Predictive Load Balancing for Parallel Adaptive Finite Element Computation

J. E. Flaherty R. M. Loy M. S. Shephard
B. K. Szymanski J. D. Teresco L. H. Ziantz

Computer Science Department and
Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY, U.S.A.

Abstract *This paper describes two predictive load balancing schemes designed for use with parallel adaptive finite element methods. We also provide an overview of data structures suitable for distributed storage of finite element mesh data as well as software designed for mesh adaptation and load balancing. During the course of a parallel computation, processor load imbalances are introduced at adaptive enrichment steps. The predictive load balancing methods introduced here use* a priori *estimates of work load for adaptive refinement and subsequent computation to improve enrichment efficiency and reduce total balancing time. These components have been used to build a system for solving compressible flow problems. Computational results on an IBM SP2 computer are presented for transient solutions of the three-dimensional Euler equations of compressible flow.*

Keywords: load balancing, parallel adaptive finite element methods, scientific computing

1 Introduction

The finite element method (FEM) has become a standard analysis tool for solving partial differential equations (PDEs). Adaptive FEMs have gained importance because they provide reliability, robustness, and efficiency in time and space. During the solution process, portions of the discretized domain are spatially refined or coarsened (h-refinement), the method order is varied (p-refinement), and/or the mesh is moved to follow evolving phenomena (r-refinement), to concentrate or dilute the computational effort in areas needing more or less resolution.

Parallel computation is essential for computationally demanding three-dimensional problems; however, it 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. However, with adaptive software, a good initial partition is not sufficient to assure high performance. Load imbalance caused by adaptive enrichment necessitates a dynamic partitioning and redistribution of data. In Section 2, we briefly detail tools developed by the Scientific Computation Research Center (SCOREC) at Rensselaer that facilitate the development and use of parallel adaptive finite element software.

These tools have been used in the construction of a parallel finite element code described in Section 3 which can solve three-dimensional conservation laws. Parallel mesh enrichment routines are used for spatial refinement and coarsening (h-refinement) [1]. Refinement is also performed in time using a spatially dependent local refinement method [2]. The ability to handle heterogeneous element weights allows us to balance processor loads based on the temporal as well as spatial refinement.

Previously, balancing followed mesh refinement and coarsening. The ability to predict and correct for imbalance prior to refinement can improve performance during the refinement stage while maintaining the same efficiency in the solution phase as compared to *a posteriori* balancing. Two strategies for this are described in Section 4. One focuses on balancing part of the spatial enrichment process itself. The other establishes computational balance by estimating the results of spatial refinement while also considering its impact on subsequent temporal refinement.

We solve compressible transient flow problems on an IBM SP2 computer to demonstrate the capabilities of the parallel adaptive system. The solution of a transient flow in a perforated shock tube is shown in Section 5 along with results showing the advantages of our predictive balancing methods. In Section 6, we discuss results and present future research directions.

2 SCOREC Tools

A collection of tools have been developed which allow software for parallel adaptive computation to be written in a uniform way. This enables the programmer to concentrate on issues specific to the problem at hand rather than the details of the underlying mesh structures or parallelization concerns.

Initial meshes are created using the *SCOREC Finite Octree Automatic Mesh Generator* [3]. The *SCOREC Mesh Database* (MDB) [4] provides an object-oriented hierarchical representation of a finite element mesh and a set of operators to query and update the mesh data structure. The basic mesh entity hierarchy consists of three-dimensional *regions*, and their bounding *faces*, *edges*, and *vertices*, with bidirectional links between mesh entities of consecutive dimensional order. In three-dimensional meshes, tetrahedral regions are used as finite elements while triangular faces serve as elements in two-dimensional meshes. Mesh entities are explicitly classified relative to a geometric model of the problem domain to allow for the appropriate representation of the geometry as the mesh is enriched. All entities can have attached attributes such as solution and error indicator data.

