, designed to minimize interprocessor adjacency.

1 Introduction

Adaptive computational techniques provide a reliable, robust, and efficient means of solving problems involving partial differential equations (PDEs) by finite difference, finite volume, or finite element technologies [11]. With adaptivity, the initial discretization of the computational domain and the numerical method are enhanced during the solution process to concentrate computational effort in areas of interest or activity. The enhancement may involve h-refinement [38], where a mesh is refined or coarsened, respectively, in regions of low or high accuracy; r-refinement [1, 2], where a mesh of a fixed topology is moved to follow evolving dynamic phenomena; and p-refinement [3, 36], where the method order is increased or decreased, respectively, in regions of low or high accuracy.

*Williams College Department of Computer Science Technical Report CS-03-02

†Corresponding author. E-mail: terescoj@cs.williams.edu, Tel: (413)597-4251, Fax: (413)597-4250

Parallelism greatly complicates an adaptive computation. Domain decomposition, data management, and interprocessor communication must be dynamic since adaptive h- and p-refinement introduce computational imbalance. The dynamic data structures used with adaptive software severely limit automatic parallel optimization. A dynamic load balancing procedure is needed to compute a new decomposition following an adaptive step. The quality of the decomposition affects the efficiency of the solution, and different algorithms produce decompositions with different properties and costs [9].

A large number of dynamic load-balancing algorithms have been developed. The *Zoltan* library [14, 16, 17] provides implementations of or interfaces to several state-of-the-art load balancers, including coordinate bisection methods [6, 19, 38], octree/space filling curve methods [4, 15, 23, 24, 29, 30, 32, 35], refinement tree methods [31], and multilevel graph partitioning [26, 27, 39, 43]. Zoltan allows an application to choose from among these algorithms through a consistent interface and the specification of a number of callback functions that provide necessary application-specific information to Zoltan. We take advantage of this flexibility to compare balancing algorithms.

We begin by describing the software, load balancing algorithms, meshes, partition quality metrics, and the computer system used in our studies (§2). We next present a comparison of the Zoltan load balancers, used both as initial partitioning procedures (§3) and as dynamic rebalancers for an adaptive computation (§4). We consider partition quality and running times for the partitioning examples. For the adaptive computations, total solution times are also compared. We continue with a study of octant granularity, the maximum number of objects (*e.g.* mesh elements) that are allowed in a terminal octant, in the octree partitioning algorithm (§5). Our final study describes a modification to the recursive coordinate bisection (RCB) algorithm that fixes all cuts along one coordinate axis (§6). This leads to a "slice" partitioning that may be useful to minimize the interprocess adjacency, the number of processes with which each process must communicate during a solution phase. Finally, we discuss the results and future research directions (§7).

2 Background

This section describes the software tools, meshes, partition quality metrics, and parallel computer system used in the studies that follow.

2.1 Parallel Adaptive Software

We apply the Zoltan load balancers to several software packages developed at Rensselaer Polytechnic Institute. The *Rensselaer Partition Model* (RPM) [41, 42] provides distributed mesh data structures and information about the parallel computational environment in which a program is executing. The basic mesh data structures in RPM are provided by the *SCOREC Mesh Database* (MDB) [5]; 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 [37] and facilitates p-refinement [36] by allowing attachment of degrees of freedom to the mesh entities and by providing necessary geometric information.

Partitioning studies are done using PMDBtool [41], a program that reads a script of commands, each of which results in a call to an appropriate library function. The input file is processed using the standard Unix tools for compiler and interpreter implementation, *lex* and *yacc* [28]. Lex converts the input script into tokens, which are parsed by yacc. Yacc calls PMDBtool C functions with appropriate arguments. Each PMDBtool function, in turn, calls the necessary functions in RPM, Zoltan, or other libraries.

Our adaptive computation studies use *LOCO* [24, 29], which implements a parallel adaptive discontinuous Galerkin [8, 12, 13] solution of the compressible Euler equations. We consider the "perforated shock tube" problem, which models the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent [22]. This problem was motivated by flow studies in perforated muzzle brakes for large calibre guns [18]. Our focus is on the quasisteady flow that exists behind the contact surface for a short time. Using symmetry, the flow may be solved in one half of the domain bounded by a plane through the vent. The initial mesh contains 69,572 tetrahedral elements, and has been prerefined near the interface between the shock tube and vent. The larger cylinder (the shock tube) initially contains air moving at Mach 1.23 while the smaller cylinder (the vent) is quiet. A Mach 1.23 flow is prescribed at the tube's inlet and outlet. The walls of the cylinders are given reflected boundary conditions, and a far field condition is applied at the vent exit [23]. Flow begins as if a diaphragm between the two cylinders were ruptured. After 19 adaptive steps, the mesh contains 451,692 elements.

2.2 Meshes

The following tetrahedral meshes were selected for our studies to represent a variety of mesh sizes and geometric complexity.

- **Cone**: This is a 42,786-element mesh, used to model a shock wave in a compressible flow impacting the side of the cone. Only half of the cone is modeled [21].
- **Muzzle**: This is a 69,572-element mesh used in the perforated shock tube simulation described above. This mesh is a quarter cylinder with half of a cylindrical venting hole [22].
- **Two Pipes**: This is a 3240-element mesh that represents the junction of two pipes.
- **Quartercone**: This is a 32,866-element mesh, used to model the steady flow at Mach 5 past a cone having a half-angle of 10 degrees [20].

2.3 Load Balancing Procedures

Our studies consider four geometric algorithms, where the partitioning is determined only by object coordinates. *Recursive Coordinate Bisection* (RCB) [6] proceeds by bisecting the

domain along the Cartesian coordinate of its longest dimension. Elements are sorted according to the bisecting coordinate, and half of the elements are assigned to each subdomain. *Recursive Inertial Bisection* (RIB) [19, 38, 40] proceeds similarly, but in a direction orthogonal to the subdomain's principal axis of inertia at each step rather than along the coordinate axes. *Octree Partitioning* (OCTPART) [10, 15, 23, 30, 32] uses an octree decomposition of the computational domain in which the root of the tree, representing the entire domain, is recursively divided into eight "child" octants until each subregion (leaf octant) holds at most an application-specified number of objects. To achieve a partitioning, a Hilbert space-filling curve [7, 33, 34] tree traversal is used to define a global ordering on leaf octants. Each partition is assigned a contiguous segment of this global ordering. *Hilbert Space-Filling Curve* (HSFC) partitioning [16] proceeds in a manner similar to OCTPART, except that the SFC is constructed directly on the objects to be partitioned rather than on a tree structure.

We also consider two graph partitioning algorithms, where the graph vertices are the objects to be partitioned and the graph edges are the connections between them (*e.g.* a face shared by two mesh regions). *Jostle* [39, 43] and *Parmetis* [26, 27] each utilize a multilevel approach. These are libraries separate from Zoltan. Zoltan provides an interface that allows Zoltan load balancing calls and callbacks to be used to construct the input arrays required by these packages.

2.4 Partition Quality Metrics

The quality of the partitions used to distribute a computation has a significant effect on solution time for many applications [9]. While many factors may be important when considering the "quality" of a partitioning [25], the most common are computational balance and partition boundary size. We will quantify partition quality using two metrics: (*i*) surface index, and (*ii*) interprocess adjacency. Our examples assign exactly one partition per process, so the terms "process" and "partition" are used interchangeably.

2.4.1 Surface Indices

Surface indices measure interprocess communication volume. In our examples, they are analogous to surface to volume ratios, with the number of element faces on partition boundaries being viewed as "surface area" relative to the "volume" of element faces in a partition. A large ratio of faces on partition boundaries to total faces indicates a large inter-process communication volume.

Thus, for NP partitions $P = P_1, P_2, ..., P_{NP}$, let b_i denote the number of partition-boundary faces and f_i denote the total number of faces of P_i. The *maximum local surface index*,

$$r_M = \max_{i=1,...,NP} \frac{b_i}{f_i}, \tag{1}$$

measures the maximum communication volume on any process. Let b_t denote the total number of boundary faces in all partitions and f_t the total number of faces in all partitions. The *global surface index*,

$$r_G = \frac{b_t}{f_t}, \tag{2}$$

measures the total communication volume. This can be the most important measure when the communication network is shared among all compute nodes.

2.4.2 Interprocess Adjacency

Interprocess adjacency is the percentage of other processes with which each process must communicate. This correlates to the number of message startups needed during a solution procedure. Thus, for NP processes $P = P_1, P_2, ..., P_{NP}$, let c_i denote the number of processes with which process i must communicate. The *maximum interprocess adjacency* is

$$adj_M = \max_{i=1,...,NP} \frac{c_i}{NP-1}, \tag{3}$$

and the *average interprocess adjacency* is

$$adj_A = \frac{1}{NP} \sum_{i=1}^{NP} \frac{c_i}{NP-1}. \tag{4}$$

These measures are especially important for an interconnection network with high message latency. Exchanging information with a number of neighboring processes often requires a serialization of message setup, making adj_M an important measure of scalability. These measures can also be significant when network topology allows nearest-neighbor communication to be significantly faster than more general communication.

