JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, November 2005

# A Parallel Adaptation Package for Three-Dimensional Mixed-Element Unstructured Meshes

Peter A. Cavallo*
*Combustion Research and Flow Technology, Inc. (CRAFT Tech)*
*6210 Keller's Church Road, Pipersville, PA 18947*
*cavallo@craft-tech.com*
*Phone: 215-766-1520/Fax: 215-766-1524*

and
Matthew J. Grismer†
*Air Force Research Laboratory*
*Wright-Patterson AFB, OH 45433*
*matthew.grismer@wpafb.af.mil*

**Parallel mesh adaptation methods are developed to treat decomposed, mixed-element unstructured meshes comprised of any combination of basic element types. Emphasis is placed on developing conforming mesh modification methods that are solver-independent. Specific developments include the implementation of a treatment for viscous, high aspect ratio near wall tetrahedra, and cell subdivision methods for pyramids, prisms, and hexahedra. Rebalancing of the adapted grid, and particularly issues associated with processor assignment of parent/child cell sets, is addressed. The new methods are demonstrated on decomposed, mixed-element meshes employing up to 64 processors. The resulting parallel adaptation package is a powerful, versatile tool for obtaining grid-converged, steady state results, and may readily be applied to other unstructured flow solvers.**

## I. Introduction

THREE-DIMENSIONAL numerical simulations on parallel computing platforms have become commonplace in recent years. A number of structured and unstructured grid solvers for fluid dynamics and finite element analysis have been extended to operate in parallel environments. Not surprisingly, parallel adaptive mesh refinement methods have also received a fair amount of attention.[1–8] For fixed boundary applications, these include treatments for embedded quadrilateral[1] and Cartesian meshes,[2] $r$-refinement of block structured grids,[3] cell subdivision of unstructured triangular[4] and tetrahedral meshes,[5,6] octree-based tetrahedral grids,[7] and unstructured hexahedral grids.[8] Previous work by the first author[9,10] demonstrated parallel mesh coarsening and refinement methods for tetrahedral meshes, with a focus on moving boundary applications with deforming cells. Parallel adaptation of a computational mesh offers several improvements over traditional, serial mesh refinement schemes. These advantages include a reduction in memory requirements for large, multi-million cell models, efficiency gains from the use of multiple processors, and a closer coupling with the flow solution process, removing the need for separate preprocessing and repartitioning steps.

Received 21 February 2005; revision received and accepted for publication 17 August 2005. 

* Senior Research Scientist, AIAA Senior Member.
† Senior Research Aerospace Engineer, AIAA Senior Member.

Accurate resolution of viscous phenomena such as boundary layers and shear layers requires the use of high aspect ratio cells. While it is possible to compute such flows using anisotropic tetrahedral meshes, mixed-element meshes are more advantageous from the standpoint of efficiency.[11] A single hexahedron or prism can occupy the same volume as several tetrahedra with fewer edges and faces, resulting in fewer flux calculations. With the increased flexibility in geometric modeling and resolution offered by mixed-element meshes, adaptation methods for such meshes are of interest.

Mixed-element unstructured mesh adaptation has been investigated by several researchers.[12–15] Parthasarathy and Kallinderis[12] explored subdivision and redistribution methods for tet/prism grids. The prism subdivision was restricted to vertical refinements propagating from the tet/prism interface down to the wall, producing additional prisms. The first author also considered such an approach in previous work.[13] Biswas and Strawn[14] devised a conforming mesh adaptation method for unstructured hexahedral grids, in which child cells of tetrahedra, pyramids, or prisms were introduced to close hanging nodes between levels of refinement. Mavriplis[15] implemented a variety of cell subdivision techniques for each element type in a conforming approach for fully mixed-element meshes. Each of these prior research efforts focused on single CPU operations. The implications of these methods for parallel operation and load rebalancing were not addressed.

This work presents methods for the adaptation of viscous, unstructured mixed-element grids in parallel computing environments, where the mesh is partitioned by domain decomposition. Attention is restricted to conforming mesh modifications, to produce adapted grids that are not solver-dependent. Parallel cell subdivision methods are developed for mixed topology meshes with hexahedral and prismatic regions, and includes an interface pyramid refinement scheme. Purely tetrahedral viscous meshes with high aspect ratio near wall cells are now treated. A load balancing strategy is implemented which permits multiple adaptation passes by preventing child cells to reside on separate processors, ensuring a consistent restart process for each partition. These advances are demonstrated on practical applications of interest.

## II. Mesh Adaptation Methods

### A. Overview

The unstructured mesh adaptation package used in this work is *CRISP CFD*®,[9] a mesh modification and quality improvement code for three-dimensional mixed-element unstructured meshes. Meshes comprised of tetrahedral, prismatic, and hexahedral regions may be readily modified to generate more accurate flow solutions through local refinement and coarsening. In moving body applications, these coarsening and refinement methods may also be employed to accommodate boundary motion.

An estimate of the solution error is obtained to drive mesh adaptation. The method currently employed[9] is based on forming a higher order approximation of the solution at each mesh point using a least squares approach. The difference between the higher order reconstruction from incident nodes and the current solution forms the error measurement. If the current mesh is sufficiently fine to support the spatial variation in the solution, the estimated error will be low, allowing coarsening to take place. Conversely, a high degree of error indicates additional refinement is needed. This approach has proven successful in a variety of applications and is capable of detecting shear layers, separation, vortical flows, and weak gradients in coarse regions, as well as shocks and expansions.

Mixed-element meshes are treated as two regions, a tetrahedral region and a prism/hex region. Current mesh generators capable of creating mixed-element meshes, such as *GRIDGEN*®,[16] *SolidMesh*,[17] and *VGRIDns*,[18] form the unstructured mesh in a manner that produces distinctly separate tetrahedral, prismatic, and hexahedral topologies. The tetrahedral region is treated using a Delaunay refinement and edge collapse methods, while the prism/hex region is refined using cell subdivision methods. It should be noted that while *VGRIDns* is a tetrahedral mesh generator, the near wall viscous cell layers it creates can be combined into prisms through a post-processing step.

