PARALLEL PARTITIONING STRATEGIES FOR THE ADAPTIVE SOLUTION OF CONSERVATION LAWS *

KAREN D. DEVINE†, JOSEPH E. FLAHERTY‡, RAYMOND M. LOY‡ AND STEPHEN R. WHEAT§

Abstract. We describe and examine the performance of adaptive methods for solving hyperbolic systems of conservation laws on massively parallel computers. The differential system is approximated by a discontinuous Galerkin finite element method with a hierarchical Legendre piecewise polynomial basis for the spatial discretization. Fluxes at element boundaries are computed by solving an approximate Riemann problem; a projection limiter is applied to keep the average solution monotone; time discretization is performed by Runge-Kutta integration; and a p-refinement-based error estimate is used as an enrichment indicator. Adaptive order (p-) and mesh (h-) refinement algorithms are presented and demonstrated. Using an element-based dynamic load balancing algorithm called tiling and adaptive p-refinement, parallel efficiencies of over 60% are achieved on a 1024-processor nCUBE/2 hypercube. We also demonstrate a fast, tree-based parallel partitioning strategy for three-dimensional octree-structured meshes. This method produces partition quality comparable to recursive spectral bisection at a greatly reduced cost.

Key words. Adaptive methods, hyperbolic systems of conservation laws, massively parallel computation, Galerkin finite element method, h-refinement, p-refinement, load balancing, tiling, domain decomposition, octree-derived meshes.

AMS(MOS) subject classifications. 65M20, 65M50, 65M60.

1. Introduction.

Adaptive finite difference and finite element methods, which automatically refine or coarsen meshes and vary the order of accuracy of the numerical solution, offer greater robustness and computational efficiency than traditional methods. High-order methods and the combination of mesh refinement and order variation (hp-refinement) have been shown to produce effective solution techniques for elliptic [7,28] and parabolic [2,3,10,26] problems. With few exceptions [11,16], work on

* This research was supported by the U.S. Army Research Office Contract Number DAAL03-91-G-0215 and DAALO3-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center (AHPCRC) and the DoD Shared Resource Center at the AHPCRC (Flaherty, Loy); Sandia National Laboratories, operated for the U.S. Department of Energy under contract #DE-AC04-76DP00789 (Devine, Wheat), and Research Agreement AD-9585 (Devine); a DARPA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland (Loy); and the Grumman Corporate Research Center, Grumman Corporation, Bethpage, NY 11714-3580 (Loy).

† Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

‡ Department of Computer Science and Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590; and Applied Mathematics and Mechanics Section, Benét Laboratories, Watervliet Arsenal, Watervliet, NY 12189.

§ Massively Parallel Computation Research Laboratory, Sandia National Laboratories, Albuquerque, NM 87185-1109.

The discontinuous Galerkin method is well suited to parallelization on massively parallel computers. The computational stencil involves only nearest-neighbor communication regardless of the degree of the piecewise polynomial approximation and the spatial dimension. Additional storage is needed for only one row of "ghost" elements along each edge of a processor's subdomain. Thus, the size of the problem scales easily with the number of processors. Indeed, for two-dimensional problems on rectangular domains with periodic boundary conditions, scaled parallel efficiencies in excess of 97% are achieved [11].

To achieve parallel efficiency with irregular structures, parallel finite element methods often use *static* load balancing [19,21] as a precursor to obtaining a finite element solution. Parallel efficiency degrades substantially due to processor load imbalances with adaptive enrichment. Even with the lower parallel efficiency, however, execution times for comparable accuracy are shorter with adaptive methods than for fixed-order methods.

We have developed an adaptive p-refinement method for two-dimensional systems that uses *dynamic* load balancing to adjust the processor decomposition in the presence of nonuniform and changing work loads. *Tiling* [37] is a modification of a dynamic load balancing technique developed by Leiss and Reddy [24] that balances work within overlapping processor neighborhoods to achieve a global load balance. Work is migrated from a processor to others within the same neighborhood. We demonstrate the improved performance obtained from a combination of p-adaptivity and parallel computation on several examples using a 1024-processor nCUBE/2 hypercube.

For three-dimensional problems with irregular grids of tetrahedral elements and adaptive h-refinement, we have developed a tree-based mesh partitioning technique that exploits the properties of tree-structured meshes. The rich, hierarchical structure of these meshes allows them to be divided into components along boundaries of the tree structure. Our partitioning technique is based on two tree traversals that (*i*) calculate the processing costs of all subtrees of a node, and (*ii*) form the partitions. Our method is inexpensive and, thus, has an advantage relative to other global partitioning techniques [21,23,27]. We demonstrate the performance of the tree-based mesh partitioning technique on a variety of three-dimensional meshes and discuss extension of the technique for parallel implementation and dynamic load balancing. We present results, using a Thinking Machines CM-5 computer, for the adaptive h-refinement solutions of an Euler flow past a cone.

