Abstract

We describe predictive load balancing schemes for use with parallel adaptive finite element methods. We provide an overview of an infrastructure suitable for distributed storage of finite element mesh data as well as software designed for mesh adaptation and load balancing. The predictive load balancing methods improve enrichment efficiency and reduce total balancing time by using *a priori* estimates of work needed for adaptive refinement and subsequent computation. Analysis codes developed with these components for solving compressible flow problems are used to obtain predictive load balancing results on an IBM SP2 computer. Our three-dimensional test problems involve compressible steady flow about a rotor blade and transient flow in a perforated shock tube.

1 Introduction

The finite element method (FEM) is a standard tool for solving systems of partial differential equations (PDEs). Adaptive methods provide greater reliability, robustness, and efficiency by refining or coarsening the mesh (h-refinement), varying the method order (p-refinement), and/or moving the mesh to follow evolving phenomena (r-refinement) to concentrate or dilute the computational effort and to satisfy prescribed accuracy criteria.

When solving demanding three-dimensional problems, parallel computation is essential to obtain solutions in reasonable times. However, in order to exploit the benefits of parallelism, one must face the added complexities of processor load balancing, interprocessor communication coordination, and distributed data management. In some circumstances, the parallelization of FEM programs may be done using a static partitioning of the mesh across the cooperating processors. However, for an adaptive computation, a good initial partition is not sufficient to assure high performance throughout the computation. Dynamic repartitioning is needed to correct for load imbalance introduced by adaptive enrichment.

We describe reusable tools that facilitate the development and use of parallel adaptive finite element software (Section 2) in many applications, including steady [4] and unsteady [14] compressible three-dimensional fluid flow. Parallel mesh enrichment routines are used for spatial h-refinement [24]; however, h-refinement may also performed in time both to accelerate convergence to a steady state [4] and to improve the efficiency of transient computations [14]. The consideration of heterogeneous element weights allows us to balance processor loads based on both the temporal and spatial enrichment.

Typically, processor load balancing follows an h-refinement step. The ability to predict and correct for imbalance prior to enrichment can improve performance during refinement and coarsening while maintaining a balanced computation during the successive solution phase. Strategies for predicting a load imbalance are described in Section 4. We use error indicator and refinement threshold values to assign weights to elements in proportion to the number of elements that will be created during refinement. Balancing these weights across the processors before enrichment distributes the work of the enrichment process and eliminates the need for *a posteriori* load balancing. Balancing the work of the enrichment stage is not necessarily the same as balancing the computation phase. A cost function designed solely to maintain *a posteriori* computational balance may not provide for an efficient enrichment. Combining the predicted results of the refinement stage with an appropriate weighting function allows reasonable computational load balance to be achieved while moving a coarser (smaller) structure between processors as compared to migration after enrichment.

The capabilities of the parallel adaptive system are demonstrated by solving three-dimensional compressible flow problems involving the steady flow about a rotor blade and the transient flow in a perforated shock tube (Section 5). Results illustrate the physical solution and the advantages of predictive load balancing strategies. Additional discussion and future research directions follow in Section 6.

2 Mesh Tools

Reusable tools allow the author of an analysis code to concentrate on issues specific to a given problem rather than the details of the underlying data structures or parallelization issues. A set of such tools which facilitate implementation of reusable parallel adaptive software have been developed for parallel mesh generation, adaptive mesh enrichment, and dynamic load balancing.

2.1 Mesh Data Structures

Mesh data is stored using the SCOREC (Scientific Computation Research Center) Mesh Database (MDB) [2], which provides operators to build, query and update hierarchical finite element data structures. The entity hierarchy consists of three-dimensional *regions* (which serve as elements for calculations here), their bounding *faces*, *edges*, and *vertices*, with bidirectional links between mesh entities of consecutive dimensional order. The Finite Octree Automatic Mesh Generator [25] creates initial meshes from a geometric (CAD) model of the problem domain. This geometric model is used to maintain appropriate domain geometry as the mesh is generated and subsequently enriched, through explicit classification of mesh entities onto the model.

A Parallel Mesh Database (PMDB) [12, 24] is built on top of MDB to create and manipulate distributed mesh data. The Message Passing Interface (MPI) is used for interprocessor communication. Each processor holds MDB data associated with a subset of the complete mesh. Partition boundary data structures account for entities along partition boundaries that are shared by more than one processor. Fast boundary query and update operations are provided. Arbitrary multiple migration operators are used to move elements between processors to maintain a balanced computation. Any element may be marked for migration, but it is frequently boundary elements which are moved. Partition boundary structures are updated during migration using an owner-updates rule, allowing the migration to scale with the number of processors.