Upon completing the mesh modifications, the solution is interpolated onto the adapted mesh using a grid-transparent procedure based on nearby point clouds.[19] The method is founded on the use of volume coordinates for a containing tetrahedron as a set of basis functions. Using an efficient octree search procedure, four close nodes in the original mesh are identified that form a tetrahedron containing the new point in the adapted grid. These nodes may belong to any type of element, and the search is independent of the underlying cell topology. The use of volume coordinates guarantees the boundedness of the reconstructed values. Higher-order reconstruction is possible by adding a quadratic correction to the linear interpolation.[19]

### B. Coarsening and Refinement of Tetrahedra

The tetrahedral region of the grid is locally refined by means of a constrained Delaunay refinement algorithm combined with a circumcenter point placement strategy.[13] Any inconsistency between the circumradius of a tetrahedron and some desired point spacing triggers the point insertion procedure. This iterative cell refinement is repeated until the cell circumradii are consistent with the prescribed point spacing. Coarsening of the tetrahedral region is also permitted through an edge collapse procedure.[13] In regions where the grid is distorted or where solution errors are negligible, edges may be selected for removal. All cells incident to the deleted edge are removed from the mesh, the adjacent cells are redefined, and the two nodes of the edge are collapsed to a single vertex. This procedure is applicable for boundary edges as well as interior edges.

### C. Refinement of Hexahedra

The hexahedral cell refinement method implemented in *CRISP CFD*® is based on the method devised by Biswas and Strawn.[14] Each hexahedral cell is subdivided to generate new hexahedra, in 2:1, 4:1, or 8:1 patterns, as shown in Fig. 1. The center vertex pattern of Fig. 1(d) is used to terminate the propagation of refinement patterns by inserting a vertex at the cell center, creating six pyramids that may then be further subdivided to conform to the surrounding mesh. This procedure creates child cells of tetrahedra and pyramids, which are introduced between successive regions of refinement to close the grid and remove hanging nodes. In subsequent adaptation passes, these child cells are removed, the parent hexahedra are restored for further subdivision, and the cell splitting and child cell creation process is repeated. In addition to these patterns, a hexahedral cell may be subdivided into three prisms.

### D. Prism Refinement

The subdivision method for prism cells borrows from that employed for hexahedra, and is more general than the approach previously explored for tet/prism grids.[12,13] Each of the prism subdivision patterns depicted in Fig. 2 has been incorporated. The refinement of the three vertical edges of the cell, shown in Figs. 2(a), (d), and (e), requires the introduction of child elements to close hanging nodes. The center vertex pattern accomplishes this task by splitting the original prism into two tetrahedra and three pyramids. Each of these child cells is then further subdivided into additional tetrahedra or pyramids as necessary, based on how each of the five faces of the original prism is marked.

### E. Interface Pyramids

Proper subdivision of any interface pyramids bridging tetrahedra with either prisms or hexahedra requires that the pyramid base edges be split in the same manner as the adjacent prism or hex cells. The pyramids are then subdivided

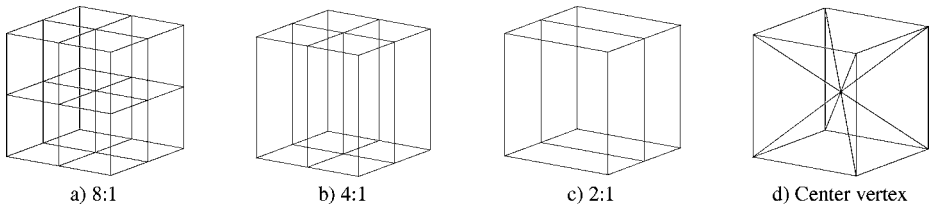


**Fig. 1 Subdivision of hexahedral cells.**

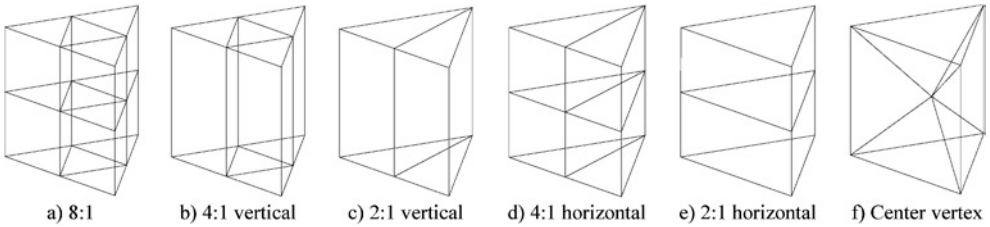


**Fig. 2 Prism subdivision patterns.**

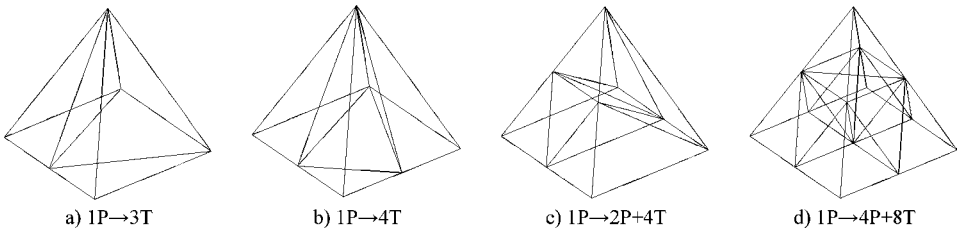


**Fig. 3 Subdivision of interface pyramids.**

into more pyramids and tetrahedra based on predefined patterns, illustrated in Fig. 3. In subsequent passes, child pyramids formed by 2P+4T and 4P+8T splits may be subdivided further. The tetrahedra created by these patterns may be lumped with the rest of the tetrahedral region. Child tetrahedra formed by 3T and 4T splits however must be kept separate from the Delaunay refinement and coarsening processes. As the pyramid refinement progresses, the base and vertical edges are marked so that the incident tetrahedra may also be split using patterns. The incident tetrahedra are split 2:1, 4:1, or 8:1, as in Fig. 4, to conform to the pyramids. Any incident boundary triangles are also subdivided in the process.

### F. Viscous Tetrahedral Meshes

Viscous meshes comprised purely of tetrahedra, such as those generated using an advancing layers method like *VGRIDns*,[18] are treated in two stages. The high aspect ratio, boundary layer tetrahedra are protected from the normal coarsening and Delaunay refinement methods applied to the isotropic, inviscid region of the mesh, and are instead treated in a separate cell subdivision procedure.