2.5 Computing Environment

Our studies were performed using a cluster of Sun Microsystems servers at Williams College. The compute nodes consist of two Enterprise 420R servers, each with four 450MHz Sparc UltraII processors and 4 GB memory, and six Enterprise 220R servers, each with two 450MHz Sparc UltraII processors, and 512 MB or 1 GB memory. All nodes are connected by a fast (100 Mbit) Ethernet. Four of the nodes have a gigabyte interconnect from Dolphin, Inc., but only the fast Ethernet is used for our studies.

Most studies were run for even numbers of processes ranging from two to twenty. Multiple combinations were used for the two to eight processor runs. For two processes, two nodes with one processor per node (2,1) and one node with two processors per node (1,2) were used. For four processes, one node with four processors per node (1,4), two nodes with two processors per node (2,2), and four nodes with one processor per node (4,1) were used. Similar combinations were used for six ((3,2) and (6,1)) and eight ((2,4), (4,2), (8,1)) processes. This results in sixteen combinations.

3 Initial Partitioning

We first examine the effectiveness of the Zoltan load balancers as initial partitioners for the meshes described in §2.2. Default Zoltan parameter values were used.

For each run, the total run time, surface index, and interprocessor adjacency statistics were gathered. Run time was not included for the HSFC partitioning, as an optimized version was unavailable, making any comparison unfair.

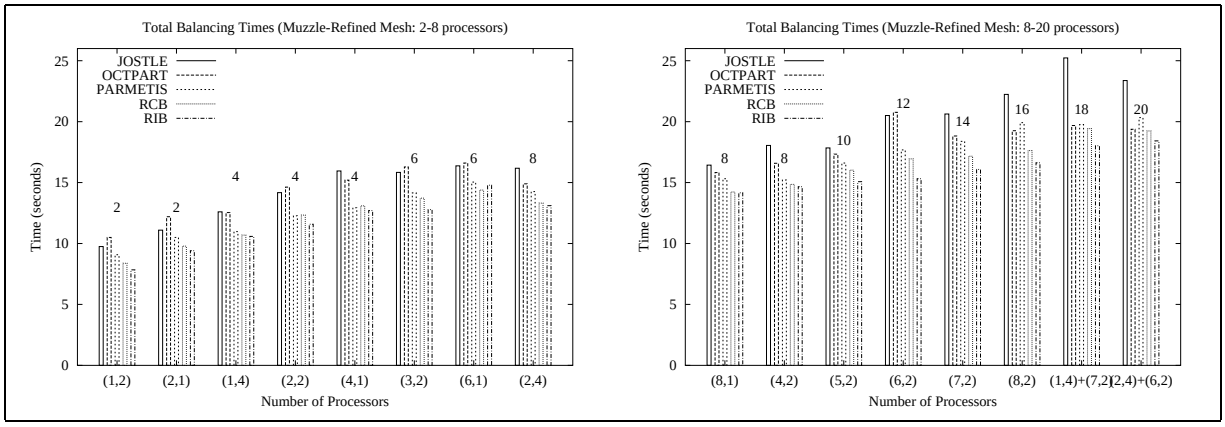

Figure 1: Total initial partitioning times for the Muzzle mesh. Jostle and Parmetis are often the slowest and the recursive bisection procedures are usually the fastest. The notation for the number of processors along the x-axis is (# nodes, # processes per node).

Figure 1 shows total balancing times for the initial partitioning of the Muzzle mesh. This includes setup time, the time for Zoltan to compute the decomposition, and the time to migrate the mesh to achieve the decomposition. In most cases, the methods were ranked RIB, RCB, Parmetis, OCTPART, and Jostle, fastest to slowest, though RCB was sometimes faster than RIB and Jostle was sometimes faster than OCTPART for small numbers of processes. Timings for the other meshes are not shown, but show the same trends. Partitioning time increased with the number of processes, since the timings include migration time, and more partitions require a larger percentage of the mesh to be migrated. Runs with the same number of processes but with intranode communication ran faster than those that required Ethernet communication.

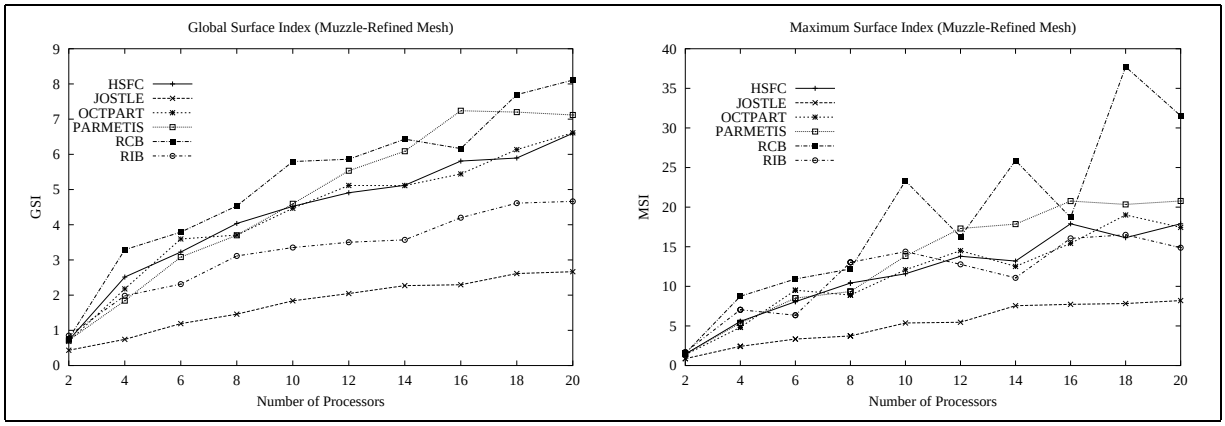

Figure 2: r_G (left) and r_M (right) for the Muzzle mesh. Parmetis consistently achieved the lowest surface indices.

Figure 2 compares surface indices for the Muzzle mesh. Jostle produced the lowest surface indices, RCB the highest, with the others clustered in the middle. Statistics for the other meshes are not shown, but Jostle consistently achieved the best surface indices. Partitioning the same size mesh into more parts will almost always increase surface indices. This is the

case in our studies.

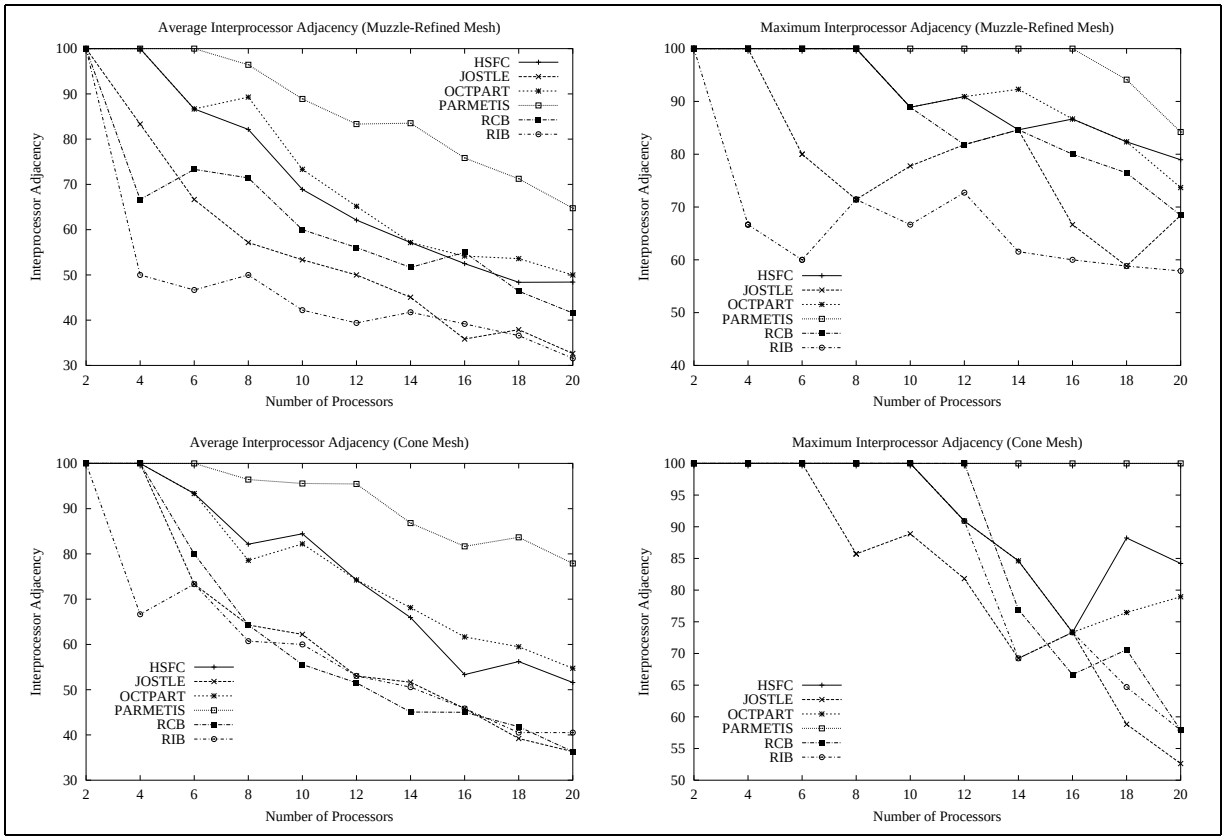

Figure 3: adj_A (left) and adj_M (right) for the Muzzle mesh (top) and Cone mesh (bottom). RIB and Jostle produced the lowest adjacencies in most cases for the Muzzle mesh, while RCB also performed well for the Cone mesh.