2. The Discontinuous Galerkin Method. Partition the domain Ω into polygonal elements Ω_j, $j = 1, 2, \ldots, J$, and construct a weak form of the problem by multiplying (1.1a) by a test function $\mathbf{v} \in L^2(\Omega_j)$ and

In regions where the numerical solution is smooth, the discontinuous Galerkin method produces the $O(h^{p+1})$, $h = \max_{i=1,2,\ldots,d,\ j=1,2,\ldots,J}(\Delta x_{i,j})$, convergence expected in, e.g., L^1 for a p^{th}-degree approximation [11,14]. To prevent spurious oscillations that develop near discontinuities with high-order methods, we have developed a projection limiter that limits solution moments [11,14,36]. Using a one-dimensional ($d = 1$) scalar problem and the Legendre polynomial basis

$$U_j(\xi, t) = \sum_{l=0}^{p} c_{jl}(t) P_l(\xi) \tag{2.8}$$

as an illustration, the coefficient c_{jk} is proportional to the k^{th} moment $\mathcal{M}_{jk}$ of U_j; i.e.,

$$\mathcal{M}_{jk} := \int_{-1}^{1} U_j(\xi, t) P_k(\xi)\, d\xi = \frac{2}{2k+1} c_{jk}, \qquad k = 0, 1, \ldots, p-1, \quad j = 1, 2, \ldots, J. \tag{2.9}$$

Thus, to keep $\mathcal{M}_{jk}$ monotone, we must keep c_{jk} monotone on neighboring elements, which we do by specifying

$$(2k+1)c_{j,k+1} =$$

$$minmod((2k+1)c_{j,k+1}, c_{j+1,k} - c_{j,k}, c_{j,k} - c_{j-1,k}), \tag{2.10a}$$

where

$$minmod(a, b, c) =$$

$$\begin{cases} \text{sign}(a) \min(|a|, |b|, |c|), & \text{if} \quad \text{sign}(a) = \text{sign}(b) = \text{sign}(c) \\ 0, & \text{otherwise.} \end{cases} \tag{2.10b}$$

The limiter (2.10) is applied adaptively. First, the highest-order coefficient c_{jp} is limited. Then the limiter is applied to successively lower-order coefficients whenever the next higher coefficient on the interval has been changed by the limiting. The higher-order coefficients are re-limited using the updated low-order coefficients when necessary. In this way, the limiting is applied only where it is needed, and accuracy is retained in smooth regions. For two- and three-dimensional problems, the one-dimensional limiter is applied in the direction $\mathbf{n}$ normal to $\partial\Omega_j$.

For vector systems, the scalar limiting is applied to the characteristic fields of the system [13]. The diagonalizing matrices $\mathbf{T}(\mathbf{u})$ and $\mathbf{T}^{-1}(\mathbf{u})$ (consisting of the right and left eigenvectors of the Jacobian $\mathbf{f_{n_u}}$) are evaluated using the average values of $\mathbf{U}_j$, $j = 1, 2, \ldots, J$, on Ω_j; the scalar limiting is applied to each field of the characteristic vector; and the result is transformed back to physical space by post-multiplication by $\mathbf{T}^{-1}(\mathbf{U}_j)$.

as t increases. Then, for a two-dimensional approximation using a basis of tensor products of Legendre polynomials on rectangular elements,

$$\text{(3.4c)}\quad \mathbf{K}_j(\xi,\eta,t) = \begin{cases} \mathbf{c}_{j11}(t)P_1(\xi)P_1(\eta) \\ \quad +\mathbf{c}_{j10}(t)P_1(\xi)P_0(\eta) \\ \quad +\mathbf{c}_{j01}(t)P_0(\xi)P_1(\eta), & \text{if } p=0 \\ \\ \mathbf{c}_{j,p+1,p+1}(t)R_{p+1}(\xi)R_{p+1}(\eta) \\ \quad +\sum_{k=0}^{p}(\mathbf{c}_{jk,p+1}(t)P_k(\xi)R_{p+1}(\eta) \\ \qquad +\mathbf{c}_{j,p+1,k}(t)R_{p+1}(\xi)P_k(\eta)), & \text{if } p>0. \end{cases}$$

To compute $\mathbf{K}_j(\mathbf{x},t)$, let $\hat{\mathbf{U}}_j = \mathbf{U}_j + \mathbf{K}_j$, $j = 1,2,\ldots,J$, substitute $\hat{\mathbf{U}}_j$ into (2.3), and solve for the coefficients of $\mathbf{K}_j(\mathbf{x},t)$ with $\mathbf{U}_j$ fixed.

To compute E_j using (3.4), we solve $2p+3$ additional ordinary differential equations in two dimensions, compared to an additional $(p+2)^2$ differential equations required for (3.3). The movement of the superconvergence points from the Legendre points at $t=0$ toward the Radau points for $t>0$ is gradual, occurring over several time steps [11]. Thus, the effectiveness of the estimate improves as the computation progresses.

After each time step, we compute E_j, $j=1,2,\ldots,J$, and increase the polynomial degree of $\mathbf{U}_j$ by one if $E_j > \text{TOL}$. The solution $\mathbf{U}_j$ and the error estimate are recomputed on enriched elements, and further increases of degree occur until $E_j \le \text{TOL}$ on all elements.

We reduce the need for back-tracking by predicting the degree of the approximation needed to satisfy the accuracy requirements during the next time step. After a time step is accepted, if $E_j > H_{max}\text{TOL}$, $H_{max} \in (0,1]$, we increase the degree of $\mathbf{U}_j(\mathbf{x},t+\Delta t)$ for the next time step. If $E_j < H_{min}\text{TOL}$, $H_{min} \in [0,1)$, we decrease the degree of $\mathbf{U}_j(\mathbf{x},t+\Delta t)$ for the next time step.