The boundary layer tetrahedra are first identified by computing a parameter $\beta$, which is a function of the cell aspect ratio $AR$ and distance from the wall $S$.

$$\beta = \frac{AR/\overline{AR}}{S/\bar{S}} \tag{1}$$

In Eq. (1), $\overline{AR}$ and $\bar{S}$ are the average values computed for all cells in the mesh. Including the wall distance serves to filter out high aspect ratio cells or slivers in the field that are not part of the boundary layer grid. The parameter $\beta$ decays as one leaves the boundary layer region, distance to the wall increases, and the cells become more isotropic. The boundary layer cells are selected as the set of all cells where $\beta$ is above a given threshold. In practice, selecting a threshold of 5 to 10 for $\beta$ is sufficient to identify the boundary layer cells. Once identified, these cells are protected from the coarsening and refinement processes and are treated in a separate subdivision step. These cells are split in 2:1, 4:1, or 8:1 patterns according to their edge markings. These subdivision patterns are depicted in Fig. 4. Using this approach, the structure of the boundary layer mesh is readily retained.

Figures 5 and 6 depict isosurfaces and contours of this quantity for a viscous grid about a finned missile. This case is a purely tetrahedral mesh decomposed on 4 processors. The decay in $\beta$ is observed as one leaves the boundary layer region. Figure 7 illustrates the improvement in viscous mesh adaptation for this case at two selected axial locations, with the original approach without boundary layer treatment presented on the left half of each image and the new

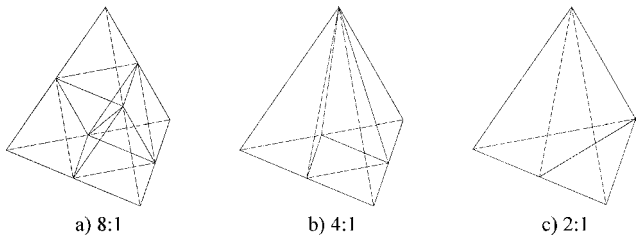


**Fig. 4 Subdivision of tetrahedral cells.**

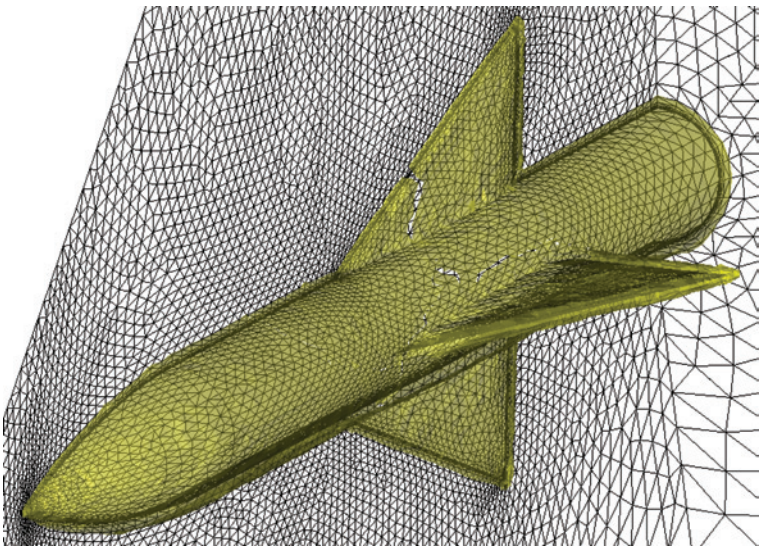

**Fig. 5 Isosurface of $\beta = 5$ for viscous tetrahedral mesh of a finned missile.**

approach on the right. When the viscous cells are not identified and protected, the boundary layer structure is clearly lost as these "poor quality" high aspect ratio cells are removed through the Delaunay refinement and smoothing processes.

Note that the tetrahedral cell subdivision procedure may also be used for the inviscid tetrahedra, as an alternative to the Delaunay refinement method.

### G. Parent Cells and Restart Data

The edge-based refinement procedures require restart data in order to restore parent cells for further subdivision in subsequent adaptation attempts. Parent cell definitions and their child cell numbers and cell types are stored for subsequent refinement passes. This parent/child cell data is formed for each parent cell type. In a fully mixed-element mesh one may have child cells of various types associated with tetrahedra, pyramids, prisms, and hexahedra. In parallel, a separate set of restart data is created for each processor after the mesh is rebalanced.