Figures 3 and 4 show interprocess adjacency values for the four meshes. Adjacency values generally decrease and become more meaningful as the number of processes increases. For the Muzzle mesh, RIB usually had the lowest adj_A, followed by Jostle and RCB, while Parmetis was consistently the highest for both average and maximum adjacency. Jostle, RCB, and RIB achieved the best adjacency values for the Cone mesh and Parmetis remained the worst. Jostle had the lowest values for the Two Pipes mesh and Jostle and RCB were the best for the Quartercone mesh. As these studies show, interprocess adjacency statistics often depend heavily on mesh structure. Consider RIB, which performed well in this measure for the Muzzle and Cone meshes, but not as well for Quartercone and Two Pipes.

4 Dynamic Load Balancing for an Adaptive Analysis

To examine dynamic load balancing performance, we solve the "perforated shock tube" problem described in §2.1. We compare Jostle, Parmetis, RCB, and RIB, with the default parameter values for each algorithm. Load balancing uses element weights specified as the inverse of the radius of a sphere inscribed in each element to account for the computational

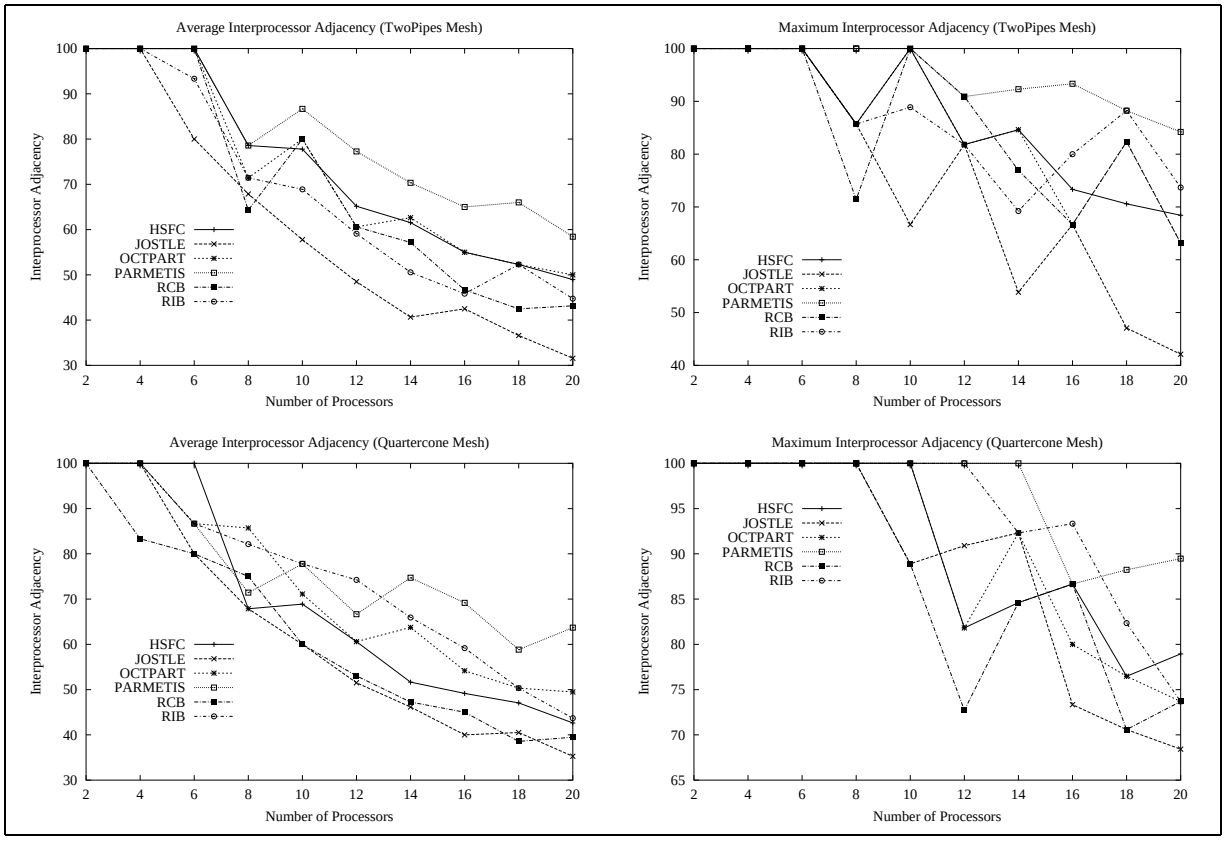

Figure 4: adj_A (left) and adj_M (right) for the Two Pipes mesh (top) and Quartercone mesh (bottom). Jostle produced the lowest adjacencies in most cases for the Two Pipes mesh, while RCB also performed well for the Quartercone mesh.

imbalance introduced by the Local Refinement Method [23]. Timings are averaged over two runs. Times are broken down as rebalance times, refinement times, solution times, and total times (the sum of the other three). Partition quality and timing statistics are presented at each refinement step and, and times as cumulative totals. To keep the number of graphs manageable, statistics broken down by refinement step are shown only for 8 and 20 processes.

We first examine partition quality metrics for the meshes at each adaptive refinement step. Figure 5 presents surface indices and Figure 6 presents interprocess adjacencies. Surface index rankings were consistent, with Jostle the best, followed by Parmetis, RIB, and RCB. Jostle also had the lowest average adjacency, but Parmetis became much worse. For the 20 process adj_M, RCB was actually the best and Jostle the worst.

Figure 7 shows the times for each adaptive step for the perforated shock tube on 8 and 20 processes, broken down into rebalancing times, refinement times, and solution times. RCB usually had the fastest rebalancing time, and Jostle was consistently the slowest. In many cases, Jostle took several times longer than the other methods. For refinement time, the costs for Jostle, Parmetis and RIB were similar, but RCB was significantly slower in most 8-process cases and in some 20-process cases. The most significant and most consistent differences were in the time spent in the solution process. This time includes the actual computation and the communication to exchange solution information across partition boundaries. Differences in