2.2 Dynamic Load Balancing

To be useful in a parallel adaptive environment, a dynamic load balancing procedure must operate on distributed mesh data. Recursive spectral bisection (RSB) [20] and, more recently, the multilevel partitioning algorithms included within the MeTiS system [16], are generally regarded as good static partitioners. Enhancements to RSB [1, 26, 28, 29] may make it more useful as a dynamic repartitioner, but it is generally regarded as being too slow. The recent extension of the MeTiS algorithms to operate in parallel (ParMeTiS [22]) make them worthy of consideration in this context. The JOSTLE package [30] provides an iterative partition optimization process which may also be useful for balancing adaptive computation. Iterative Tree Balancing [12, 24] (ITB) is a diffusive algorithm based on repeated local migration. Each processor requests load from its most heavily loaded neighbor, and these requests are used to create a forest of trees. Load flows are computed, and elements on interprocessor boundaries are moved from heavily loaded to lightly loaded processors to balance each tree. This process may be iterated to a global balance. Parallel Sort Inertial Recursive Bisection [24] (PSIRB) is an implementation of inertial bisection. The domain is recursively bisected in a direction orthogonal to its principal axis of inertia. Inertial coordinates are sorted in parallel, allowing operation on a distributed mesh. Octree Partitioning [14] (OCT) uses a one-dimensional ordering of the nodes of an octree structure underlying the mesh. The ordered list of nodes is divided into segments corresponding to nearly equal load. Octants adjacent to one another in the ordered list tend to be spatially adjacent and, thus, form a nearly contiguous partition. The use of space-filling curves [19] produces similar results, keeping neighboring elements of the ordering in close spatial proximity. Our software includes implementations of ITB, PSIRB, and OCT which operate directly on distributed PMDB meshes, and an interface to convert our data structures into a format usable by the ParMeTiS package.

When the computation required on each element is uniform, the computational load on each processor may be estimated using a cost function which is simply the number of elements on a processor. However, 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), a more complex cost function is necessary. These costs may be reflected in a per-element weight. In this context, load balancing consists of balancing these weights across the processors. ITB, PSIRB, and OCT each allow such weighting when determining partitions.

3 Adaptive Techniques

3.1 Spatial Refinement

Spatial h-refinement is performed on distributed meshes using the SCOREC mesh enrichment procedure [24]. Mesh edges are marked to be coarsened, refined, or to remain unchanged using error indicator information and threshold values provided by the analysis code. The enrichment is a multi-step process done in the order of (*i*) coarsening, (*ii*) optimization, (*iii*) refinement, (*iv*) optimization, (*v*) refinement vertex snapping, and (*vi*) optimization, with each optimization stage being optional. Coarsening is done by an edge-collapsing process. Mesh optimization improves the quality of triangulations with respect to a given criterion (*e.g.*, element shape). Refinement is performed using subdivision patterns. Faces which are on interprocessor boundaries and have marked edges are triangulated using two-dimensional refinement templates, then each processor independently applies three-dimensional patterns to the elements (Figure 1), which account for all possible configurations of marked edges. A global shape control parameter can prevent any step of the enrichment process from creating elements with shapes worse than a specified quality measure. Optional over-refinement reduces element shape degradation at the expense of creating more elements. In the refinement vertex snapping stage, any newly created vertex classified as belonging to a curved boundary must be "snapped" to the appropriate model entity to ensure mesh validity with respect to the geometry of the problem domain. With a small per-iteration cost, ITB is often executed for a few iterations between stages of mesh enrichment to improve balance without a large time penalty.

3.2 Temporal Refinement

In a time-dependent calculation (see Section 5.2), we use the Local Refinement Method (LRM) [14], in which elements choose spatially-dependent time steps based upon a Courant stability condition for explicit time integration. To advance the global solution time, a small number of larger time steps will be taken on large elements, while smaller elements, with smaller stable time steps, will require a larger number of time steps. Periodic synchronization points are necessary to calculate error estimates or indicators and to perform h-refinement. The sychnronization "goal time" is typically a small multiple of the smallest time step on any element of the mesh. The simpler Method of Lines (MOL) approach, where each element is advanced only by the smallest stable time step of any element in the mesh, is far less efficient than the LRM.

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 speed of sound. The parameter α, chosen empirically as 0.65, is intended to maintain stability in areas of mesh gradation; however, a more thorough analysis of the selection of this value is necessary.

LRMs complicate load balancing, since work contributions between synchronizations vary with element size. This is corrected by weighting Ω_j by

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

Smaller elements are assigned larger weights since they will take additional time steps during the computation phase. Balancing requires a uniform distribution of the weighted elements across the processors of the computer.