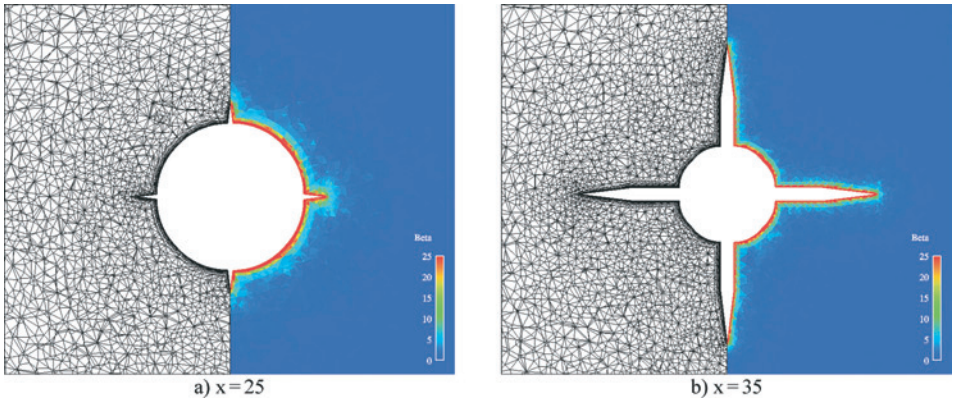


**Fig. 6 Selection of boundary layer cells using parameter $\beta$.**

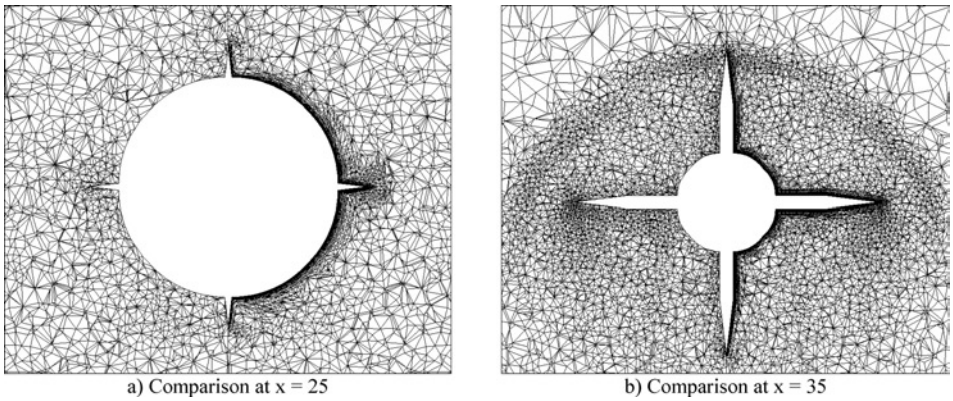


**Fig. 7 Mesh adaptation without (left) and with (right) boundary layer cell treatment.**

## III. Parallel Implementation

### A. Interprocessor Boundary Treatment

In parallel, each processor operates on its own partition, concurrent with and independent of the others. Therefore the first issue that arises in parallel mesh adaptation is how to treat the interprocessor boundaries. In the tetrahedral region, since coarsening is potentially taking place, such changes cannot be readily synchronized among the processors. The edge collapse operation is sequential, and edges must be removed in the same order for all processors sharing a common edge. Rather than modify these faces, the interprocessor boundaries are shifted using a cell migration technique.[9] The interprocessor faces and adjacent cells then become interior faces and interior cells, which may be readily modified through a second adaptation pass. In the second pass only the former interprocessor boundary region needs to be coarsened, refined, or smoothed, as the remainder of the mesh is already consistent with the prescribed point spacing.

Figure 8 illustrates the two-pass approach for coarsening and refinement of supersonic flow entering a duct. The original mesh partitions, shown in Fig. 8(a), are independently adapted to produce the mesh of Fig. 8(b). Note that the interprocessor boundaries are not modified, which leaves a region of the mesh that still requires adaptation. Several layers of cells are migrated from the right processor to the left in this example, as seen in Fig. 8(c). The interprocessor boundary is now to the right of its original location. A second coarsening and refinement pass treats the former interprocessor faces and adjacent cells, producing the final adapted grid of Fig. 8(d).

The cell migration is performed in cycles wherein pairs of adjacent partitions exchange data. In a given communication cycle, each processor forms a set of cells, faces, and vertices to exchange, starting at the interprocessor boundary and agglomerating the cells in successive layers. Two to five layers of cells are sufficient. These cell, face, and node sets are passed between adjacent processors. The decision to accept the set received or remove the set sent is based on the current load distribution, in an attempt to maintain load balance. If the grid on the current CPU is larger than that of the adjacent CPU, this domain will give up cells to its neighbor. The cell set received is discarded and the set sent is removed from the grid. Conversely, if the local grid is smaller than the adjacent grid, this domain will gain cells from its neighbor. The set sent is ignored and the set of cells, faces, and nodes received is appended to the grid. For any given partition, this approach ensures that the entire interprocessor boundary is treated since exchanges are done with each neighboring partition, thereby providing a sufficient shift in the interprocessor boundaries for the second adaptation pass for arbitrary decompositions. In the current implementation, all cells adjacent to the interprocessor boundary are migrated to the neighboring processor, for the sake of simplicity. Conceptually, it is possible that a more efficient procedure could be obtained by passing only those cells in need of refinement and/or coarsening, however such an approach has not been explored.

Parallel adaptation of the prismatic and hexahedral regions of the mesh is more straightforward. Since these elements are refined through subdivision patterns, a conforming mesh across interprocessor boundaries is ensured by first exchanging nodal tags for each of the interprocessor edges marked for refinement and then recomputing the patterns. This exchange guarantees that adjacent cells are subdivided in a consistent manner across processors.

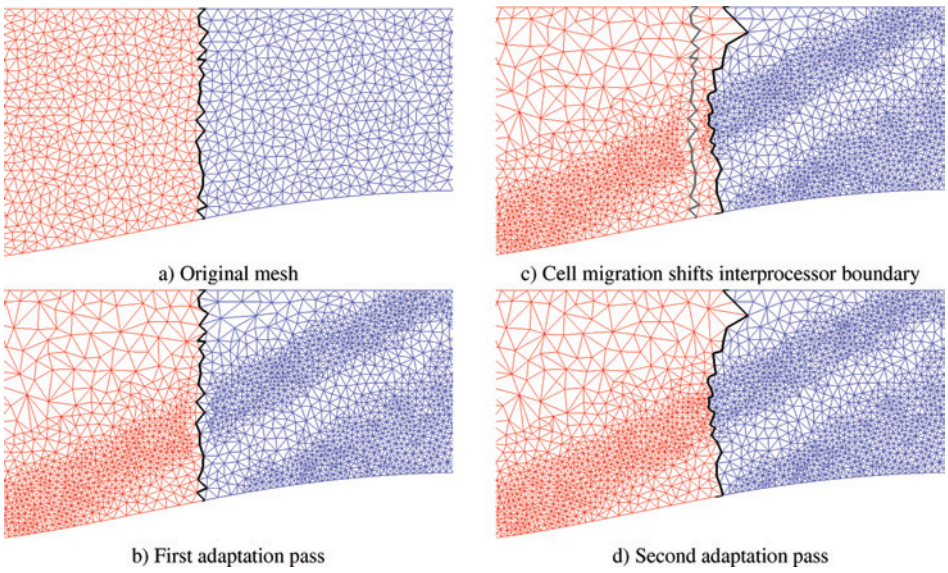


**Fig. 8 Two-pass approach for parallel coarsening and refinement of tetrahedral meshes.**

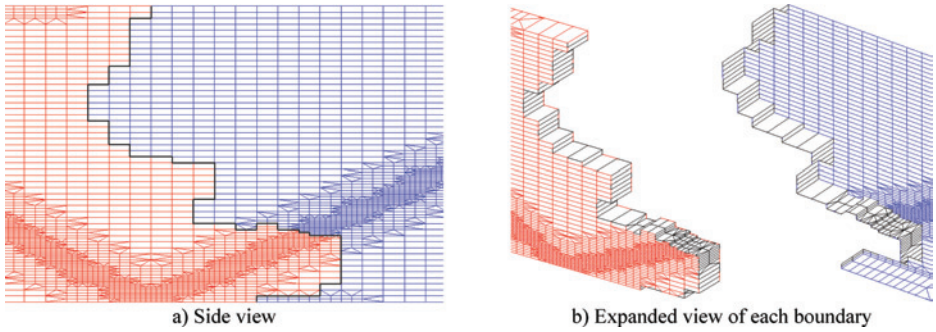


**Fig. 9 Interprocessor boundary treatment for edge subdivisions.**

Figure 9 demonstrates this pattern matching on an originally hexahedral mesh, where it may be seen that adjacent cells on the two partitions are modified in identical fashion to create a conforming mesh.