EXAMPLE 1. We demonstrate the accuracy of the error estimate (3.4) in terms of its effectivity index

$$\text{(3.5)}\qquad \Theta = \frac{\text{Estimated Error}}{\text{Actual Error}}$$

for the two-dimensional problem

$$\text{(3.6a)}\qquad u_t + u_x + u_y = 0, \quad -1 < x,y < 1, \quad t > 0,$$

with

$$\text{(3.6b)}\qquad u^0(x,y) = \sin(\pi x)\sin(\pi y), \quad -1 \le x,y \le 1,$$

and periodic boundary conditions. In Table 3.1, we show the actual errors and effectivity indices with $p = 0, 1,$ and 2. Each time the mesh is refined,

the time step is halved, and the number of time steps is doubled. Effectivity indices are near unity for the entire range of computation when $p = 0$. For $p = 1$ and 2, the error estimate improves as the mesh is refined since the superconvergence points move closer to the Radau points after each time step.

	Number of Elements	Actual Error	Θ
$p = 0$	16×16	$2.66838e-1$	0.967
	32×32	$1.33946e-1$	0.969
	64×64	$6.70306e-2$	0.973
	128×128	$3.35206e-2$	0.976
	256×256	$1.67605e-2$	0.978
$p = 1$	16×16	$1.45948e-2$	0.540
	32×32	$4.21090e-3$	0.805
	64×64	$1.11300e-3$	0.975
	128×128	$2.79793e-4$	1.000
	256×256	$6.99557e-5$	1.000
$p = 2$	16×16	$6.41413e-4$	0.557
	32×32	$9.68358e-5$	0.646
	64×64	$9.68224e-6$	1.128
	128×128	$1.26721e-6$	1.009
	256×256	$1.58712e-7$	1.000
	512×512	$1.98384e-8$	1.000

TABLE 3.1
Errors and effectivity indices Θ at $t = 0.025$ using (3.4) for Example 1.

EXAMPLE 2. Consider

$$u_t + 2u_x + 2u_y = 0, \quad 0 < x, y < 1, \quad t > 0, \tag{3.7a}$$

with initial and Dirichlet boundary conditions specified so that the exact solution is

$$u(x, y, t) = \frac{1}{2}(1 - \tanh(20x - 10y - 20t + 5)), \quad 0 \le x, y \le 1. \tag{3.7b}$$

In Figure 3.1, we show the exact solution of (3.7) at time $t = 0$ and the degrees generated on a adaptive 16×16-element mesh to satisfy the initial data when $\mathrm{TOL} = 10^{-5}$.

We solve (3.7) by both fixed-order and adaptive p-refinement methods on $0 < t \le 0.1$. In Figure 3.2, we show the global L^1-error versus the CPU time for fixed-order methods with $p = 0$, 1, and 2 on 8×8, 16×16, 32×32, and 64×64-element meshes, and the p-adaptive method with $H_{max} = 0.9$, $H_{min} = 0.1$, and TOL ranging from 5×10^{-9} to 5×10^{-4} on

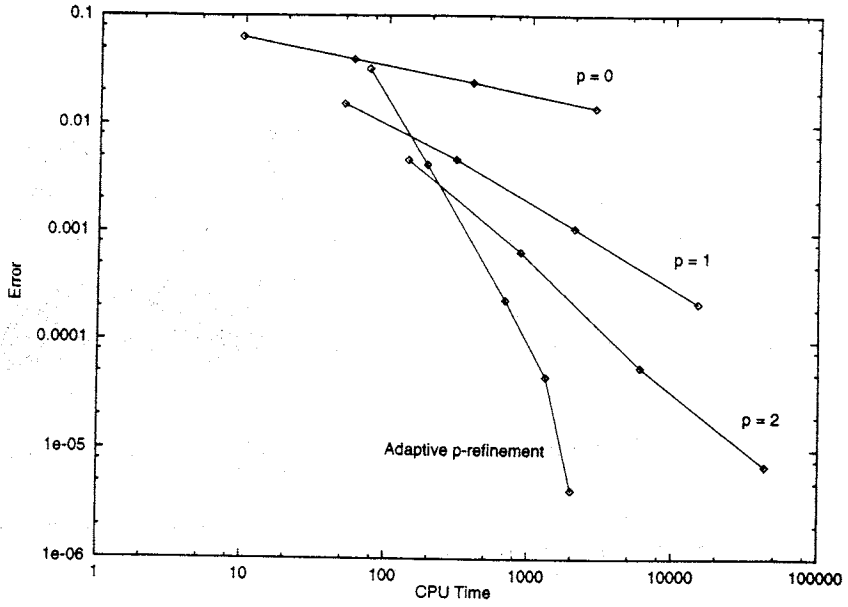

FIG. 3.2. *Convergence of the adaptive p-refinement method and fixed-order methods with* $p = 0, 1,$ and 2 *for Example 2.*

inter-processor boundary data, and processes the boundary data. A balancing phase restores load balance following a given number of computation phases. Each balancing phase consists of the following operations:

1. **Determine work loads.** Each processor determines its work load as the time to process its local data since the previous balancing phase less the time to exchange inter-processor boundary data during the computation phase. Neighborhood average work loads are also calculated.
2. **Determine processor work requests.** Each processor compares its work load to the work load of the other processors in its neighborhood and determines those processors having loads greater than its own. If any are found, it selects the one with the greatest work load (ties are broken arbitrarily) and sends a request for work to that processor. Each processor may send only one work request, but a single processor may receive several work requests.
3. **Select elements to satisfy work requests.** Each processor prioritizes the work requests it receives based on the request size, and determines which elements to export to the requesting processor. Elemental processing costs are used so that the minimum number of elements satisfying the work request are exported. (This approach differs from Wheat [37], where the average cost per element is used to determine the number of export elements). Details of the selection algorithm follow.
4. **Notify and transfer elements.** Once elements to be exported

and one neighbor in some other processor (-2). Elements 6 and 9 share the highest priority, but element 6 is selected because it has a greater work load. Element 5 becomes eligible for export, but its priority is low since it has three local neighbors. The priorities are adjusted, and element 9 is selected, making element 8 a candidate. The priorities are again updated, and the selection process continues with elements 3 and 12 being selected. Although the work request is not completely satisfied, no other elements are exported, since the work loads of the elements with the highest priority, 5 and 8, are greater than the remaining work request.

EXAMPLE 3. We solve (3.7) with a fixed-order method ($p = 3$) on a 32×32-element mesh and tiling on 16 processors of the nCUBE/2 hypercube. In Figure 4.3 (left), we show the processor domain decomposition after 20 time steps. The tiling algorithm redistributes the work so that processors containing elements on the domain boundary have fewer elements than those in the interior of the domain. The global error of the numerical solution is 4.76766×10^{-3}. The total processing time was reduced by 5.18% from 128.86 seconds to 122.18 seconds by balancing once each time step. The average/maximum processor work ratio without balancing is 0.858; with balancing, it is 0.942. Parallel efficiency is increased from 90.80% without balancing to 95.58% with tiling.

We then solve (3.7) using the adaptive p-refinement method on a 32×32-element mesh with TOL $= 3.5 \times 10^{-5}$ and tiling on 16 processors. In Figure 4.3 (right), we show the processor domain decomposition after 20 time steps. The shaded elements have higher-degree approximations and, thus, higher work loads. The tiling algorithm redistributes the work so that processors with high-order elements have fewer elements than those processors with low-order elements. The global error of the adaptive solution is 4.44426×10^{-3}, less than the fixed-order method above. The total processing time for the adaptive method was reduced 41.98% from 63.94 seconds to 37.10 seconds by balancing once each time step. The average/maximum processor work ratio without balancing is 0.362, and with balancing, it is 0.695. Parallel efficiency is increased from 35.10% without balancing to 60.51% with tiling.

EXAMPLE 4. We solve (3.7) for 225 time steps on all 1024 processors of the nCUBE/2 without balancing and with balancing once each time step. A fixed-order method with $p = 2$ produces a solution with global error 6.40644×10^{-2}. Using the tiling algorithm reduced the total execution time 6.25% from 1601.96 seconds without balancing to 1501.90 seconds with balancing (see Table 4.1). Parallel efficiency without balancing was 82.7%; with balancing, it was 88.2%.

The adaptive p-refinement method produced a solution with global error 6.32205×10^{-2}, comparable to the fixed-order solution. With balancing, the maximum computation time (not including communication or balancing time) was reduced by 49.8% (see Table 4.1). The irregular subdomain boundaries created by the tiling algorithm increased the average

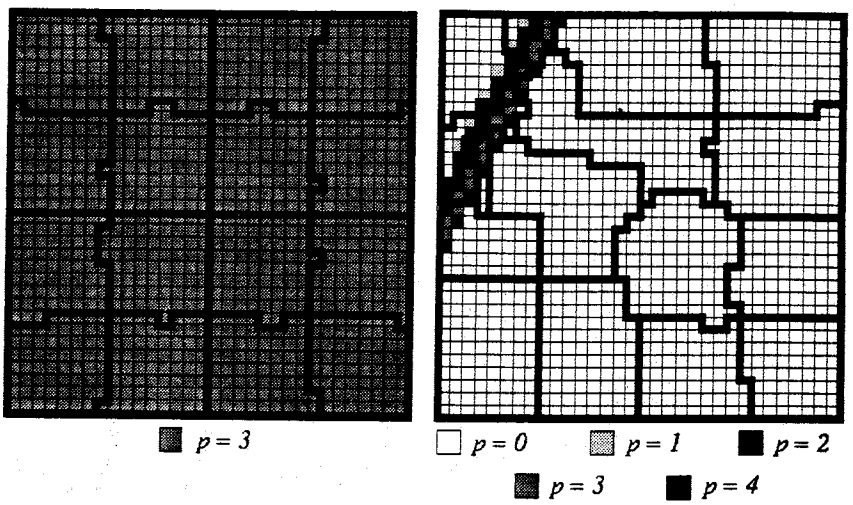

FIG. 4.3. *Processor domain decompositions after 20 time steps for Example 3 using fixed-order (left) and adaptive order (right) methods. Dark lines represent processor subdomain boundaries.*

communication time by 2.5%. Despite the extra communication time and the load balancing time, however, we see a 36.3% improvement in the total execution time.