A *Parallel Mesh Database* (PMDB) [1, 5], which provides operators to create and manipulate distributed mesh data, is built on top of MDB. Using PMDB, each processor holds MDB data associated with a subset of the complete mesh. Entities along partition boundaries are shared by more than one processor and are maintained by a partition boundary data structure. The database provides fast query and update operations on the boundary. For example, a finite element procedure can obtain scatter/gather maps of data for use in its communication phase. PMDB also handles arbitrary multiple migration of elements between processors to maintain a balanced computation. Any element can be marked for migration, although boundary elements are frequently the ones that are migrated. The migration procedure uses an owner-updates rule to collect and update any modifications to links on partition boundaries. All interprocessor communication is done using the *Message Passing Interface* (MPI).

A dynamic load balancing scheme that operates on distributed mesh data is essential for adaptive computation. Recursive spectral bisection is perhaps the best static partitioner [6]. However, while *Multilevel Recursive Spectral Bisection* (MRSB) has improved the efficiency of RSB, its parallelization relies heavily on a shared memory architecture and is unlikely to be efficient in a true message passing environment [7]. Other enhancements to RSB [8, 9, 10] may make it more useful as a dynamic repartitioner, but doubts remain.

Three dynamic load balancing schemes are available for use with PMDB. *Iterative Tree Balancing* [1, 5] (ITB) performs repeated local migrations to achieve balance. Lightly loaded processors request load from their most heavily loaded neighbors. The algorithm views the requests as forming a forest of trees. Each tree is then linearized, and a logarithmic-time scan operation is used to compute load flows. Layers of elements on interprocessor boundaries are moved from heavily loaded to lightly loaded processors to achieve balance within each tree. This operation is iterated to achieve a global balance within a specified tolerance. *Parallel Sort Inertial Recursive Bisection* [1] (PSIRB) repeatedly bisects the domain in the direction orthogonal to its principal axis of inertia (inertial coordinates are sorted in parallel). *Octree Partitioning* [2] (OCT) uses octree topology to create a one-dimensional ordering of the octree nodes. The ordered list of nodes is divided into segments corresponding to nearly equal load. Members of any given segment tend to be spatially adjacent and, thus, form a good partition. The use of space-filling curves [11] would be another alternative that keeps neighboring elements of the ordering in close spatial proximity.

With h-refinement, the cost function that reflects the computational load on each processor is generally chosen as the number of elements on a processor. However, heterogeneous costs are necessary when (*i*) using p-refinement or spatially-dependent solution methods, (*ii*) using spatially-dependent time steps (Section 3), (*iii*) enforcing boundary conditions, or (*iv*) using predictive load balancing (Section 4). PMDB provides an element weighting scheme that can be used to address each of these needs, and both ITB and OCT have been extended to use this scheme.

3 Adaptive Techniques

Results presented in Section 5 were obtained using a parallel adaptive Euler solver in which three-dimensional conservation laws are discretized using a discontinuous Galerkin finite element method [12, 13, 14]. More details on the solver may be found in [2]. The software makes use of both spatial and temporal refinement to concentrate computational effort in areas of the problem domain where it is most needed.

The SCOREC *mesh enrichment* procedure [1] performs spatial (h-) refinement and coarsening in parallel using error indicator information and threshold values provided by an application code. This information is used to mark mesh edges to be coarsened, refined, or unchanged. Mesh enrichment is done in stages following the order of (*i*) coarsening, (*ii*) optimization, (*iii*) refinement, (*iv*) optimization, (*v*) refinement vertex snapping, and (*vi*) optimization. Note that all optimization stages are optional. Coarsening collapses a marked edge to one of its end vertices. Regions connected to the collapsed vertex that cease to exist are deleted to form a polyhedral cavity, and the faces of the cavity are connected to the target vertex to form new mesh regions. Mesh optimization improves the quality of triangulations with respect to a given criterion (*e.g.* , element shape). Refinement is performed using subdivision patterns. First, faces on partition boundaries with marked edges are triangulated using two-dimensional refinement templates. Each processor then independently applies three-dimensional patterns that have been determined for every configuration of marked edges (Figure 1). The enrichment process has an over-refinement option which reduces element shape degradation at the expense of creating more elements. In the final stage of enrichment, any vertex created by the refinement process that is classified as belonging to a curved model boundary must be "snapped" to the appropriate model entity to ensure mesh validity with respect to the geometry of the problem domain. Due to its low per-iteration costs, ITB is often executed for a few iterations between stages of mesh enrichment without the large time penalty of a global repartitioning.