### B. Implications of Cell Migration

A consequence of the cell migration approach is that the shape and extent of the decomposed domains change. The cell migration process may introduce new pairs of adjacent domains that did not initially communicate. Similarly, pairs of processors that once shared common nodes, edges, and faces may become disconnected as a result of cell migration, since the decomposed domains are dynamically changing shape.

Updating the interprocessor communication schedule proceeds in two stages. First, the current communication lists are updated for each pair of adjacent domains. If no common nodes are found between two domains, the communication is removed from the cycle. The second stage involves checking for any new communication pairs introduced as a result of migration.

An efficient octree search procedure[20] is used to facilitate the communication update. First, the list of all interprocessor boundary nodes is stored in a message to be sent to the adjacent processor. Each pair of domains exchanges node lists, and the list received from the neighboring processor is placed into the octree. For each point in the list sent, the octree is searched to find a matching node on the adjacent domain. This process re-establishes the common node pairs for exchanging data in parallel once we return to the flow solver.

The search for new communication pairs involves additional steps. For each domain, the $(x, y, z)$ extrema are stored in a message to be sent to other processors. A list is formed of processors the local domain does not currently communicate with. For each domain in this list, the $(x, y, z)$ extrema are exchanged, and a check is performed as to whether the extrema of the two domains overlap. The subset of processors that pass this test are then put through the interprocessor node matching procedure to determine if a new communication pair is present.

In addition, one can no longer refer to the original decomposed grid to obtain data for solution transfer or for establishing point spacing after the coarsening phase. This issue is remedied by recomposing the global grid arrays at the start of the mesh adaptation process, such that the list of global vertex coordinates, solution vectors, and computed point spacings may be readily available to all processors. This allows global grid data to be readily retrieved as needed. Another motivation for re-establishing the global grid is to form the cell adjacency structure required by the load balancing method employed in this work. The method operates on the global cell numbers to diffuse cells across processors. After all mesh modifications are complete, the individual domains for each processor are concatenated to determine global cell and node numbering.

### C. Load Rebalancing

An adapted grid is inherently unbalanced, with mesh refinement taking place on certain partitions and coarsening taking place on others. Re-establishing load balance is desirable for steady-state problems, and essential for transient simulations, particularly after several adaptations. Load rebalancing is accomplished using the *ParMETIS* package[21] developed at the University of Minnesota. The global cell numbers and cell adjacencies are first formed for each of the existing partitions. At interprocessor boundaries, the adjacent global cell number on the neighboring processor must also be determined. The adaptive repartitioning routines in *ParMETIS* redistribute the existing partitions of the adapted grid using a diffusion concept.[21] New processor assignments for each cell in the global grid are obtained. From these new assignments, the balanced partitions may be re-formed.

Special care must be exercised when rebalancing the refined cells in the prism and/or hex regions of the grid. Figure 10 shows how *ParMETIS* may rebalance an adapted hexahedral mesh. The new partitions may be defined such that child cells of a given parent lie on different processors, as highlighted in Fig. 10 by the dotted circle. In this instance, the parent cell cannot be restored for further subdivision, and the refinement process cannot proceed. To properly restore a parent cell of any type, tetrahedron, pyramid, prism, or hexahedron, all child elements of that cell must reside on the same processor. This issue is remedied by assigning all child cells belonging to the same parent to the same processor. After the initial cell assignments are obtained from *ParMETIS*, the child cell list is examined for each parent cell, and all child cells are assigned to the same processor as the first child in the list. The processor assignment for the parent is set in the process. Figure 11 illustrates an example of hexahedral refinement and rebalancing with this new constraint applied. Examining the interprocessor region, it may be seen that the partition boundary does not cut across child cells. It should be noted that in Fig. 11 only a portion of the domain is shown in order to highlight the change in interprocessor boundary.

Another key issue associated with rebalancing grids adapted using pattern schemes and transition elements is the formation of consistent restart data for subsequent adaptations. Assembling the restart data for each processor is