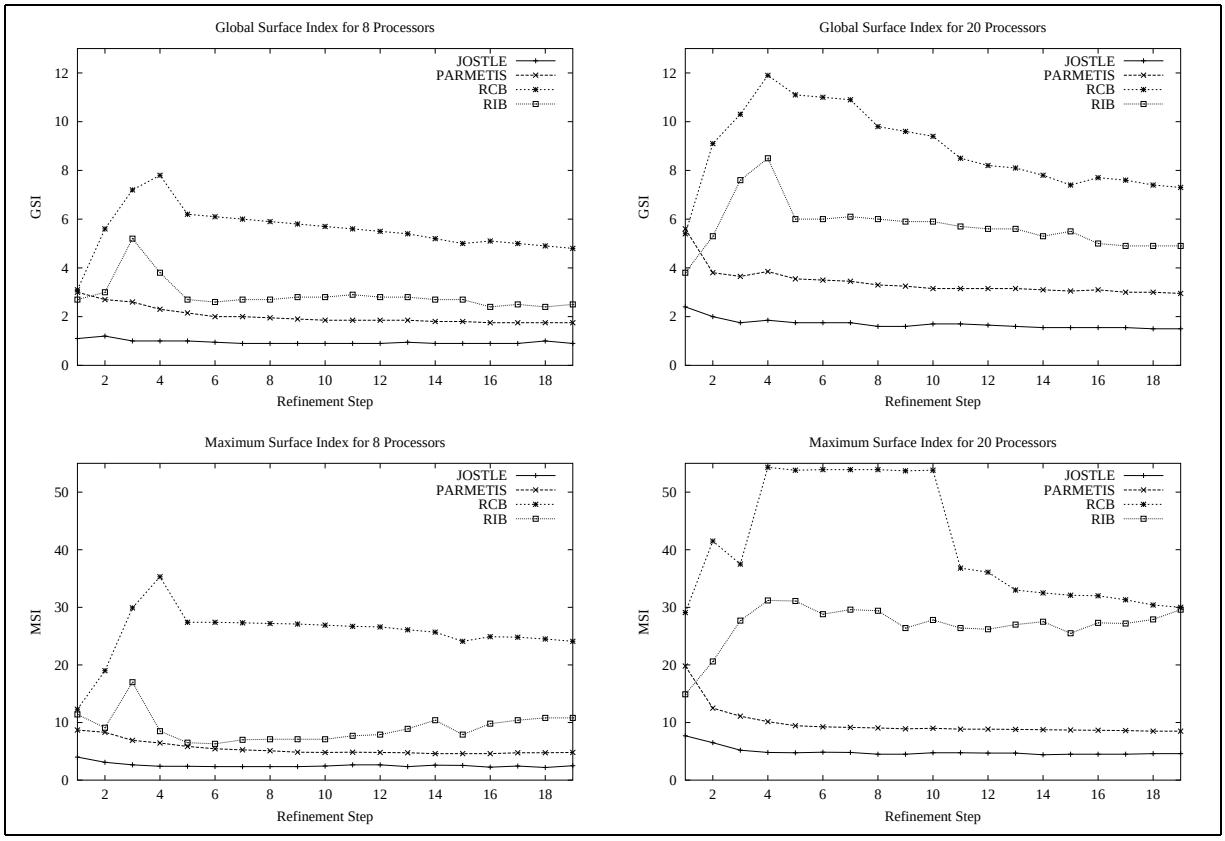

Figure 5: r_G (top) and r_M (bottom) at each adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. Jostle achieved the best surface indices in all cases.

this stage may be caused by load imbalance or by interprocess communication delays. Here, the Jostle runs were the fastest, followed by Parmetis, RIB, and RCB.

Figure 8 shows the total time to solution, which sums the corresponding values from the graphs in Figure 7. As solution time is the largest portion of the overall time, Jostle's slower partitioning time was dominated by its faster solution time. Jostle consistently achieved the fastest time to solution, followed by Parmetis and RCB in both the 8 and 20 process cases.

Figure 9 shows the cumulative times across all adaptive refinement steps for the perforated shock tube as the number of processes is increased from 4 up to 20. Times are shown for the substages (rebalancing, refinement, solution) and the total time. Across the board, Jostle and Parmetis achieved the best overall times. Overall times with RCB and RIB were several times slower than the graph partitioners. The longer rebalancing times for Parmetis and especially Jostle were not a significant factor when compared to the total solution times.

5 OCTPART Leaf Octant Granularity

The OCTPART algorithm constructs an octree decomposition of the computational domain, and uses a traversal of that tree to assign octants to partitions. The depth of the tree is determined by placing a threshold on the number of objects can exist in a leaf octant. If

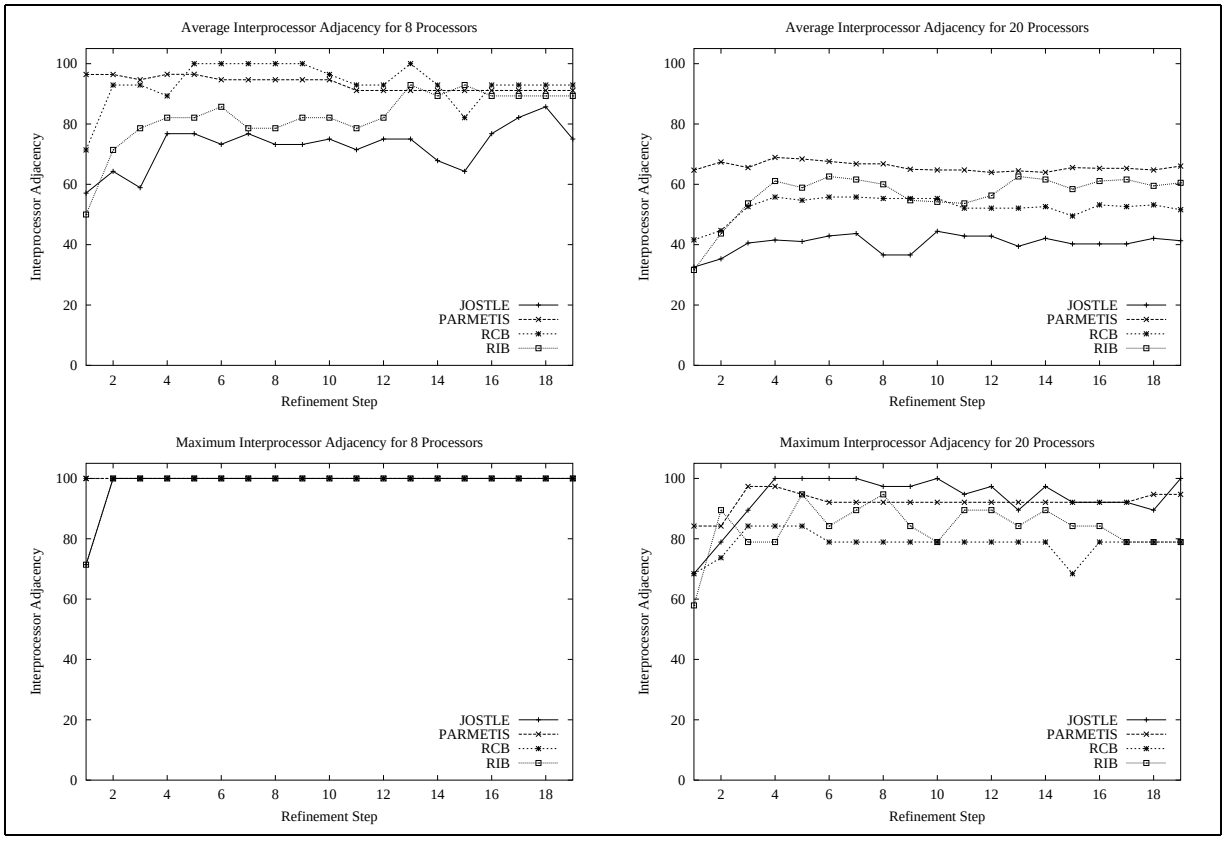

Figure 6: adj_A (top) and adj_M (right) at each adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. Jostle achieved the best adj_A in both cases. All methods had $adj_M = 100\%$ for 8 processes. RCB achieved the lowest adj_M for 20 processes.

a leaf octant contains too many objects, it is refined into eight new leaf octants, and the objects contained in the original are dispersed among the children. Setting this threshold low results in a larger tree structure, but allows for a more exact balance. Larger thresholds allow a smaller tree structure at the expense of a coarser partitioning, which may introduce small imbalances.

To determine the effect of leaf octant granularity on the octree, we vary the refinement threshold setting from 1 to 200. We increase by 5 up to 50, then 10 up to 100, and by 20 up to 200. We use the same combinations of processes as in previous studies.

Figure 10 shows that r_G and adj_A were nearly identical for all octant granularities. However, there were some differences introduced in the strictness of the load balance. With an octant granularity of 1, the partitions had an imbalance of at most 1 object. With larger octant granularity, imbalances as large as the granularity may result, and we observed imbalances up to 155 objects when a granularity of 200 was used.