4 Predictive Load Balancing

A standard balancing scheme does not take h-refinement into account. By estimating the results of refinement and moving data appropriately beforehand, predictive balancing can improve the performance of enrichment. Further, for solution techniques which require the same work per element to generate a solution, the workload in the subsequent computation is also balanced. This form of predictive balancing is called Uniform element workload Predictive balancing (UP). UP was effective in balancing a transient flow problem solved using an explicit parallel adaptive Euler solution technique with time steps determined by the MOL [12]. Since the workload per element in the computation phase was homogeneous in this case, this technique also balanced the numerical calculations subsequent to each adaptive step. UP has also been used in tests with two different balancing algorithms to improve enrichment and balancing times for a nonuniform workload per element solution technique [15]. The goal was to balance refinement using UP. The heterogeneous computational workload was balanced by a standard weighted partitioner.

Predictive balancing can be applied more directly to problems with nonuniform workload per element. For example, the amount of computation per element varies based on element size for the LRM discussed in Section 3.2. In this case, a function designed to estimate the workload of an element given its subdivision can be used to balance computational load. Thus, this instance of predictive balancing is named Variable element workload Predictive balancing (VP).

4.1 Uniform Workloads

Prior to refinement, mesh edges are marked for splitting based on error indicator information and a refinement threshold. These markings determine the three-dimensional refinement templates that will be applied during refinement (Section 3.1), and they may also be used to select appropriate weighting factors for predictive balancing. The numbers in parentheses in Figure 1 indicate the weights associated with each edge marking pattern. Each weight indicates the number of elements that will result from refinement of the original element. An element with no edges marked for refinement is assigned a unit weight factor. Partition loads are then recomputed using these assignments, and a weighted load balancing is performed. Any algorithm that supports weighted elements may be used. A similar technique has been developed by Oliker, Biswas, and Strawn [18].

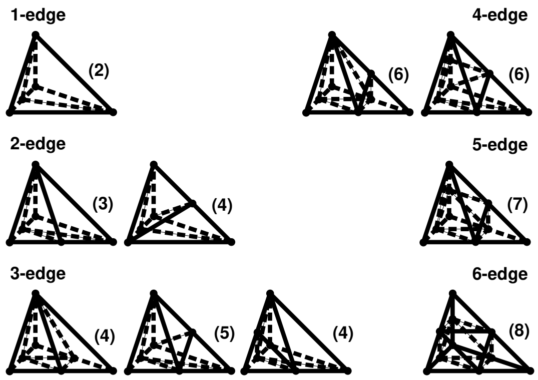


Figure 1: Weights for subdivision patterns and predictive load balancing.

Several factors may lead to small imbalances during refinement even with predictive balancing. An element is an atomic unit so it cannot be subdivided to achieve an exact balance. Geometric constraints, which are expensive to determine *a priori*, may force additional subdivisions to maintain mesh validity with respect to the domain geometry. Greater imbalances may result when over-refinement is used to maintain element shape regularity. Imbalances also occur when elements with large weights are assigned to a few processors while lightly-weighted elements are assigned to the majority of processors.

With UP, an additional source of computational imbalance may arise as a result of (*i*) the mesh element migration that occurs during refinement vertex snapping and (*ii*) the migration, deletion, and creation of elements during the optimization stage. In practice, however, these changes tend to be small and localized with little impact on the subsequent computational load balance.

4.2 Variable Workloads

A uniform workload per element is not appropriate in many situations where predictive balancing is desirable. UP has been extended to accept an element-based cost function which can be tailored to a specific solution procedure. The predictive balancing library provides access to the element subdivision weights (Figure 1), so the function can consider the effects of spatial refinement when generating load values.

As an example, consider the LRM technique discussed in Section 3.2. If an element Ω_j is to be subdivided into n elements, $\Omega_{j_1}, \ldots, \Omega_{j_n}$, we approximate the inscribed radius of Ω_{j_i}, $1 \leq i \leq n$, as

$$\hat{r}_{j_i} = \left(\frac{3}{4\pi} \cdot \frac{V_j}{n}\right)^{\frac{1}{3}} \tag{3}$$

where V_j is the volume of the inscribed sphere of Ω_j. The weight for Ω_j is then computed as

$$\hat{w}_j = \frac{n}{\hat{r}_{j_i}} \ . \tag{4}$$

Like UP, VP generates less data movement during balancing than *a posteriori* balancing and, given a good cost function, eliminates the need for balancing after refinement. However, since the weighting is based not only on the number of elements produced by refinement but also on factors needed 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 Examples

In this section, we present predictive balancing results for tests run on two types of problems. The first example computes the steady flow about a helicopter blade. The problem illustrates how accurately UP balances refinement and its capacity to prevent unnecessary memory overflow. The second case models the transient flow in a perforated tube. It shows that VP can balance the LRM almost as accurately as an *a posteriori* weighted partitioner while migrating less data and improving combined balancing and enrichment time.

5.1 Steady Rotorcraft Flow

The UP method has been applied to a steady flow problem using a parallel adaptive FEM program called Fluid Analysis in Space-Time (FAST) [4]. FAST uses the tools of Section 2 to analyze compressible fluid flows by the time discontinuous Galerkin/least-squares method [23] with a Generalized Minimum Residual (GMRES) [21] algorithm employed to solve linear algebraic systems.