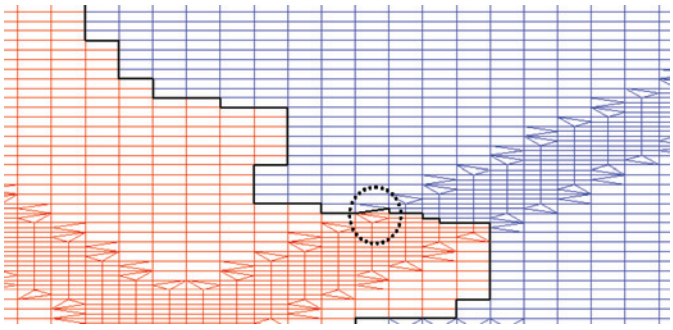

**Fig. 10 Partial reassignment of child elements during rebalancing.**

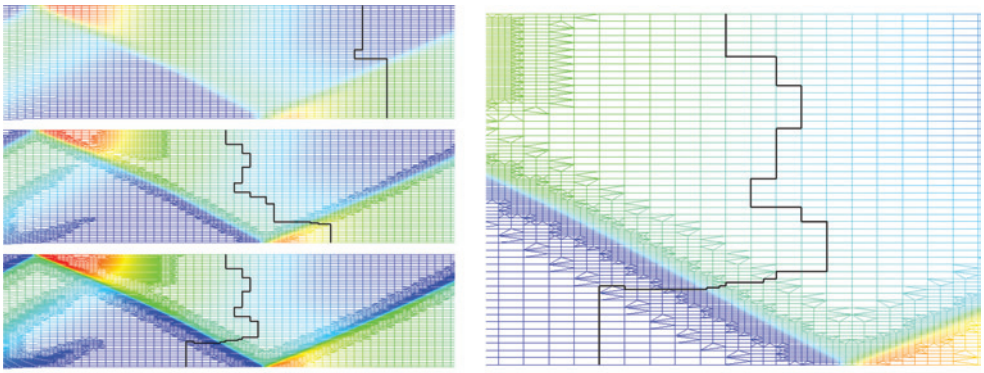


**Fig. 11 Parallel adaptation of an unstructured hexahedral mesh.**

**Table 1 Effect of migration strategy on load balancing.**

| Strategy | Rebalance time (sec) | Cells moved by ParMETIS |
|---|---|---|
| Smaller processor accepts cells | 37.44 | 61,346 |
| Larger processor accepts cells | 39.79 | 239,108 |
| Lower processor number accepts cells | 40.14 | 224,196 |
| Higher processor number accepts cells | 37.77 | 70,881 |

accomplished by referring to a global list of parent cells and their children. After all cell assignments are complete, each processor forms the new set of parent cells by searching the global list for parents assigned to it. In this manner, parent cells may shift across processors during the rebalancing procedure along with their children. This is performed for each parent cell type: tetrahedra, pyramid, prism, and hexahedra.

It is worth pointing out that the current cell migration strategy assists the load balancing process by minimizing the extent of the imbalance after mesh modifications are completed. By migrating cells from processors with more elements to those with fewer elements, the number of cells reassigned by *ParMETIS* is reduced. Table 1 details how the migration strategy impacts the CPU cost and amount of work done by *ParMETIS*. This example considers the tetrahedral grid for the finned missile depicted in Figs. 5–7, using 4 processors. The adaptation and rebalancing is performed on a Linux cluster employing 1 GHz Pentium III processors. The adapted mesh has approximately 860,000 cells total. Although the impact on CPU cost is minimal, the difference in the number of cells reassigned is significant. It is clear that the current migration strategy, where the smaller processor accepts cells, results in the fewest cells reassigned.

## IV. Flow Solution Method

A number of adaptation methods reported in the literature rely on embedded cell techniques where hanging nodes are permitted between levels of refinement. These non-conforming methods are designed for use with a specific solver, that is typically cell-centered, finite-element, or Cartesian. For such solvers a list of face pairs facilitates the flux calculation between levels of refinement on the adapted grid. The mesh adaptation methods implemented in *CRISP CFD*® are designed to be solver-independent, enforcing a conforming mesh. The resulting cell and boundary face definitions may be converted into various data structures, such as edge lists or face-cell adjacency arrays, as required.

The flow solver used in this work is *CRUNCH CFD*®.[22,23] *CRUNCH CFD*® is a parallel, implicit solver for 3-D chemically reacting turbulent real gases and dynamic domains. The basic numerical framework of the *CRUNCH CFD*® code is a finite volume higher-order Roe/Total Variation Diminishing (TVD) scheme in which the flow variables are defined at the vertices of the mesh. An edge-based data structure is employed wherein a polyhedral control volume is constructed from the union of all cells incident to a given node, and the control volume faces are associated with each edge. The inviscid flux calculation proceeds by looping over all edges in the mesh, and is

grid-transparent, while a cell-based method is employed to compute the flowfield gradients at the control volume faces for evaluating the viscous fluxes.[22] Turbulence modeling is provided by a k-$\varepsilon$ model. *CRUNCH CFD*® has also been validated for propulsive simulations involving gas phase chemical reactions. For additional details the reader is referred to the papers of Hosangadi et al.[22,23]

## V. Parallel Performance

### A. Parallel vs. Single Processor

In principle, the same adapted mesh should be produced for a given mesh and solution, irrespective of the number of processors employed. However in practice identical meshes cannot be obtained due to the iterative, sequential nature of the coarsening and refinement operations used to modify the tetrahedral cells. Figure 12 illustrates this for the supersonic duct flow considered previously. One adaptation is performed using 4 processors and a single processor. The meshes are not exactly identical, since the cells are coarsened and refined in a different order in the parallel case versus the single processor case. In parallel, each partition collapses its own list of edges, sorted according to local statistics. However, it is important to note that the resulting meshes are quite comparable, with nearly equivalent degrees of refinement and coarsening. These subtle differences between the adapted meshes are not considered to be significant.

The subdivision of the prismatic and hexahedral regions of a mesh exhibits very little variance between parallel and single processor operation, as shown in Fig. 13. One adaptation is again performed for the duct geometry using both 4 processors and a single processor, starting with a hexahedral mesh. Minor differences are observed in the edges subdivided at the fringe of the refined region.