Figure 11 shows partitioning times as the octant granularity is varied. These times include the partitioning, but not the mesh migration to achieve the partitioning. Since approximately the same percentage of the mesh is migrated regardless of octant granularity, migration time is nearly constant. Partitioning times dropped as the granularity was increased from 1 up

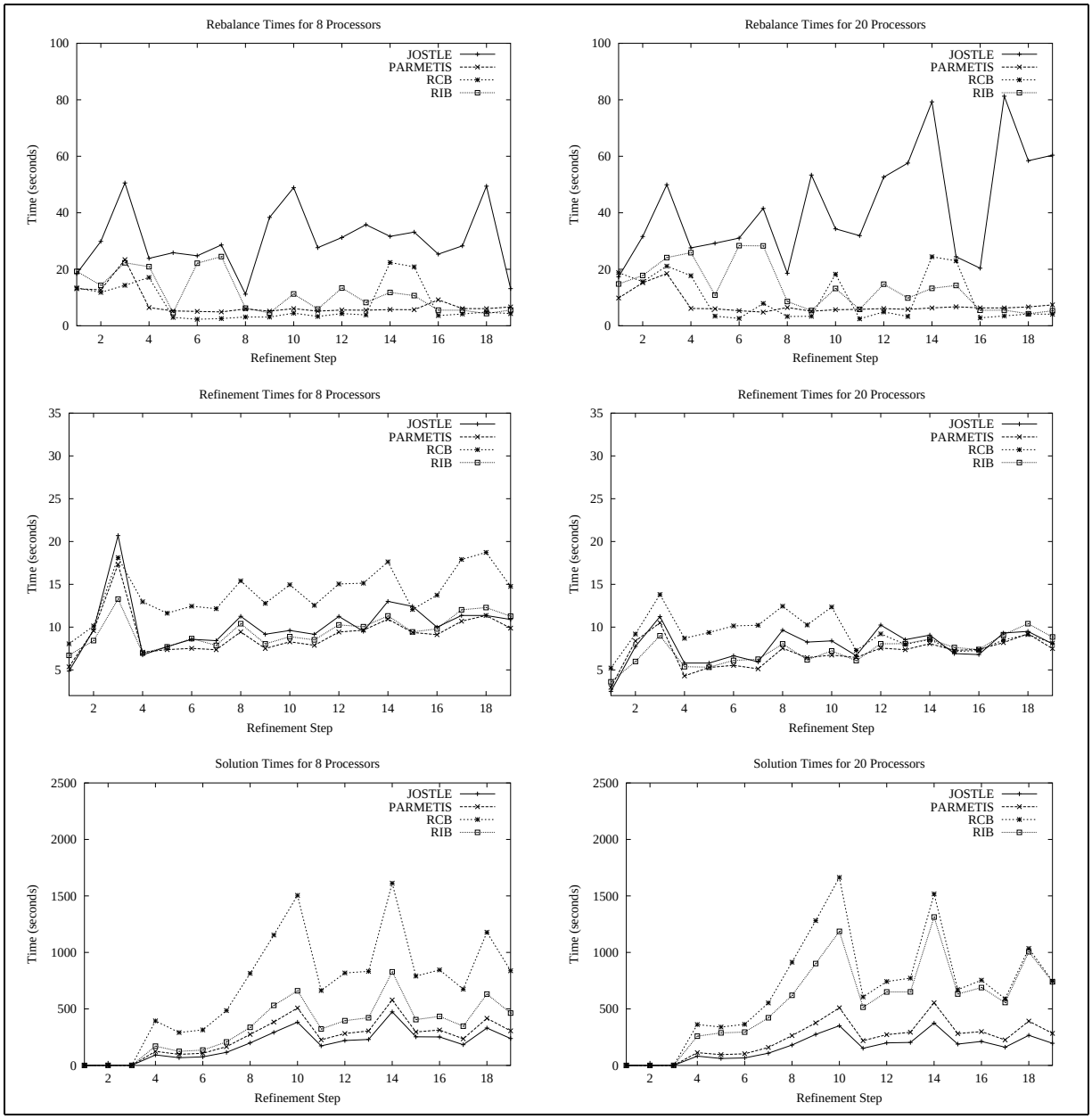

Figure 7: Solution times for each adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. The top set of graphs are the rebalancing time at each step, which includes time for Zoltan to compute the new decomposition and the time to migrate the mesh and solution data to achieve it. The middle graphs are mesh refinement times, and the bottom are the times spent computing the solution after each refinement step. A graph with the sum of these values is in Figure 8.

to around 20. As the granularity increased beyond 40, partitioning times rose steadily.

For the Muzzle mesh on this parallel computer, a granularity of 20 to 40 objects per octant was the most effective. In this range, the partitioning time was minimized, and only a small imbalance was incurred.

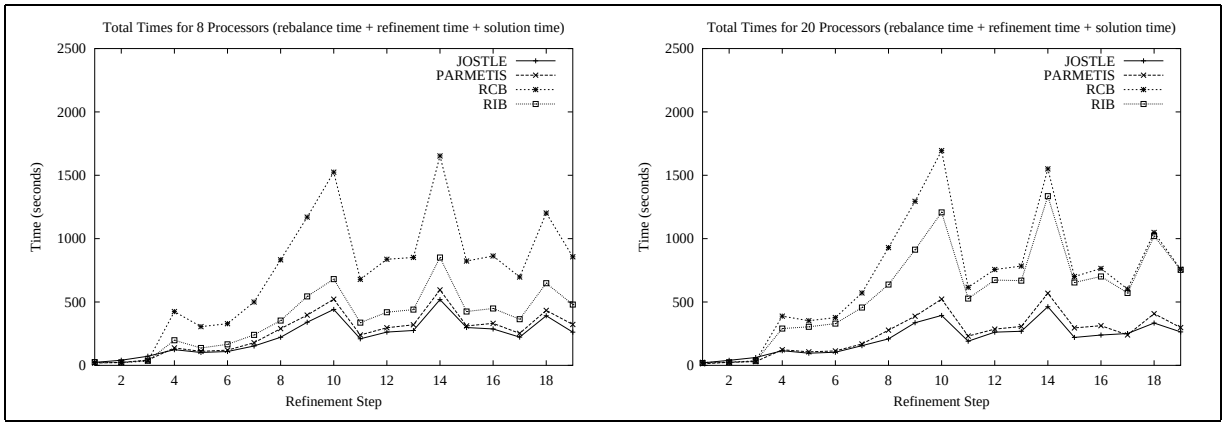

Figure 8: Total analysis times (time to solution) by adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. These graphs sum the times for rebalancing, refinement, and solution that are shown in Figure 7.

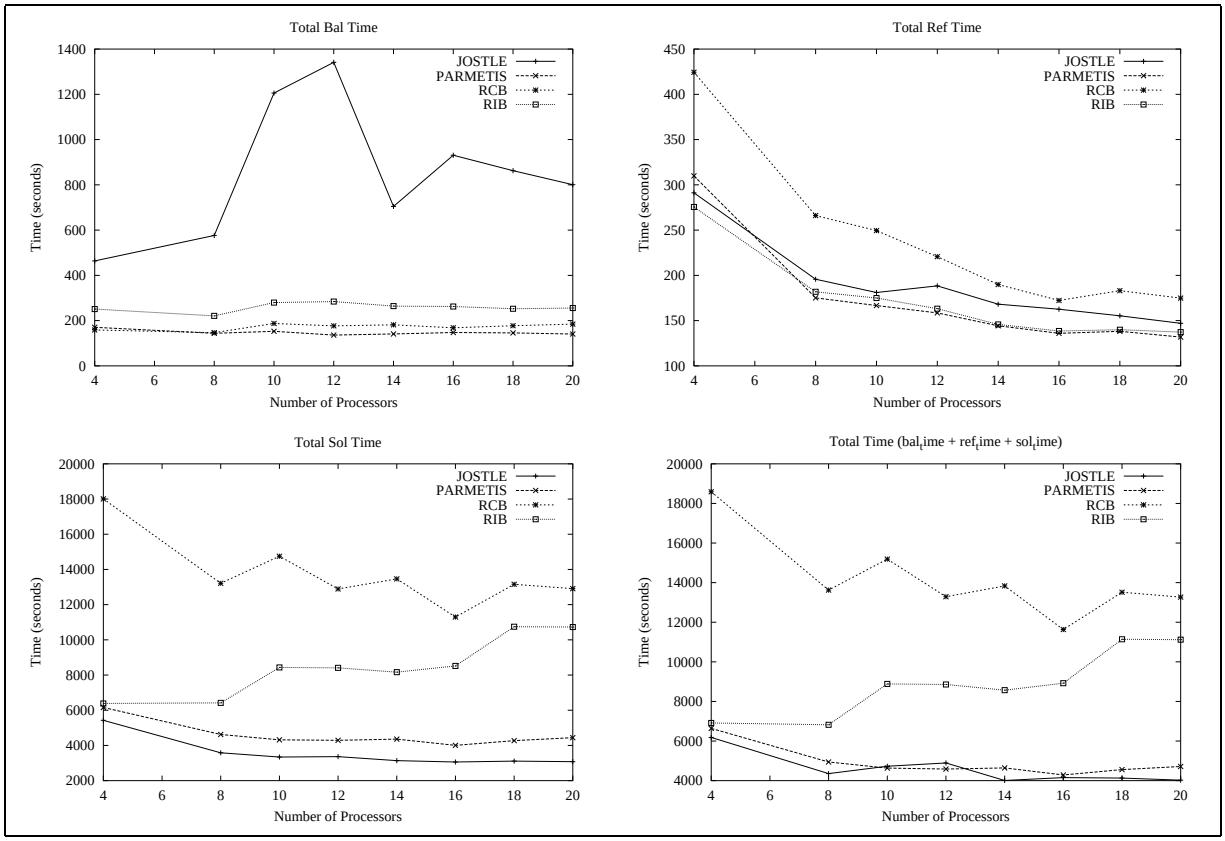

Figure 9: Times for the perforated shock tube as the number of processes ranges from 4 to 20. The graphs show cumulative rebalancing times (top left), cumulative mesh refinement times (top right), cumulative solution times (bottom left), and total analysis times (bottom right) for the 19 adaptive steps. The total analysis times are the sums of the other three.