In Figure 4.4, we show the maximum processing costs per time step, including the computation and balancing times, for the adaptive p-refinement method. The dashed and solid lines represent the maximum cost without and with balancing, respectively. The balanced computation's maximum cost per time step is significantly lower than without balancing. The spikes in both curves occur when the error tolerance was not satisfied on some elements and the adaptive p-refinement method back-tracked to compute a more accurate solution. In Figure 4.5, we show the cumulative maximum processing times with and without balancing. The immediate and sustained improvement of the application's performance is shown.

	Fixed-Order (p=2) Global Error: 0.06406		Adaptive p-refinement Global Error: 0.06322	
	Without Tiling	With Tiling	Without Tiling	With Tiling
Total Execution Time (seconds)	1601.96	1501.90	858.50	546.75
Max. Computation Time (seconds)	1549.77	1429.24	782.93	393.32
Average/Maximum Work Ratio	0.855	0.927	0.427	0.851
Avg. Communication Time (seconds)	59.09	59.09	70.85	72.65
Max. Balancing Time (seconds)	0.0	20.88	0.0	23.46
Parallel Efficiency	82.7%	88.2%	38.98%	61.21%

TABLE 4.1

Performance comparison for Example 4 using fixed-order and adaptive methods without and with balancing at each time step.

5. Three-Dimensional Mesh Partitioning. We describe a tree-based partitioning technique which utilizes the hierarchical structure of octree-derived unstructured meshes to distribute elemental data across processors' memories while reducing the amount of data that must be exchanged between processors. An octree-based mesh generator [30] recursively subdivides an embedding of the problem domain in a cubic universe into eight octants wherever more resolution is required. Octant bisection is initially based on geometric features of the domain but solution-based criteria are introduced during an adaptive h-refinement process. Finite element meshes of tetrahedral elements are generated from the octree by subdividing terminal octants.

In Figure 5.1, we illustrate the tree and mesh for a two-dimensional flow domain containing a small object. The root of the tree represents the entire domain (Figure 5.1c). The domain is recursively quartered until an adequate resolution of the object is obtained (Figure 5.1a). A smooth gradation is maintained by enforcing a one-level maximum difference between adjacent quadrants. After appropriate resolution is obtained, leaf quadrants are subdivided into triangular elements that are pointed to by leaf nodes of the tree (Figures 5.1b,c). The leaf quadrant containing the object must be decomposed into triangles based on the geometry of the object boundary. Smoothing, which normally follows element creation, is not shown.

Our tree-based based partitioning algorithm creates a one-dimensional ordering of the octree and divides it into nearly equal-sized segments based

subtrees are accumulated into successive partitions. The subtree rooted at the visited node is added to the current partition if it fits. If it would exceed the optimal size of the current partition, a decision must be made as to whether it should be added, or whether the traversal should examine it further. In the latter case, the traversal continues with the offspring of the node and the subtree may be divided among two or more partitions. The decision on whether to add the subtree or examine it further is based on the amount by which the optimal partition size is exceeded. A small excess may not justify an extensive search and may be used to balance some other partition which is slightly undersized. When the excess at a node is too large to justify inclusion in the current partition, and the node is either terminal or sufficiently deep in the tree, the partition is closed and subsequent nodes are added to the next partition.

This partitioning method requires storage for nonterminal nodes of the tree which would normally not be necessary since they contain no solution data. However, only minimal storage costs are incurred since information is only required for tree connectivity and the cost metric. For this modest investment, we have a partitioning algorithm that only requires $O(J)$ serial steps.

Partitions formed by this procedure do not necessarily form a single connected component; however, the octree decomposition and the orderly tree traversal tend to group neighboring subtrees together. Furthermore, a single connected component is added to the partition whenever a subtree fits within the partition.

A tree-partitioning example is illustrated in Figure 5.2. All subtree costs are determined by a post order traversal of the tree. The partition creation traversal starts at the root, Node 0 (Figure 5.2a). The node currently under investigation is identified by a double circle. The cost of the root exceeds the optimal partition cost, so the traversal descends to Node 1 (Figure 5.2b). As shown, the cost of the subtree rooted at node 1 is smaller than the optimal partition size and, hence, this subtree is added to the current partition, p0, and the traversal continues at Node 2 (Figure 5.2c). The cost of the subtree rooted at Node 2 is too large to add to p0, so the algorithm descends to an offspring of Node 2 (Figure 5.2d). Assuming Node 4 fits in p0, the traversal continues with the next offspring of Node 2 (Figure 5.2e). Node 5 is a terminal node whose cost is larger than the available space in p0, so the decision is made to close p0 and begin a new partition, p1. As shown (Figure 5.2f), Node 5 is very expensive, and when the traversal is continued at Node 3, p1 must be closed and work continues with partition p2.