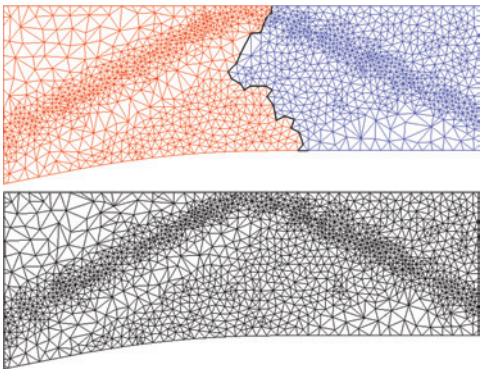

**Fig. 12 Tetrahedral meshes adapted in parallel (top) and single processor (bottom).**

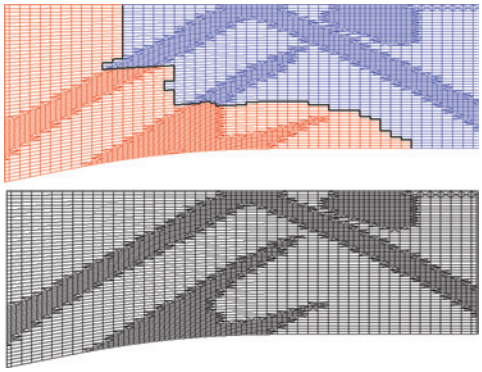

**Fig. 13 Hexahedral meshes adapted in parallel (top) and single processor (bottom).**

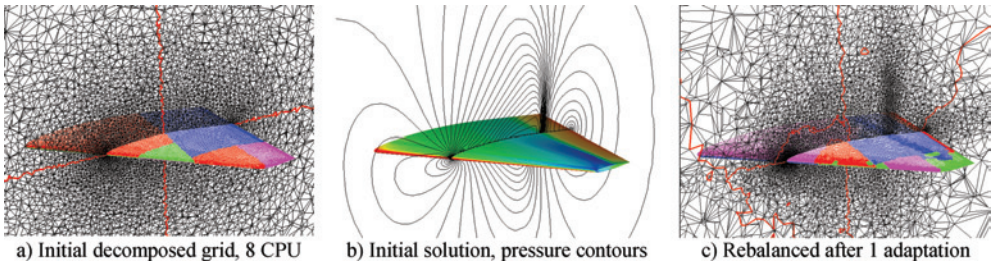


**Fig. 14 Inviscid transonic flow over a wing. Tetrahedral mesh used in scaling efficiency study.**

## B. Scaling Efficiency

An important aspect of parallel adaptation techniques is how well they scale with the number of processors employed. This is of particular interest if the methods are to be coupled with and invoked from the flow solver, as in transient applications. A scaling study was performed for the solution and adaptation of inviscid transonic flow over a wing, depicted in Fig. 14. The tetrahedral mesh used in this study was created using *VGRIDns*[18] and comprised of approximately 135,000 vertices and 768,000 cells. Flow solution and one adaptation were repeated, varying the number of processors. The performance of various segments of the *CRISP CFD*® adaptation package was assessed. As an example of how the mesh and interprocessor boundaries are modified for this case, Fig. 14 illustrates the initial mesh and solution, along with the rebalanced mesh after one adaptation, using 8 processors. The surface meshes are colored by processor assignment. A slice through the volume grid is also shown, for which the interprocessor boundaries are highlighted in red.

Table 2 summarizes the relative cost of the various operations associated with obtaining the initial solution and adapting the mesh once. Figure 15 illustrates the speedup of each operation, along with the overall speedup, relative to the ideal. Calculations were performed on an IBM P655 with 1.7 GHz Power4 processors. The mesh refinement operation scales quite well, and is comparable to the speedup of the flow solver. Mesh coarsening exhibits a reasonable

**Table 2 Performance of mesh adaptation operations.**

| Operation (all times in seconds) | | Number of processors | | | | | |
|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 8 | 16 | 32 | 64 |
| Flow solution | CPU Time | 5310 | 2720 | 1450 | 730 | 390 | 200 |
| | Speedup | 1.00 | 1.95 | 3.66 | 7.27 | 13.62 | 26.55 |
| Refinement | CPU Time | 75.420 | 32.965 | 18.661 | 9.775 | 5.353 | 2.919 |
| | Speedup | 1.00 | 2.29 | 4.04 | 7.72 | 14.09 | 25.84 |
| Coarsening | CPU Time | 5.655 | 3.165 | 1.576 | 0.988 | 0.656 | 0.320 |
| | Speedup | 1.00 | 1.79 | 3.59 | 5.74 | 8.63 | 17.69 |
| Transfer | CPU Time | 6.040 | 3.175 | 1.774 | 1.000 | 0.597 | 0.397 |
| | Speedup | 1.00 | 1.90 | 3.41 | 6.04 | 10.12 | 15.22 |
| Migration | CPU Time | 233.950 | 247.488 | 234.226 | 172.156 | 213.156 | 108.874 |
| | Speedup | 1.00 | 0.95 | 1.00 | 1.36 | 1.10 | 2.15 |
| Recomposition | CPU Time | 36.875 | 32.713 | 31.208 | 31.969 | 38.034 | 56.519 |
| | Speedup | 1.00 | 1.13 | 1.18 | 1.15 | 0.97 | 0.65 |
| Rebalancing | CPU Time | 69.745 | 39.383 | 23.095 | 23.738 | 19.766 | 26.798 |
| | Speedup | 1.00 | 1.77 | 3.02 | 2.94 | 3.53 | 2.60 |
| Overall | CPU Time | 5737.7 | 3078.9 | 1760.5 | 969.6 | 667.6 | 395.8 |
| | Speedup | 1.00 | 1.86 | 3.26 | 5.92 | 8.59 | 14.50 |

speedup, about half of the ideal value above 16 processors. The solution transfer operation also shows less than an ideal speedup, due to the fact that the same global point set must be searched regardless of the number of vertices on each adapted partition.

Cell migration, global grid recomposition, and load rebalancing are all overhead operations not present in single CPU operation of the code. These operations exhibit little to no speedup since the cost of these procedures is dependent on the size of the global adapted grid, and hence roughly the same amount of work is done regardless of the number of processors utilized, reducing the overall speedup. For instance, the rebalancing work is not merely the call to *ParMETIS*. It also includes the tasks of reforming the partition vertex, cell, and face lists from the global mesh. In addition, while the number of cells migrated to/from a processor reduce with the number of CPUs, the number of interprocessor boundaries rises, offsetting this gain. The migration method is nonetheless an effective treatment for coarsening the original grid on distributed processors. However this is also where any possible improvements in efficiency should be made, particularly before attempting transient applications. Potential improvements are currently being explored.