The SCOREC temporal *Local Refinement Method* (LRM) selects spatially-dependent time steps based upon the Courant stability condition for explicit time integration. Thus, in a given time period, a smaller number of larger time steps will be taken on large elements, and the opposite will occur on the small elements. The solution is synchronized periodically to calculate error estimates or indicators and to perform h-refinement. The sychnronization "goal time" to which we wish to advance the solution is typically determined to be a small multiple of the smallest time step on any element of the mesh.

The time step for an element Ω_j is determined from the Courant condition as

$$\Delta t_j = \alpha \frac{r_j}{v_j}, \quad \alpha \leq 1, \tag{1}$$

where r_j is the radius of Ω_j's inscribed sphere and v_j is the maximum signal speed on Ω_j. For the Euler equations, v_j is the sum of the fluid's speed and the sound speed. The parameter α is introduced to maintain stability in areas of mesh gradation. We empirically chose $\alpha = 0.65$, but a more thorough analysis is necessary.

Previously, each element could only be advanced by the smallest stable time step of any element in the mesh. This *Method of Lines* (MOL) was far less efficient than the technique described above.

LRM introduces complications in computational load balancing in that the work load varies from element to element. To take this into account, a size-based weighting scheme is used. The weight w_j assigned to each element Ω_j is

$$w_j = \frac{1}{r_j} \tag{2}$$

where r_j is the radius of Ω_j's inscribed sphere. Thus, smaller elements are given larger weights since a large number of time steps will be taken on them during the computation phase. Balancing the load amounts to distributing weight evenly among the processors.

4 Predictive Load Balancing

Normally, a full load balancing follows enrichment, although a few ITB iterations are performed between stages of the enrichment to maintain some balance. However, the work done during the refinement stage procedure is not necessarily proportional to the number of elements on a processor at the start of refinement. Thus, balancing using a unit load per element may not be appropriate to maintain a high degree of efficiency during the enrichment process. Furthermore, anticipating the results of the refinement stage in conjunction with a user-supplied weighting function can achieve a reasonable computational load balance while moving a coarser structure between processors compared to migration after refinement.

We improve balance during the enrichment process by using error indicator data generated during the preceding computation phase to select element weightings and perform load balancing before the refinement stage of the enrichment process. A similar technique has also been used by Oliker, Biswas, and Strawn [15] in their enrichment procedure. This *a priori* load balancing reduces imbalance during the refinement stage of enrichment and results in fewer elements being migrated than the standard method. Without predictive balancing, some processors may have nearly all of their elements scheduled for refinement which can lead to a memory overflow. In such an instance, the predictive technique would tend to disperse these elements among several other processors, thus reducing the likelihood of a memory allocation problem. At present, predictive load balancing is performed using either ITB or OCT; however, any load balancing procedure that recognizes elemental weights may be used.

Weight assignments for h-refinement are made after edges have been marked for splitting based on error indicator information and a refinement threshold but before the refinement is actually performed. The weighting is based on the three-dimensional subdivision patterns used by the refinement algorithm (Section 3) as indicated in Figure 1 by the numbers in parentheses. An element is assigned a unit weight if none of its edges are marked.