6 Slice Partitioning

Previous studies have shown interprocess adjacency, the number of neighbors with which each processor must communicate during the solution phase, to be an important efficiency

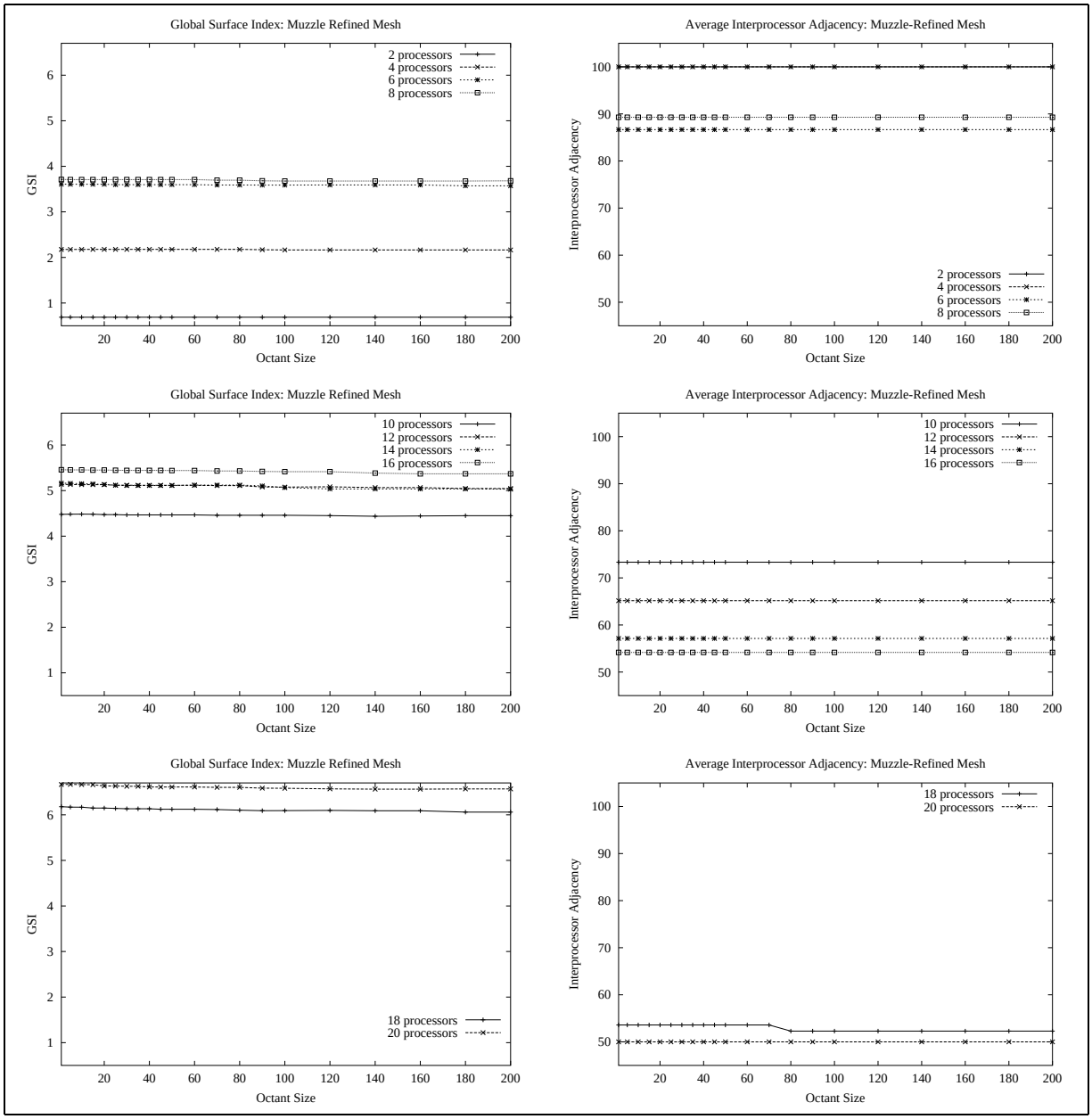

Figure 10: r_G (left) and adj_A (right) for partitions of the Muzzle mesh on from 2 (top) to 20 (bottom) processes as the octant granularity is varied. Granularity had little or no effect on these statistics.

factor [10]. To minimize adjacency, we modified the RCB algorithm to produce "slice" partitions, where each cut is along the same axis.

We compare the original RCB algorithm with the slice partitioner used to cut along each coordinate axis. We again use the perforated shock tube problem described in §2.1.

Figure 12 shows surface indices at each adaptive refinement step of the perforated shock tube. RCB x-cut achieved the best r_G and r_M values in all cases. RCB y-cut was also better than the original RCB in many cases, but RCB z-cut was much worse.

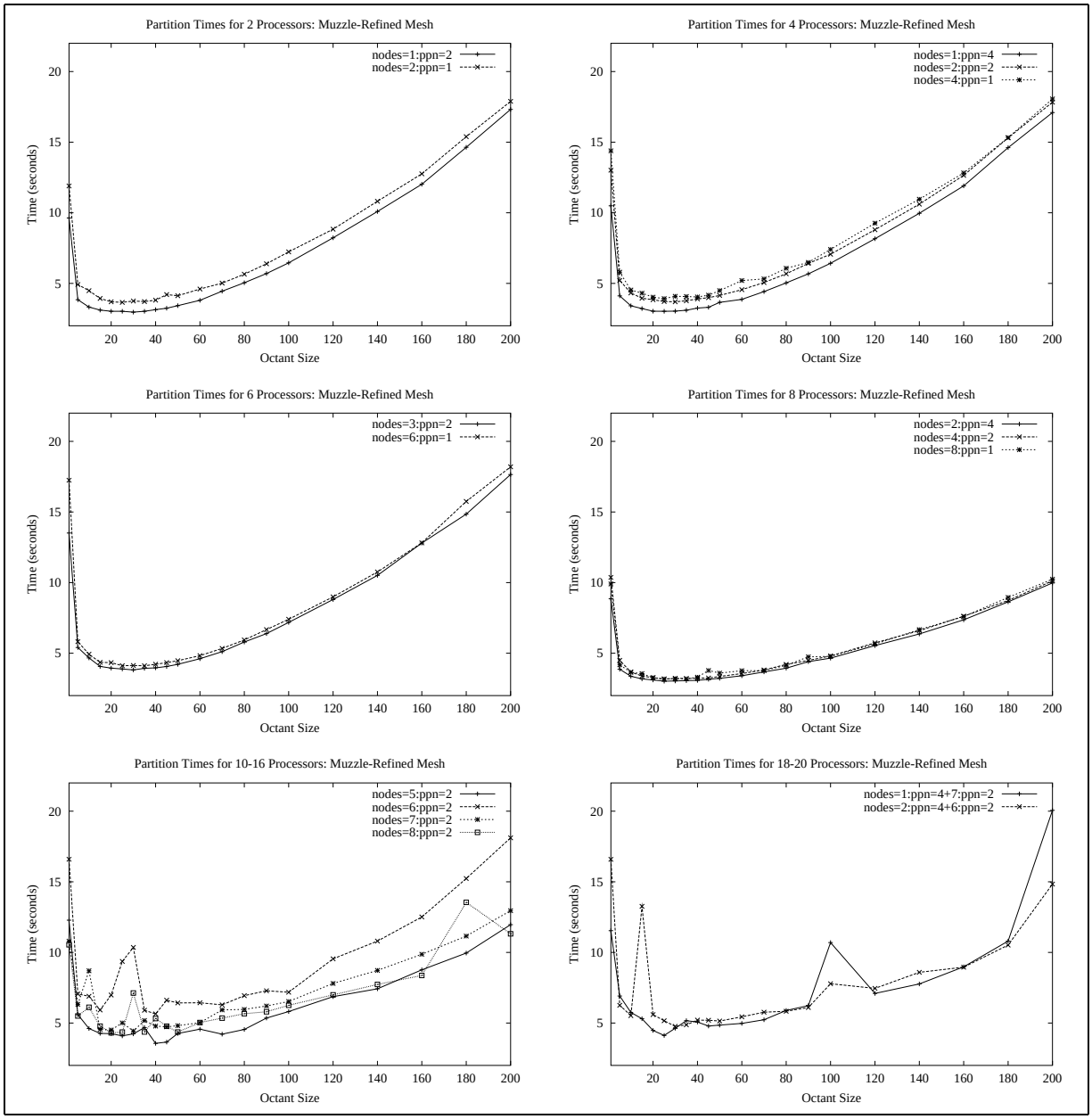

Figure 11: OCTPART partitioning time for the Muzzle mesh as the octant granularity is varied from 1 to 200 objects on from 2 to 20 processes. Different numbers of processes are placed on different graphs for readability.