The tree-traversal partitioning algorithm may easily be extended for use with a parallel adaptive environment. An initial partitioning is made using the serial algorithm described above. As the numerical solution advances in time, h- and/or p-refinement introduces a load imbalance. To obtain a new partitioning, let each processor compute its subtree costs us-

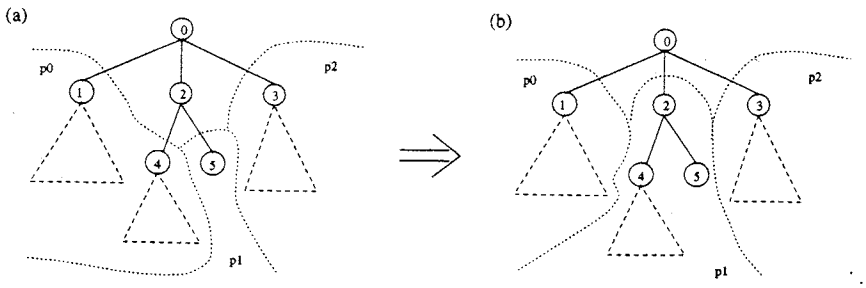

FIG. 5.3. *Iterative rebalancing of tree-based partitions. The subtree rooted at Node 4 (a) has been shifted from p0 to p1 (b) to relieve a load imbalance. The new root of p1 is Node 2, the common parent of Nodes 4 and 5.*

data movement is likely to be high and it would be desirable to amortize this by tolerating small imbalances. A strategy to delay the need for complete repartitioning would simply shift partition boundaries, thus, migrating subtrees from a processor P_n to its neighbors P_{n-1} and P_{n+1}. If, for example, processor P_n seeks to transfer cost m to P_{n-1}, it simply traverses its subtrees accumulating their costs until it reaches m. The nodes visited comprise a subtree which may be transferred to P_{n-1} and which is contiguous with the subtrees in P_{n-1}. Likewise, if P_n desires to transfer work to P_{n+1}, the reverse traversal could remove a subtree from the trailing part of P_n. Consider, as an example, the subtree rooted at Node 4 of Figure 5.3a and suppose that its cost has increased through refinement. In Figure 5.3b, we show how the partition boundary may be shifted to move the subtree rooted at Node 4 to partition p1. The amount of data to be moved from processor to processor may utilize a relaxation algorithm or the tiling procedure discussed in Section 4.

EXAMPLE 5. Performance results obtained by applying the tree-based mesh partitioning algorithm to various three-dimensional irregular meshes are presented in Figure 5.4. The meshes were generated by the Finite Octree mesh generator [30]. "Airplane" is a 182K-element mesh of the volume surrounding a simple airplane [17]. "Copter" is a 242K-element mesh of the body of a helicopter [17]. "Onera," "Onera2," and "Onera3" are 16K-, 70K-, and 293K-element meshes, respectively, of the space surrounding a swept, untwisted Onera-M6 wing which has been refined to resolve a bow shock [18]. "Cone" is a 139K-element mesh of the space around a cone having a 10° half-angle and which also has been refined to resolve a shock.

The quality of a partition has been measured as the percent of element faces lying on inter-partition boundaries relative to the total number of faces of the mesh. Graphs in Figure 5.4 display these percentages as a function of the optimal partition size. In all cases the cost variance between the partitions is very small (about as small as the maximum cost of a leaf octant). The proportion in Figure 5.4 is, in a sense, the total surface area

that partitions hold in common. Smaller ratios require less communication relative to the amount of local data access. This measure is closely related to the number of "cuts" that the partition creates [23,20,32]; however, we have chosen to normalize by the total number of faces in order to compare partition quality over a wide range of mesh sizes and number of partitions.

In large scale (top) the data of Figure 5.4 show the expected behaviour that the interface proportion approaches zero as the partition size increases (due to the number of partitions approaching unity). Conversely, as the optimal partition size approaches unity (due to number of partitions approaching the number of elements), the interface proportion goes to unity. Examination of the small scale (bottom) results reveals that the interface proportion is less than 12% when the partition size exceeds 1000 for these meshes. Interfaces drop to below 9% and 8%, respectively, for partition sizes of 2000 and 3000. This performance is comparable to recursive spectral bisection [22] but requires much less computation ($O(J)$ as opposed to $O(J^2)$ [27]).

The best performance occurred with the helicopter mesh, which was the only mesh of a solid object (as opposed to a flow field surrounding an object). The solid can easily be cut along its major axis to produce partitions with small inter-partition boundaries, and was included for generality. The lowest performance occurred with the cone mesh. This is most likely due to the model and shock region being conically shaped, which is somewhat at odds with the rectangular decomposition imposed by the octree.

In general, inter-partition boundaries should be less than 10%, indicating partition sizes of 2000 or more. This minimum partition size is not an excessive constraint, since a typical three-dimensional problem employing a two million-element mesh being solved on a 1024-processor computer would have about 2000 elements per processing element.

Another measure of partition quality is the percent of a partition's element faces lying on inter-partition boundaries relative to the total number of faces in that partition. This number is, in a sense, the ratio of surface area to volume of a partition. For our example meshes, this measure was below 22% and 18%, respectively, for partition sizes of 1000 and 1500.

EXAMPLE 6. In Figure 5.5 we show partitions of several meshes from Example 5. The partitions exhibit a blocked structure; however, several partitions of the airplane mesh appear to be made up of disconnected components. While this is possible, although unlikely, in this case the partitions appear to be disconnected because the display is a two-dimensional slice through the three-dimensional domain.