Using this technique, a small imbalance still may occur during the refinement. An element remains an atomic unit regardless of its weight, so its weight cannot be subdivided to achieve exact balances in all cases. For some templates, the refinement can perform more subdivisions than predicted by the original edge marking

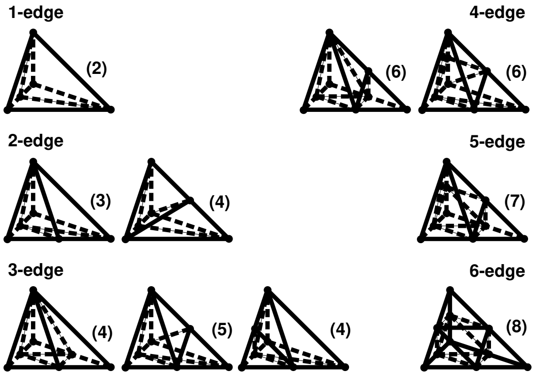

Figure 1: Weights for subdivision patterns and predictive load balancing.

to satisfy topological constraints which cannot be determined efficiently beforehand. For similar reasons, greater imbalance can be produced when over-refinement (Section 3) is used to maintain better element shapes. Imbalance may also occur during refinement if many elements with large weights come to reside on a few processors while the remaining processors have mostly unit weight elements.

This method of predictive balancing has been used successfully to balance a transient flow problem which was solved using the parallel Euler solver with the MOL [5]. When the workload per element in the computation phase is homogeneous, this technique will also balance the subsequent numerical calculations. Hence, it may be referred to as *Uniform element workload Predictive balancing* (UP). In addition to the previously mentioned sources of refinement imbalance, the computation phase may be imbalanced as a result of the migration of mesh elements that occurs during refinement vertex snapping and mesh optimization. Imbalance may also result from the deletion and creation of elements in the optimization stage. In practice, however, these enrichment stages tend to produce small localized changes in the mesh, so they generally have little impact on the subsequent computation.

As discussed in Section 2, not every problem has a uniform workload per element. To account for this, UP has been extended to accept a user-defined cost function to be used in conjunction with the predictive method. This allows the user to design an element-based load estimator tailored to the solution technique. The predictive balancing library makes the element subdivision weights available to the estimator function, so it can take into consideration the effects of spatial refinement in generating its load values.

As an example, consider the spatially-dependent temporal refinement technique discussed in Section 3 (LRM). To balance after spatial enrichment, element weights are assigned as in (2). If an element Ω_j is to be subdivided into w elements, $\Omega_{j_1}, \ldots, \Omega_{j_w}$, we may approximate the inscribed radius of Ω_{j_i}, $1 \leq i \leq w$, as

$$r_{j_i} = \left(\frac{3}{4\pi} \cdot \frac{V_j}{f(w)} \right)^{\frac{1}{3}} \tag{3}$$

where V_j is the volume of the inscribed sphere of element Ω_j and $f(w)$ is a function of w. Some preliminary results for this simple estimator with $f(w) = w^3$ are presented in Section 5.

Using *Variable element workload Predictive balancing* (VP) generates less data movement during balancing than *a posteriori* balancing and, given a good estimator, eliminates the need for balancing after refinement. However, since the weighting used in the balancing is based not only on the number of elements produced by refinement but also on other factors to balance the computational load afterwards, this is a compromise between balancing the refinement stage of enrichment and allowing computation to be balanced without an *a posteriori* balancing. In addition, there may be solution techniques that generate heterogeneous workloads which are not easily estimated before new elements are created.

5 Results

Consider the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent. This problem was motivated by flow studies in perforated muzzle brakes for large calibre guns [16]. We match flow conditions to those of shock tube studies of Dillon [16] and Nagamatsu *et al.* [17]. Our focus is on the quasi-steady flow that exists behind the contact surface for a short time. The initial mesh contains 77,231 tetrahedral elements. The larger cylinder (the shock tube) initially contains air moving at Mach 1.23 while the smaller cylinder (the vent) is quiescent. A Mach 1.23 flow is prescribed at the tube's inlet. Figure 2 illustrates the Mach number with velocity vectors at time $t = 0.6$ in the simulation. These flow features compare favorably with experimental and numerical results of Nagamatsu *et al.* [17]. All results were obtained using 16 processors of an IBM SP2 computer.