These results are highly dependent upon mesh structure. For the perforated shock tube, the main cluster of refinement is along the interface between the main barrel and the vent hole, so any cut plane that passes through that region will incur poor surface index values. Since this interface is in the xy-plane, the z-cut RCB is most likely to make an unfortunate cut through this region.

Figure 13 shows interprocess adjacencies at each step of the perforated shock tube. adj_A values were lowest for the z-cut, followed by the x-cut and then the y-cut for both the 8 and

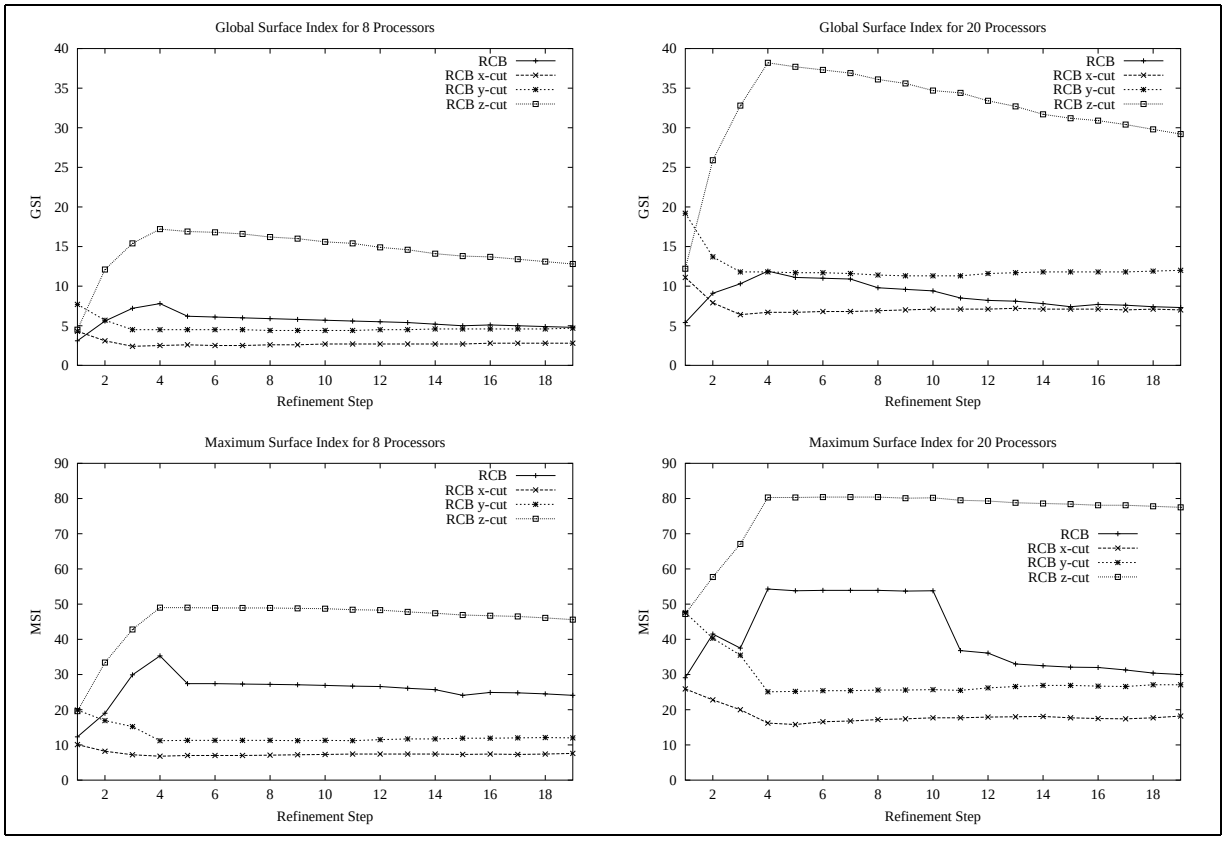

Figure 12: r_G (top) and r_M (bottom) at each adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. RCB x-cut achieved the best surface indices in all cases.

20 process cases. The original RCB algorithm had the lowest adj_A and adj_M for 20 processes, and similar values to y-cut on 8 processes.

The interprocess adjacency statistics were at first surprising. The initial motivation for the slice partitioning was to minimize interprocess adjacency. Yet, for the 20 process case, the original RCB algorithm had a lower adj_A at each refinement step than any of the slice methods. The structure of the mesh, more specifically, the refinement pattern, caused this. The domain away from the interface of the barrel and vent hole consists of large elements. The cut planes for the x-cut and y-cut RCB are determined almost completely by the highly refined interface. Slices were formed in this region, but away from the dense refinement, there are larger mesh elements that spanned three or more slices, causing a small number of these elements to be "adjacent" to more distant slices.

Figure 14 shows that x-cut RCB achieved a solution to the perforated shock tube most quickly, followed closely by the y-cut. These correlate closely to the surface index values in Figure 12. As with the studies in Section 4, surface indices were more important than interprocess adjacency for the Sun cluster.

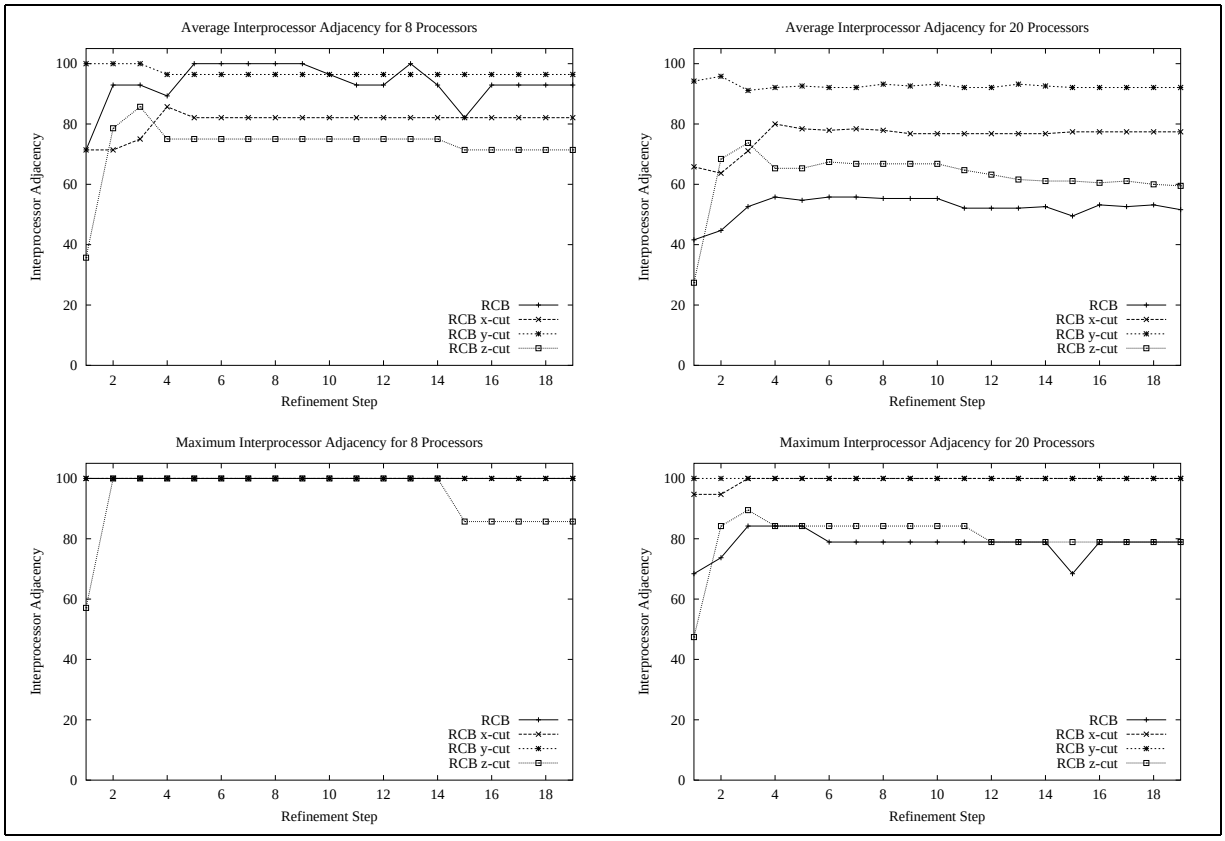

Figure 13: adj_A (top) and adj_M (bottom) at each adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processes. RCB z-cut achieved the best adj_A for 8 processes, while the original RCB was best in this measure for 20 processes.