We use FAST to study the steady flow of a Caradonna-Tung rotor blade [6]. This two-bladed rotor is untwisted and untapered with an aspect ratio of 6 and has a NACA 0012 airfoil section. The case analyzed has a tip Mach number of 0.439, a collective setting of 8^o and a rotor speed of 1250 rpm. Far field boundaries are approximated by enclosing the rotor system in a cylinder having a radius of three times the blade length. This reduces reflections of waves back into the domain. For computational efficiency, only one blade is modeled, resulting in a half-cylinder domain. The initial mesh contains 255,203 tetrahedral elements with a piecewise linear finite element basis. The inflow and outflow boundary conditions of a hovering blade are applied as in [27].

Figure 2 shows the solution of this problem at three refinement levels. The problem was begun on 16 processors of the IBM SP2 computer at Rensselaer. As mesh sizes increased, the number of processors needed increased, and the computation was continued using the IBM SP computer at the Cornell Theory Center. Predictive load balancing was used throughout the computation. In the figure, the first picture (Level 0) is the solution following an initial refinement step, the second (Level 2) is the solution following two refinement steps, and the other (Level 4) is after four additional refinement steps have been applied. By Level 4, the mesh consists of 1,771,920 elements and the problem is run on 48 SP processors. The surface of the blade is shown in gray. The most important flow feature of a hovering rotor blade is the "tip vortex." A vortex tube is a cylinder where the velocity re-circulates around a "vortex core" [11]. Vortex tubes are shed from the tip of each blade and significantly influence the lift and drag characteristics of the blade. The main goal of adaptivity here is to capture tip vortices accurately, since they are small scale features and require finer mesh resolution than the remainder of the flow domain. Shaded areas near the blade tip (lower left in each plot) indicate the presence of tip vortices. Elements that a vortex tube might contain are identified and subsequently refined to enrich the solution [11]. Thus, vortex tubes serve as a refinement indicator to concentrate computation near the tip vortices.

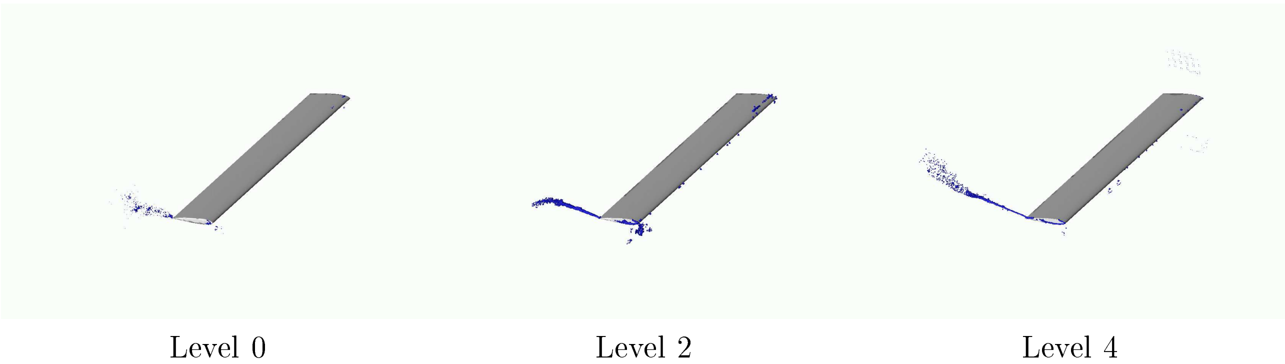


Figure 2: Tip vortex adaptation for the rotor blade. Level 0 – solution after an initial refinement. Level 2 – solution after two additional refinement steps. Level 4 – solution after four additional refinement steps (beyond Level 0). The rotor blade is shown in gray, and the darkened area trailing the blade tip indicates tip vortices.

A representative mesh from this problem is used to indicate some of the advantages of UP. This test is performed on 16 processors of Rensselaer's IBM SP2 computer. The initial mesh has 255,203 elements distributed across the processors, and a single refinement is attempted based on previously computed error indicator information. Here, predictive load balancing is performed using ITB. For this example, the terms load and number of elements are used interchangeably.

Table 1 compares the number of edges marked for refinement on each processor for the same mesh with and without predictive balancing. Let E_{min} and E_{max}, respectively, denote the minimum and maximum number of marked edges on any processor, and E_{avg} the average number of marked edges across all processors. The percent imbalance is then computed as $(E_{max} - E_{avg})/E_{avg} \times 100$, and the percent variation is computed as $(E_{max} - E_{min})/E_{avg} \times 100$. The difference in average edges marked between the two methods arises from the fact that boundary edges are duplicated within PMDB; thus, the number of marked boundary edges can change as data is migrated during predictive balancing. Table 1 shows that the maximum number of marked edges on any processor is reduced by 37% as a result of predictive balancing; the percent imbalance is reduced by 85%; and the percent variation is decreased by 80%. This illustrates that UP establishes a reasonable load balance for the refinement stage of enrichment for this steady problem.