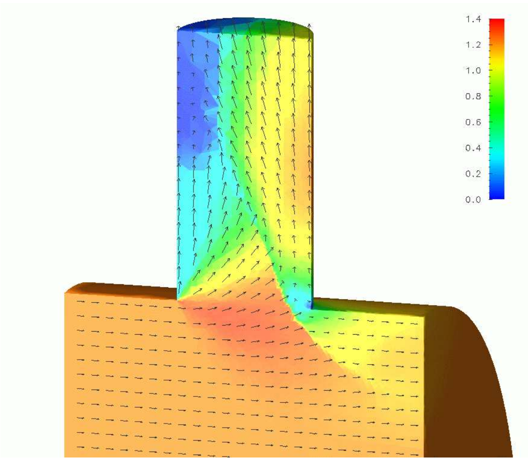

Figure 2: Projections of the Mach number and velocity vectors onto the surfaces of a perforated cylinder at time 0.6.

As previously indicated, the standard nonpredictive method executes a few iterations of ITB between spatial enrichment stages followed by a global size-weighted repartitioning after enrichment. Global repartitioners tend to maintain better partition quality than ITB [18]. Thus, OCT was used in balancing after enrichment for nonpredictive runs.

For tests of the predictive methods, both ITB and OCT were used with UP (denoted UPITB and UPOCT). Since UP will not balance load for the LRM, enrichment is followed by OCT (but OCT is not considered in the UP results as we are evaluating the performance of UP in balancing enrichment). No such global repartitioning is necessary after VP, but the predictive balancing must maintain reasonable partition quality. Thus, only OCT was used with VP (VPOCT) since ITB is not a global repartitioner.

The over-refinement and mesh optimization options were turned off for all runs. For a given quantity, q, being compared, the relative percent differences given in this section were computed as

$$q_{\text{diff}} = \frac{q_{NONPRED} - q_{PRED}}{q_{NONPRED}} \times 100 \qquad (4)$$

where $q_{NONPRED}$ is the quantity for the nonpredictive method and q_{PRED} is the quantity for a predictive technique. Thus, a positive q_{diff} indicates that the predictive method outperforms the nonpredictive one.

Figure 3 illustrates the reduction in data migration that a predictive method can produce over a nonpredictive one. The figures for ITB are summed across all iterations of the balancer during each mesh enrichment. Though there are variations across runs, the volume of data moved by UPITB is 43-79% less than ITB per mesh. Overall, UPOCT outperforms ITB by 21% in this regard even though there is a spike for mesh 5. While the changes in meshes 1-4 were relatively large, the changes in meshes 5 and 6 were reasonably small. Given a small mesh change, ITB will tend to outperform a global repartitioner in data migration, and, as seen in mesh 5, a global repartitioner may make a more radical change to the partitioning than an iterative method, even for small mesh changes.

The improvement in data migration using VPOCT is shown in Figure 4. Since VPOCT attempts to balance the computational load without any further balancing after a spatial enrichment, its data movement is compared to the combined migration generated by ITB during enrichment and by OCT afterwards. The average improvement on a mesh by mesh basis ranged from 71-92%. Note that the increased migration for mesh 5 in the nonpredic-