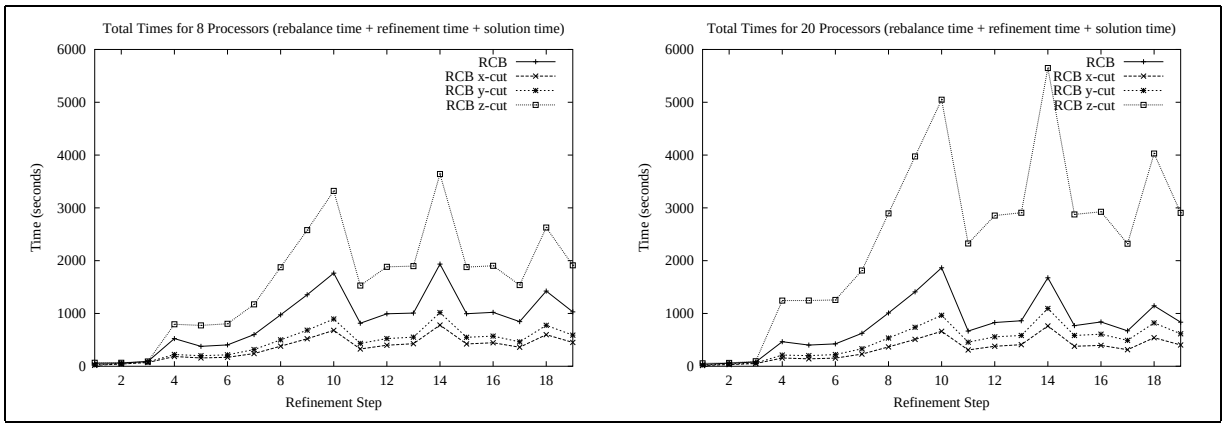

Figure 14: Total analysis times (time to solution) by adaptive refinement step for the perforated shock tube on 8 (left) and 20 (right) processors.

7 Discussion

These studies were an initial step in a more thorough study that will include additional load balancing algorithms, a larger variety of mesh structures and sizes, other solution methods,

and different computational environments. OCTPART and HSFC need to be included in all studies. Significantly larger meshes must be considered. Simulations involving millions of elements are becoming more common and new scalability concerns may arise. The perforated shock tube needed only 19 adaptive refinement steps for solutions that run for over an hour. Many problems of interest require much more frequent refinement, which would make the running time of the rebalancing a more significant factor. Much larger numbers of processors also must be considered, as well as a variety of processor and network speeds and hierarchies.

We can draw some conclusions from the study to this point. Though Jostle uses a more computationally expensive procedure, the resulting partitions, with lower surface index values than the others, allowed for a significantly faster solution process. Surface index was the more significant factor here, using the relatively slow Ethernet connection. Previous studies indicate that the interprocessor adjacency may be the most significant factor when communication is across a network with a higher bandwidth, but a relatively large latency such as the IBM SP [10].

We have determined that octant granularity for OCTPART should be set to allow a maximum of 20–40 objects per leaf octant. The partitions produced were nearly equal in quality across all granularities, but this range allows for the fastest running time. Imbalance may be introduced that are as large as the octant granularity, and in most cases, an imbalance of 20–40 objects is acceptable.

The slice partitioning modification to Zoltan's RCB algorithm is also promising. It incurred the same cost as the regular RCB, but produced partitions of significantly higher quality, when the axis to cut along was chosen correctly. This choice is highly dependent on the shape of the domain and on the refinement patterns. Use of slice partitioning on larger or more uniformly-refined meshes, and on systems with a faster network may reduce message startup latency. This will be investigated.

Previous studies have shown that a postprocessing "partition boundary smoothing" step can significantly improve the quality of partitions produced by the original OCTPART algorithm [23]. A similar procedure for the Zoltan geometric partitioners, especially OCTPART, RIB, and RCB, would be beneficial and is being investigated.

The appropriate choice of dynamic load balancing algorithm often depends on the target computer system [42]. A machine model for Zoltan to help guide architecture-aware load balancing is in development. As part of this effort, the studies described herein will be extended to a variety of parallel environments ranging from networks of workstations to the largest supercomputers.

Acknowledgments

Teresco was supported by contract 15162 with Sandia National Laboratories, a multi-program laboratory operation by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. Ungar was supported by the Williams College Summer Science research program.

References

[1] S. Adjerid and J. E. Flaherty. A moving finite element method for time dependent partial with error estimation and refinement. *SIAM J. Numer. Anal*, 23:778–796, 1986.

[2] S. Adjerid and J. E. Flaherty. A moving mesh finite element method with local refinement for parabolic partial differential equations. *Comput. Methods Appl. Mech. Engrg.*, 55:3–26, 1986.

[3] S. Adjerid, J. E. Flaherty, P. Moore, and Y. Wang. High-order adaptive methods for parabolic systems. *Physica-D*, 60:94–111, 1992.

[4] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *Proc. International Conference on High-Performance Computing*, pages 230–235, 1997.

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

[6] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36:570–580, 1987.

[7] T. Bially. Space-filling curves: their generation and their application to band reduction. *IEEE Trans. Inform. Theory*, IT-15:658–664, Nov. 1969.

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

[9] 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.

[10] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.

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

[12] B. Cockburn, S.-Y. Lin, and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-Dimensional systems. *J. Comput. Phys.*, 84:90–113, 1989.

[13] B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: General framework. *Math. Comp.*, 52:411–435, 1989.

[14] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[15] K. D. Devine, J. E. Flaherty, R. Loy, and S. Wheat. Parallel partitioning strategies for the adaptive solution of conservation laws. In I. Babuška, J. E. Flaherty, W. D. Henshaw, J. E. Hopcroft, J. E. Oliger, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, volume 75, pages 215–242, Berlin-Heidelberg, 1995. Springer-Verlag.

[16] K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John, and C. Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; Developer's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1376.

[17] K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John, and C. Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377.

[18] R. E. Dillon Jr. A parametric study of perforated muzzle brakes. ARDC Tech. Report ARLCB-TR-84015, Benét Weapons Laboratory, Watervliet, 1984.

[19] 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.

[20] 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.

[21] 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.

[22] J. E. Flaherty, R. M. Loy, M. S. Shephard, M. L. Simone, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Distributed octree data structures and local refinement method for the parallel solution of three-dimensional conservation laws. In M. Bern, J. Flaherty, and M. Luskin, editors, *Grid Generation and Adaptive Algorithms*, volume 113 of *The IMA Volumes in Mathematics and its Applications*, pages 113–134, Minneapolis, 1999. Institute for Mathematics and its Applications, Springer.

[23] 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 Distrib. Comput.*, 47:139–152, 1997.

[24] J. E. Flaherty, R. M. Loy, M. S. Shephard, and J. D. Teresco. Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In B. Cockburn, G. Karniadakis, and S.-W. Shu, editors, *Discontinous Galerkin Methods Theory, Computation and Applications*, volume 11 of *Lecture Notes in Compuational Science and Engineering*, pages 113–124, Berlin, 2000. Springer.

[25] B. Hendrickson. Load balancing fictions, falsehoods and fallacies. *Appl. Math. Modelling*, 25:99–108, 2000.

[26] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scien. Comput.*, 20(1), 1999.

[27] G. Karypis and V. Kumar. Parallel multivelel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[28] J. R. Levine, T. Mason, and D. Brown. *lex & yacc*. O'Reilly and Associates, Inc., 1992.

[29] R. M. Loy. *Adaptive Local Refinement with Octree Load-Balancing for the Parallel Solution of Three-Dimensional Conservation Laws*. PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, 1998.

[30] 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.

[31] W. F. Mitchell. The refinement-tree partition for parallel solution of partial differential equations. *NIST Journal of Research*, 103(4):405–414, 1998.

[32] M. Parashar and J. C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proc. 29th Annual Hawaii International Conference on System Sciences*, volume 1, pages 604–613, Jan. 1996.

[33] E. A. Patrick, D. R. Anderson, and F. K. Brechtel. Mapping multidimensional space to one dimension for computer output display. *IEEE Trans. Computers*, C-17(10):949–953, October 1968.

[34] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[35] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with space filling curves. *IEEE Trans. on Parallel and Distributed Systems*, 7(3):288–300, 1996.

[36] 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.

[37] 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.

[38] 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. In *Parallel Comput. in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.

[39] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. Tech. Rep. 00/IM/58, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, April 2000.

[40] V. E. Taylor and B. Nour-Omid. A study of the factorization fill-in for a parallel implementation of the finite element method. *Int. J. Numer. Meth. Engng.*, 37:3809–3823, 1994.

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

[42] 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.

[43] C. Walshaw and M. Cross. Parallel Optimisation Algorithms for Multilevel Mesh Partitioning. *Parallel Comput.*, 26(12):1635–1660, 2000.