Balance Type	Number of Marked Edges			Percent Imbalance	Percent Variation
	E_{min}	E_{max}	E_{avg}		
Nonpredictive	1622	21220	12074	75.91	162.47
Predictive	9416	13431	12063	11.24	33.25

Table 1: Comparison of edges marked for refinement for a mesh in the rotor analysis.

Figure 3 indicates that refinement is also balanced with respect to memory usage. Each processing node of the Rensselaer's IBM SP2 computer has 128 MB of memory, in which FAST can store approximately 85,000 elements.

At the beginning of the test, an equal number of elements reside on each processor before refinement. For the nonpredictive case, this is the distribution used during refinement (Figure 3, top). The expected load on each processor after refinement, as estimated using (4), is shown with dashed lines in the same graph. With upper storage limits of 85,000 elements per processor, Processors 3, 4, and 5 are expected to overflow memory during refinement. Indeed, in the actual run, processor 3 terminates with a memory allocation failure during refinement.

In the bottom of Figure 3, the pre-refinement load on each processor is unequal because elements have been moved to balance the estimated post-refinement load. By default, only ten iterations of ITB are performed by the predictive balancing. The total estimated load is 1,062,146 elements. For such a large change in load, more iterations of ITB would be required to reach balance, which explains the imbalance in estimated load. A global repartitioning (*e.g.* , OCT or PSIRB) would provide a better solution in this case. Nevertheless, the estimated load is balanced with only a 2.4% load imbalance, and the average estimated load and the average actual number of elements residing on each processor after refinement are within 0.6%. These factors lead to a load imbalance of 2.9% after refinement. Since FAST has a homogeneous workload per element, predictive balancing has eliminated the need for *a posteriori* balancing. Most importantly, because elements are moved predictively, none of the processors exhausts its memory during refinement, and computation may continue on the enriched mesh.

5.2 Transient Flow in a Perforated Tube

The effectiveness of weighted techniques in balancing the LRM, and the benefits of the VP method are demonstrated by considering the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent. This problem is motivated by flow studies in perforated muzzle brakes for large calibre guns [10]. Simulation flow parameters match those of shock tube studies of Dillon [10] and Nagamatsu *et al.* [17]. 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. The simulation begins with an initial mesh of 69,572 tetrahedral elements distributed across 16 processors of an IBM SP2 computer. The software used is a Parallel Adaptive Euler Solver in which three-dimensional conservation laws are discretized using a discontinuous Galerkin finite element method [3, 7, 8] with explicit integration. The software uses both spatial and temporal refinement to concentrate computational effort in areas of the problem domain where it is most needed [14]. Figure 4 shows the Mach number with velocity vectors at times $t = 0.3$ and $t = 0.6$ in the simulation. Flow features compare favorably with experimental and numerical results of Nagamatsu *et al.* [17].

Iterative balancing methods like ITB tend to have low per-iteration costs. However, global repartitioners generally maintain better partition quality than ITB [5]. Thus, the standard nonpredictive method executes a few iterations of ITB between spatial enrichment stages to maintain some balance, followed by a global repartitioning after enrichment to achieve better partition quality. In this section, OCT refers to a unit-weighted *a posteriori* repartitioning, and WOCT represents a size-weighted *a posteriori* balancing. No such global repartitioning is necessary after VP, since it is designed to balance the computation stage. However, a global partitioner is used with VP (VPOCT) in order to maintain good partition quality even with repeated application.

We use two important and accurate measures of computational imbalance to gauge the effectiveness of a technique in balancing the LRM [9, 14]. *Time-step imbalance* is defined as the maximum number of elements time stepped on a processor normalized by the average number stepped on all processors. Likewise, the *flux imbalance* is the maximum number of fluxes computed on a processor relative to the average number