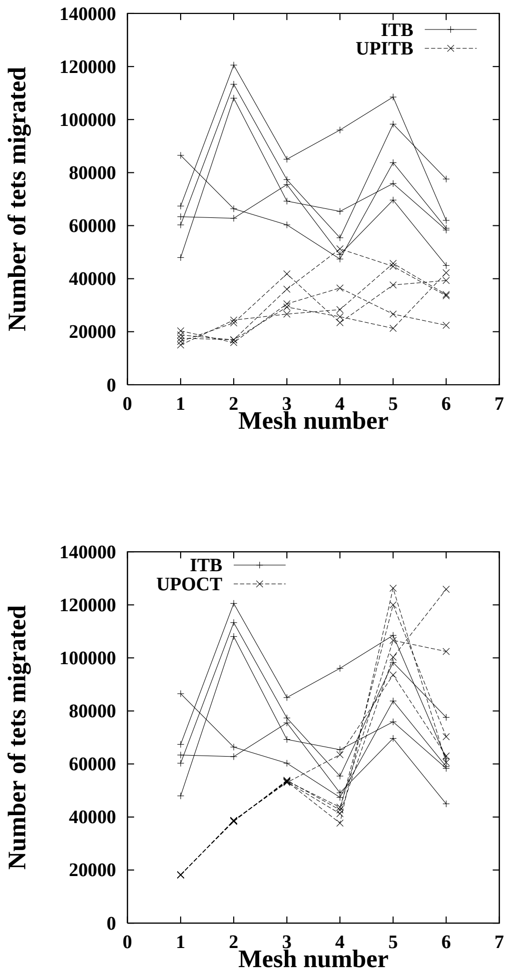

Figure 3: Comparison of data migrated for standard nonpredictive method vs. UPITB (top) and vs. UPOCT (bottom).

tive case is due mainly to the OCT balancing after enrichment; a similar jump was seen in Figure 3(bottom). Curiously, VPOCT did not produce the same effect for this mesh.

For the UP methods, the accuracy of the estimator for balancing refinement is given by the imbalance in the number of elements produced. Figure 5 shows that the percent imbalance is quite small, ranging from 1-5%. Thus, a reasonable balance is provided for the refinement stage of the enrichment process.

For the VP method, the accuracy of the estimator is reflected in the resulting weight im-

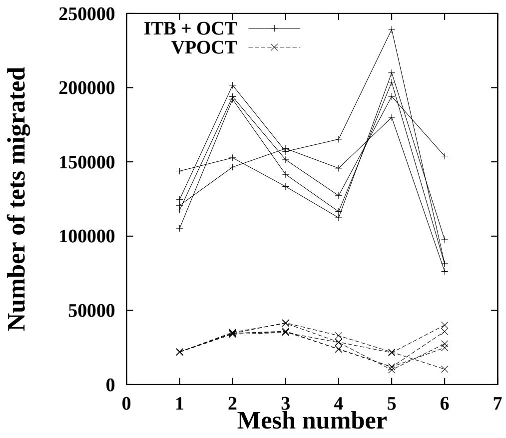

Figure 4: Comparison of data migrated for the standard nonpredictive method vs. VPOCT.

balance. The values given in Figure 6 were computed using the weight function given by (2). The fact that VPOCT did not approach a 10-20% imbalance level until mesh changes became less radical later in the run may indicate that (3) is too simple of an estimator to produce solver performance close to that produced by *a posteriori* size-weighted balancing.

Combined time spent in spatial enrichment and balancing for a sequence of runs is presented in Figure 7. These results are expressed as percent differences comparing each predictive run to a corresponding nonpredictive run. For the UP methods, the times spent in enrichment (including balancing during enrichment) are compared. For VP, the comparison is between VPOCT enrichment time and the combined enrichment and *a posteriori* OCT balancing time for the nonpredictive method. The graph shows a 5-60% improvement in combined enrichment and balancing times for UP and a 53-79% improvement for VP.

Figure 8 shows the percent improvements in the average combined time taken by refinement and balancing for the methods as compared to the nonpredictive scheme. Though the variations in runs seen in Figure 7 prevent drawing a firm conclusion as to which UP method performed better, it is clear that both UP and VP,

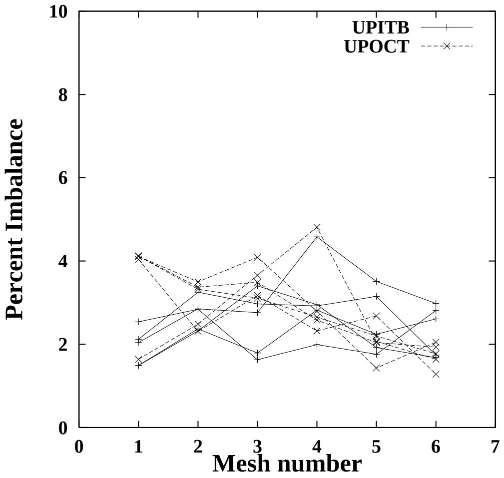

Figure 5: Estimator accuracy for UPITB and UPOCT.