EXAMPLE 7. In Figure 5.6 we show the pressure contours of a Mach 2 Euler flow (1.1,2.6) past the "Cone" mesh of Example 5. The solution employs van Leer's flux vector splitting (2.7) and was computed on a Thinking Machines CM-5 with 128 processors. Several iterations of h-refinement were required to yield this mesh. At each iteration, elements were marked with

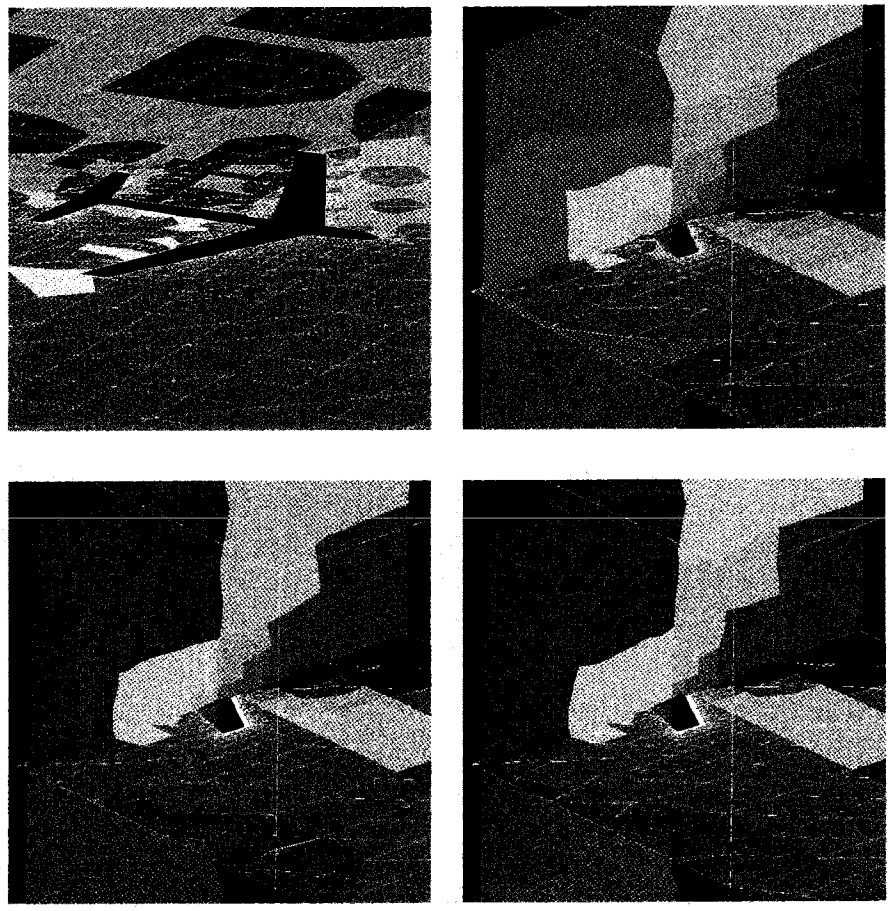

FIG. 5.5. *The airplane mesh, and three refinements of the Onera M6 wing mesh, all divided into 32 partitions. Each color represents a different partition.*

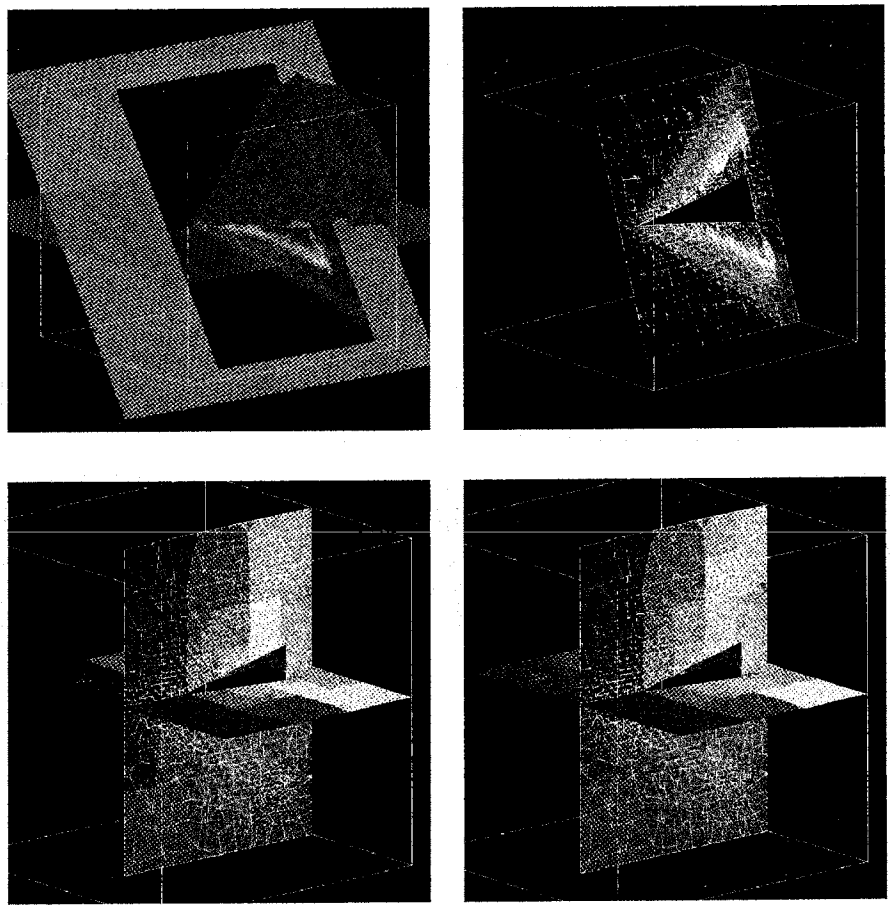

FIG. 5.6. *Shock surface and pressure contours found when computing the Mach 2 flow past a cone having a half-angle of* 10° *(top). Partitions of the mesh into 16 (left) and 32 (right) pieces (bottom). Each color represents a different partition.*

the desired tree level (either larger for refinement, or smaller for coarsening), and a new global mesh created to satisfy these constraints. The shock surface and pressure contours are shown above; below are examples of how the mesh may be partitioned for 16 and 32 processor machines. Each color represents membership in a different partition (and, hence, residence on a different processor).