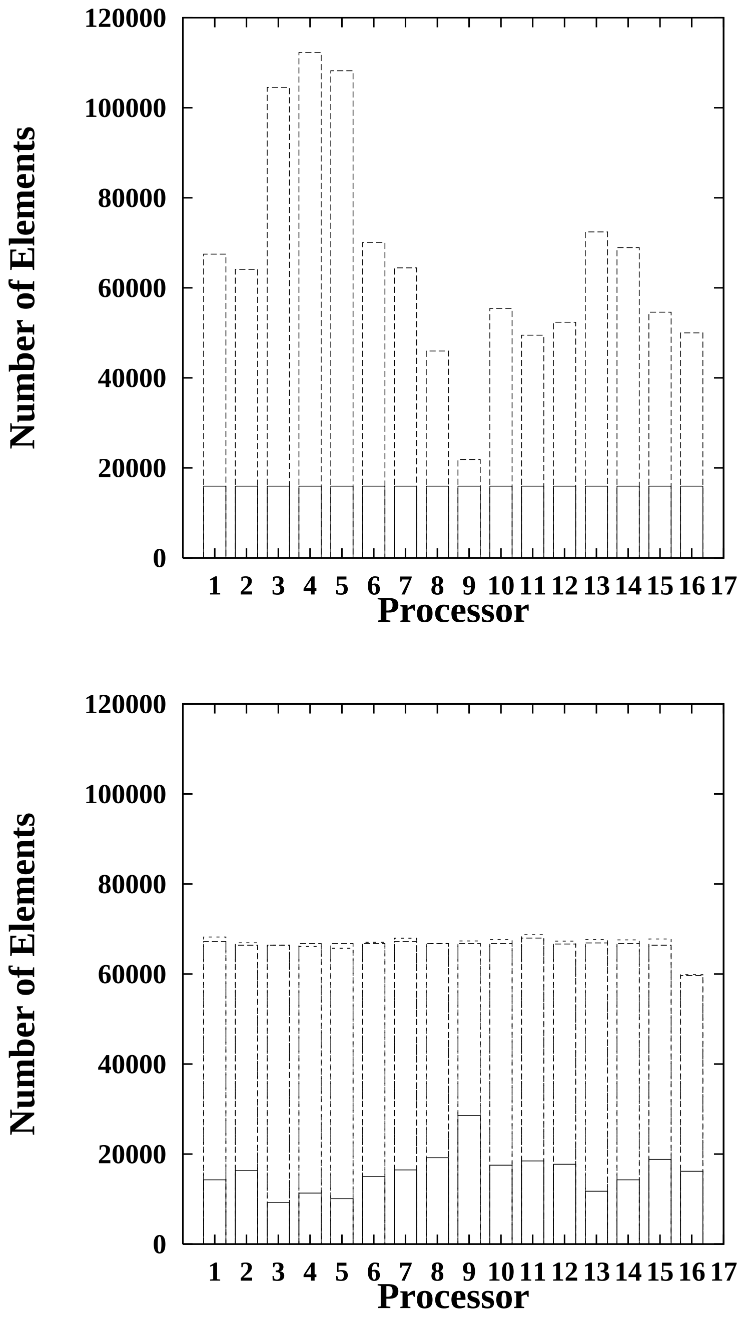


Figure 3: Top: Initial load (solid lines) and estimated post-refinement load (dashed lines) on each processor prior to predictive balancing. Bottom: Predictively balanced starting load (solid lines), estimated post-refinement load (dashed lines), and true post-refinement load (dotted lines) on each processor.

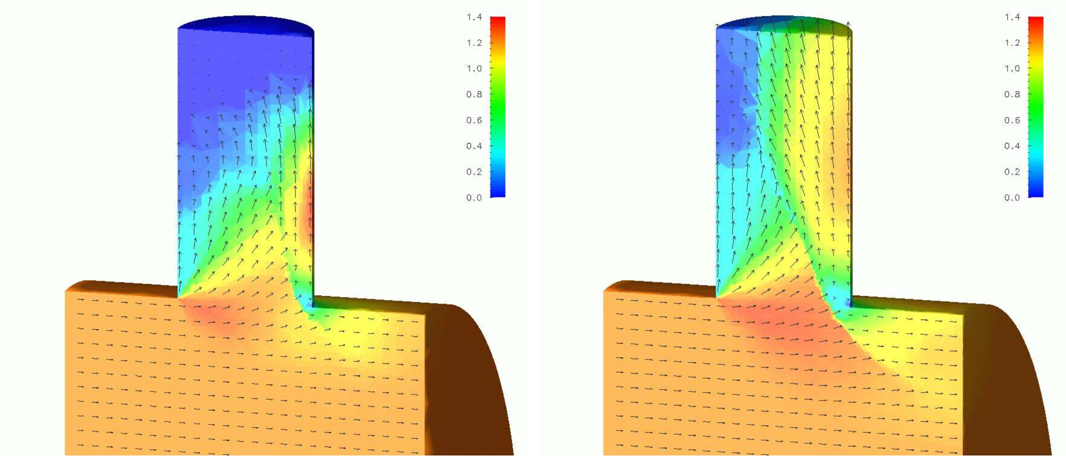


Figure 4: Projections of the Mach number and velocity vectors onto the surfaces of a perforated cylinder at time 0.3 (left) and time 0.6 (right).