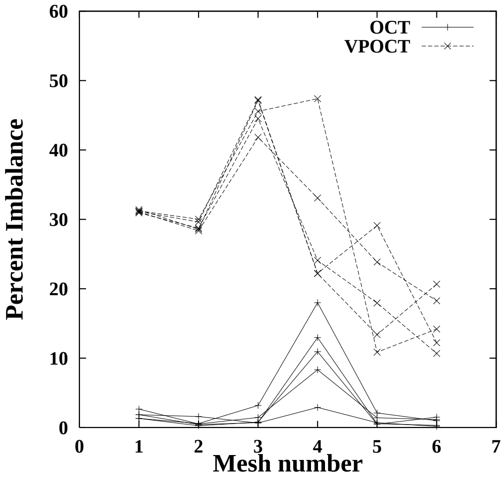

Figure 6: Estimator accuracy for VPOCT.

in general, far outperformed nonpredictive balancing.

The variation in runs is due in large part to variations in the spatial enrichment process. Mesh data migrated to a processor are added to the local data structure's linked lists in the order they arrive. Because enrichment is performed in the order that elements are encountered, different meshes can result from the same input mesh and error indicators. Additionally, the enrichment process may perform different mesh operations depending on the partitioning so as to avoid unnecessary element migration. The impact of this on enrichment and balancing times across runs has already been seen. There is also a large influence on solution times because the size distribution of elements has a major effect on the LRM used by the solver. Thus, variations in solution times prevent drawing any conclusions on the relative performance of VPOCT and size-weighted OCT after enrichment in terms of time spent in computation.

6 Conclusions

Both types of predictive balancing tend to reduce the volume of data migrated during balancing as compared to nonpredictive methods, showing a 41% average improvement for this set of runs with UP and 81% with VP. In addition, combined enrichment and balancing times were improved by approximately 36% on average for UP and 67% on average for VP. The imbalance during the refinement stage of the enrichment process and any subsequent uniform workload computation was in the 1-5% range for UP, indicating that our UP weighting scheme was quite accurate. Based on the results of VP estimation accuracy, a better estimator may be needed for balancing the LRM computation after enrichment. However, we believe that a different estimator will still produce similar advantages in data migration and times for enrichment and balancing. Even using the current estimator, VP may approximate a balanced computation closely enough for an iterative balancer to spend little time in establishing a more exact load balance after enrichment.

A predictive method generally performs best when applied to problems in which the time spent in computation is of the same order of magnitude as the combined enrichment and balancing times. This technique reduces the time spent in enrichment and allows a single load balancing call to replace several calls performed by nonpredictive balancing. However, computation performed on the resulting parti-

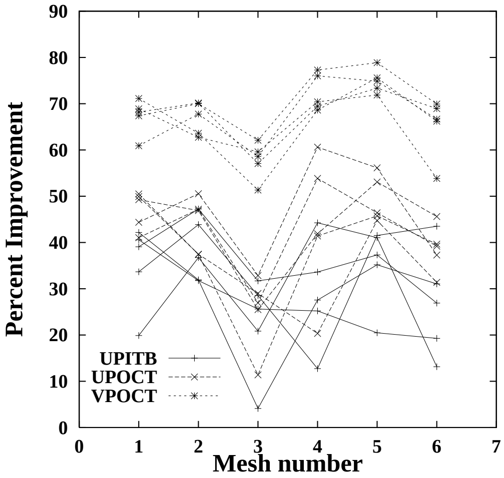

Figure 7: Time spent in enrichment and balancing for a sequence of five predictive runs relative to nonpredictive runs.