6. Conclusion. We have demonstrated the effectiveness of adaptive methods for solving systems of hyperbolic conservation laws on massively parallel computers. Using a discontinuous Galerkin finite element method with projection limiting of moments of the solution within an element, we can model problems with discontinuities sharply without spurious oscillations. The discontinuous Galerkin method has a small computational stencil, enabling its efficient implementation on massively parallel computers. Adaptive p- and h-refinement methods provide faster convergence than traditional methods, but their nonuniform work loads create load imbalance on parallel computers, reducing the parallel efficiency of the methods. We correct the load imbalance by using a dynamic load balancing technique called tiling that produces a global balance by performing local balancing within overlapping neighborhoods of processors. Using tiling and adaptive p-refinement, computation of a two-dimensional example required approximately one-third as much time as a fixed-order computation with the same global accuracy. In three dimensions, we have demonstrated the effectiveness of a tree-based mesh partitioning algorithm for reducing parallel communication costs. This algorithm performs almost as well as recursive spectral bisection, but requires much less work to compute a partitioning.

In future work, we will combine the adaptive h- and p-refinement techniques to obtain an adaptive hp-refinement method that can optimize computational effort in both smooth and discontinuous solution regions. We will extend the tiling algorithm to incorporate the changing data structures required for h-refinement, and experiment with load balancing strategies for adaptive hp-refinement meshes. The tree-based partitioning algorithm will be extended to operate in parallel, and we will experiment with dynamic rebalancing strategies.

7. Acknowledgements. We wish to thank Thinking Machines Corporation, and in particular Zdeněk Johan and Kapil Mathur, for their assistance with the CM-5.

REFERENCES

[1] S. Adjerid, M. Aiffa, and J. E. Flaherty, *Adaptive Finite Element Methods for Singularly Perturbed Elliptic and Parabolic Systems*, submitted for publication, 1993.

[2] S. Adjerid and J. E. Flaherty, *Second-Order Finite Element Approximations and A Posteriori Error Estimation for Two-Dimensional Parabolic Systems*, Numer. Math., Vol. 53, 1988, pp. 183–198.

and Octree Grids, Computers and Structures, Vol. 30, 1988, pp. 327–336.

[26] P. K. MOORE AND J. E. FLAHERTY, *A Local Refinement Finite-Element Method for One-Dimensional Parabolic Systems*, SIAM J. Numer. Anal., Vol. 27, 1990, pp. 1422–1444.

[27] A. POTHEN, H. SIMON, AND K.-P. LIOU, *Partitioning Sparse Matrices with Eigenvectors of Graphs*, SIAM J. Matrix Analysis and Applications, Vol. 11, 1990, pp. 430–452.

[28] E. RANK AND I. BABUSKA, *An Expert System for the Optimal Mesh Design in the hp-Version of the Finite Element Method*, Int. J. Numer. Meths. Engng., Vol. 24, 1987, pp. 2087–2106.

[29] H. N. REDDY, *On Load Balancing*, Ph.D. Dissertation, Dept. Comp. Sci., Univ. of Houston, Houston, TX, 1989.

[30] M. S. SHEPHARD AND M. K. GEORGES, *Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique*, Int. J. Numer. Meths. Engng., Vol. 32, No. 4, 1991, pp. 709-749.

[31] C.-W. SHU AND S. OSHER, *Efficient Implementation of Essentially Non-Oscillatory Shock-Capturing Schemes, II*, J. of Comput. Phys., Vol. 27, 1978, pp. 1–31.

[32] H. D. SIMON, *Partitioning of Unstructured Problems for Parallel Processing*, Comput. Systs. Engng., Vol. 2, 1991, pp. 135–148.

[33] P. K. SWEBY, *High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws*, SIAM J. Numer. Anal., Vol. 21, 1984, pp. 995–1011.

[34] B. SZABO AND I. BABUSKA, *Introduction to Finite Element Analysis*, J. Wiley and Sons, New York, 1990.

[35] B. VAN LEER, *Flux Vector Splitting for the Euler Equations*, ICASE Report. No. 82-30, Inst. Comp. Applics. Sci. Engng., NASA Langley Research Center, Hampton, 1982.

[36] B. VAN LEER, *Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection*, J. Comput. Phys., Vol. 23, 1977, pp. 276–299.

[37] S. R. WHEAT, *A Fine Grained Data Migration Approach to Application Load Balancing on MP MIMD Machines*, Ph.D. Dissertation, Dept. Comp. Sci., Univ. of New Mexico, Albuquerque, 1992.

[38] S. R. WHEAT, K. D. DEVINE, AND A. B. MACCABE, *Experience with Automatic, Dynamic Load Balancing and Adaptive Finite Element Computation*, Proc. Hawaii Int. Conf. System Sciences, 1994, to appear.