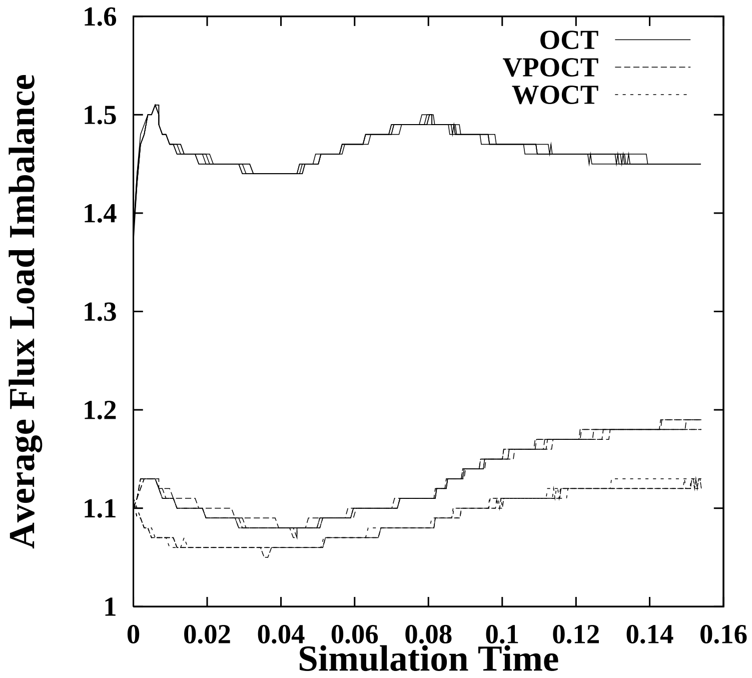
computed on all processors. In each case, let the average imbalance at simulation time t be a weighted average of all imbalances to time t. The weighting is the wall-clock duration of an imbalance relative to the total wall-clock time of the computation.

Average flux and time-step imbalances are shown in Figure 5 for a series of five runs using OCT, WOCT, and VPOCT. The graphs indicate that WOCT yields an average improvement over OCT of 19-29% for flux imbalance and of 20-29% for time-step imbalance. VPOCT also maintains good flux and time-step imbalances showing improvements of 18-26% and 20-27%, respectively, over OCT. VPOCT differs from WOCT by 1-6% in average flux imbalance and by 1-4% in average time-step imbalance. Thus, while WOCT maintains the best load balance of the three, VPOCT produces nearly the same level of computational efficiency. In addition, it has been shown to reduce data migration by 55-83% and combined enrichment and balancing times by 38-71% for this same problem [13].

6 Conclusions

One measure of load balance during refinement is the number of edges marked for refinement on each processor. For the steady rotor problem (Section 5.1), the UP method is able to reduce imbalance in marked edges by 80% relative to nonpredictive balancing for a representative mesh. Since refinement occurs in isolated regions, predictive balancing allows this problem to be solved without any processor exhausting available memory during refinement. Furthermore, the mesh produced in conjunction with UP only shows a 2.9% load imbalance with respect to the solution phase.

The comparison of OCT, WOCT, and VPOCT for the transient problem of Section 5.2 illustrates that WOCT produces the best load balance for LRM. It reduces load imbalances by 25% with respect to both average time-step and flux imbalances



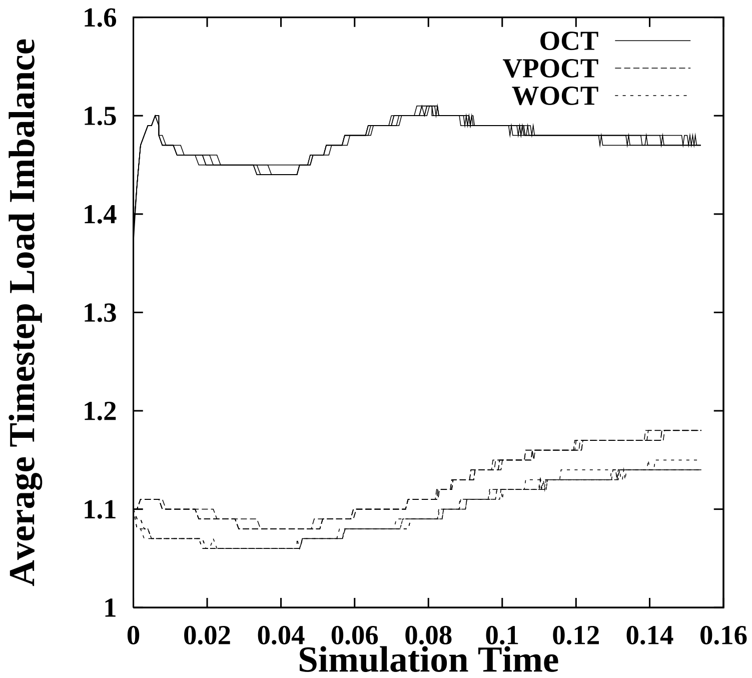


Figure 5: Average flux load imbalances (top) time-step load imbalances (bottom) for a sequence of five runs using OCT, VPOCT, and WOCT.

when compared to OCT. Overall, VPOCT reduces flux imbalance by 22% and time-step imbalance by 23% relative to OCT. It differs from WOCT's results by only 4.0% and 2.6%, respectively. When combined with previous results [13] indicating a 72% average improvement in data migrated and a 62% reduction in combined enrichment and balancing times relative to nonpredictive balancing with ITB during enrichment and WOCT after, VPOCT shows promise in balancing transient computations with LRMs.