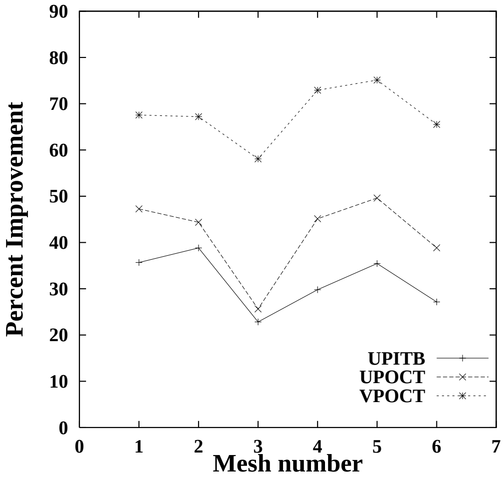

Figure 8: Average improvement for runs of Figure 7.

tions is generally no more efficient than calculations done on partitions generated by an *a posteriori* method. Thus, a problem which adapts the mesh infrequently will show less improvement in overall performance than one that requires frequent enrichments.

Adaptive p-refinement will be added to the system in the future and will require extension of the predictive balancing to account for the number of degrees of freedom associated with each element. In addition, we are investigating ways to reduce or eliminate factors that produce variations across runs having the same parameters. Currently, these variations make the use of wall-clock timings in evaluating solution efficiency very difficult.

Acknowledgements

We would like to thank Mark Beall, Carlo Bottasso, Hugues de Cougny, Hema Murty, and Wesley Turner for the generous use of their software and many valuable suggestions. Computer systems used in the development and analysis runs include the 36-node IBM SP2 computer at Rensselaer, the 400-node SP2 at the Maui High Performance Computing Center, and the 512-node SP2 at the Cornell Theory Center. Authors were supported by AFOSR Grant F49620-95-1-0407, ARO grant DAAH04-95-1-0091, and NSF Grant CCR-9527151.

References

[1] 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 Computing in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.

[2] 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. Submitted for publication, 1997.

[3] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *Int. J. Numer. Meth. Engng.*, 32(4):709–749, 1991.

[4] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure.

To appear *Int. J. Numer. Meth. Engng.*, 1997.

[5] 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. SCOREC Report 22-1996, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, 1996. Submitted for publication.

[6] A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, 1990.

[7] S. T. Barnard. PMRSB: parallel multilevel recursive spectral bisection. In Frank Baker and Janus Wehmer, editors, *Proc. Supercomputing 95*, San Diego, December 1995.

[8] A. Sohn, R. Biswas, and H. D. Simon. Impact of load balancing on unstructured adaptive computations for distributed-memory multiprocessors. In *Proc. Eighth IEEE Symposium on Parallel and Distributed Processing*, pages 26–33, New Orleans, LA, October 1996.

[9] R. Van Driessche and D. Roose. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, 21:29–48, 1995.

[10] C. H. Walshaw and M. Berzins. Dynamic load balancing for PDE solvers on adaptive unstructured meshes. *Concurrency: Practice and Experience*, 7(1):17–28, 1995.

[11] A. Patra and J. T. Oden. Problem decomposition for adaptive *hp* finite element methods. *Comp. Sys. Engng.*, 6(2):97, 1995.

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

[13] 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.

[14] 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.

[15] L. Oliker, R. Biswas, and R. C. Strawn. Parallel implementaion of an adaptive scheme for 3D unstructured grids on the SP2. In *Proc. 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems*, Santa Barbara, 1996.

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

[17] H. T. Nagamatsu, K. Y. Choi, R. E. Duffy, and G. C. Carofano. An experimental and numerical study of the flow through a vent hole in a perforated muzzle brake. ARDEC Technical Report ARCCB-TR-87016, Benet Weapons Laboratory, Watervliet, 1987.

[18] 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 Boleslaw K. Szymanski and Balaram Sinharoy, editors, *Proc. Third Workshop on Languages, Compilers, and Runtime Systems*, pages 265–277, Troy, 1996.