In general, predictive balancing reduces the time spent in enrichment and allows a single load balancing invocation to replace several calls performed by nonpredictive balancing. However, as seen here, computation performed on the resulting partitions is generally no more efficient than that done on partitions generated by an *a posteriori* method. Hence, a problem which adapts the mesh infrequently will show less improvement in overall performance than one that requires frequent enrichments.

Adaptive p-refinement is being added to the system and will require extension of the predictive balancing to account for the number of degrees of freedom associated with each element.

We are implementing a general-purpose, object-oriented, parallel adaptive analysis framework. Parallel structures will be implemented at the lowest level of the framework to allow some operations to be done in a more natural and efficient way than with PMDB, which resides on top of the sequential MDB. The additional information which will be available in the analysis framework structures will allow more sophisticated load balancing procedures to be investigated.

7 Acknowledgements

We would like to thank our colleagues in Computer Science and in the Scientific Computation Research Center at Rensselaer for the 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 which was later replaced by a 160-node SP system at the Cornell Theory Center. Authors were supported by NSF grant DMS-9318184, ARO contract DAAH04-95-1-0091, and NSF grant CCR-9527151.

References

[1] S. T. Barnard. PMRSB: parallel multilevel recursive spectral bisection. In F. Baker and J. Wehmer, eds., *Proc. Supercomputing '95*, San Diego, 1995.

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

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

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

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

[6] F. Caradonna and C. Tung. Experimental and analytical studies of a model helicopter rotor in hover. Tech. Report 81232, NASA, 1980.

[7] 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.

[8] 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.

[9] 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, eds., *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, vol. 75, pp. 215–242, Springer-Verlag, Berlin-Heidelberg, 1995.

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

[11] M. Dindar, A. Lemnios, and M. Shephard. Adaptive solution procedures for rotorcraft aerodynamics. Submitted for publication, 1997.

[12] 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. To appear *Appl. Num. Math.*

[13] J. E. Flaherty, R. M. Loy, P. C. Scully, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Load balancing and communication optimization for parallel adaptive finite element computation. To appear, *Proc. SCCC '97*, 1997.

[14] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws. IMA Preprint Series 1483, Institute for Mathematics and its Applications, University of Minnesota, 1997. Submitted for publication.

[15] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Predictive load balancing for parallel adaptive finite element computation. In H. R. Arabnia, ed., *Proc. PDPTA '97*, vol. I, pp. 460–469, 1997.

[16] G. Karypis and V. Kumar. Metis: Unstructured graph partitioning and sparse matrix ordering system. Tech. Report, University of Minnesota, Department of Computer Science, Minneapolis, MN, 1995.

[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 Tech. Report ARCCB-TR-87016, Benet Weapons Laboratory, Watervliet, 1987.

[18] 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.

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

[20] 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.

[21] Y. Saad and M. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scien. Stat. Comput.*, 7:856–869, 1986.

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

[23] F. Shakib. *Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations.* PhD thesis, Dept. of Mechanical Engineering, Stanford University, CA, 1988.

[24] 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*, num. R-807, pp. 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.

[25] 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.

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

[27] G. Srinivasen, V. Raghavan, and E. Duque. Flowfield analysis of modern helicopter rotors in hover by Navier-Stokes method. In *Proc. Int. Specialist Meeting on Rotorcraft Acoustics and Rotor Fluid Dynamics*, Philadelphia, PA, 1991.

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

[29] 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.

[30] C. H. Walshaw, M. Cross, and M. Everett. Mesh partitioning and load-balancing for distributed memory parallel systems. In *Proc. Par. Dist. Comput. for Comput. Mech.*, Lochinver, Scotland, 1997.

Author contact information

J. E. Flaherty† (*flaherje@cs.rpi.edu*), R. M. Loy (*loyr@cs.rpi.edu*), M. Dindar (*mdindar@scorec.rpi.edu*), M. S. Shephard (*shephard@scorec.rpi.edu*), B. K. Szymanski (*szymansk@cs.rpi.edu*), J. D. Teresco (*terescoj@cs.rpi.edu*), and L. H. Ziantz (*ziantzl@cs.rpi.edu*). Computer Science Department and Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY, U.S.A.

† - primary author for correspondence