## VI. Applications

### A. Afterburning Missile Exhaust

This demonstration considers afterburning of the exhaust from a generic finned missile with forward strakes. The simulation is both turbulent and reacting. A simplified chemistry model assuming infinitely fast reaction rates is applied, in which the various exhaust species are lumped as three species of “fuel”, “oxidizer”, and “products”. Figure 16 depicts the missile geometry along with selected stations downstream of the nozzle exit plane where the mesh and solution will be assessed. Hexahedra are used to encompass the anticipated exhaust region, while prism cells are extruded from the missile surface to provide boundary layer resolution. The prism cells easily accommodate

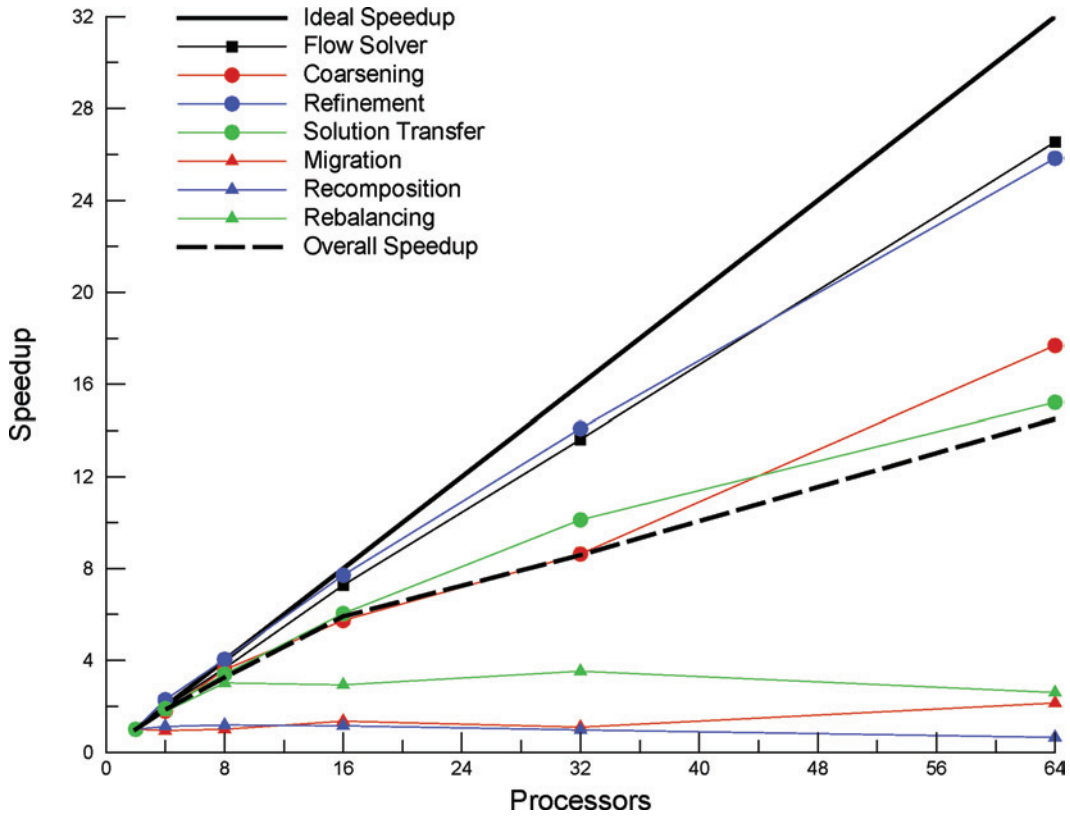


**Fig. 15 Speedup of various aspects of the parallel adaptation code.**

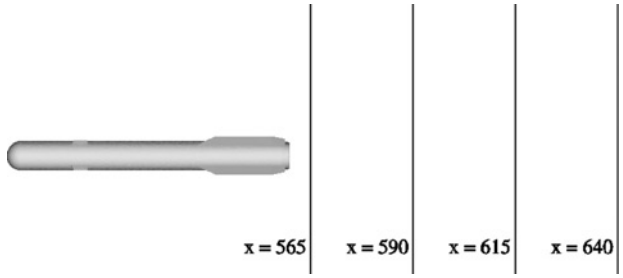


**Fig. 16 Missile geometry and plume stations.**

the geometric complexity of the fins and strakes. Tetrahedra are used to fill the remainder of the domain. Figure 17 illustrates the mesh topology employed in constructing the initial grid. The purple, orange, and green sections of the mesh denote the tetrahedral, prismatic, and hexahedral regions, respectively. Note the exhaust grid downstream of the missile also contains a small prism section. The mesh is decomposed on 16 processors.

In this scenario, the missile is at 20 degrees angle-of-attack in a Mach 0.95 freestream. At this angle-of-attack, the plume is deflected significantly, to the point where it falls outside the hexahedral region, becoming very diffuse. This traversal across cell topologies is evident in Fig. 18, which presents contours of product species mass fraction. Once the mesh is adapted, the flame region of the shear layer is thinner and better defined. Mesh induced diffusion is reduced as indicated by both the streamwise penetration of the jet and the deflection into the tetrahedral region. Figure 19 further illustrates the improvement with mesh adaptation. At each of the selected axial stations, mesh refinement better defines the cross-sectional shape of the deflected jet. Furthermore the refinement is seamless across the tetrahedral and hexahedral cell topologies.

Table 3 summarizes the changes in mesh size with each adaptation. The first adaptation was performed in less than 2 minutes, while the second adaptation required less than 15 minutes on a Linux cluster employing 1 GHz Pentium III processors. A total of 9 hours of CPU time was required to converge the solution on this sequence of meshes. There is a considerable increase in the number of cells and vertices, largely owing to the subdivision procedures for the prisms and hexahedra. The marked increase in the number of tetrahedra and pyramids is predominantly due to the introduction of child cells for hanging node closure. While the current simulation was retained on 16 processors, it is possible to increase the number of processors employed as the mesh grows in size. It is also possible that a mesh movement (or $r$-refinement) scheme combined with the current methods could serve to better optimize the mesh, providing increased resolution while reducing the number of cells refined through subdivision. Such studies could be explored in future work.

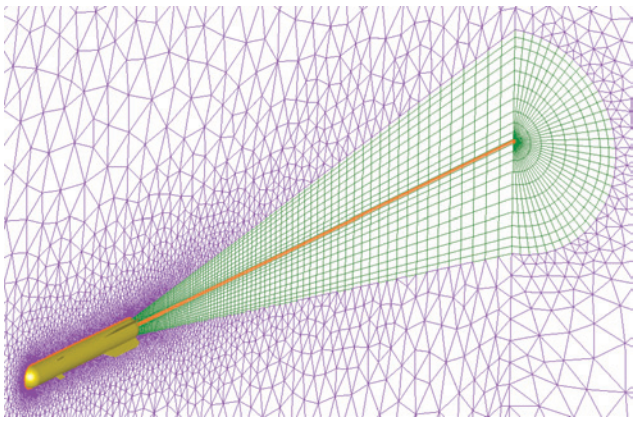

**Fig. 17 Overview of missile mesh topology.**

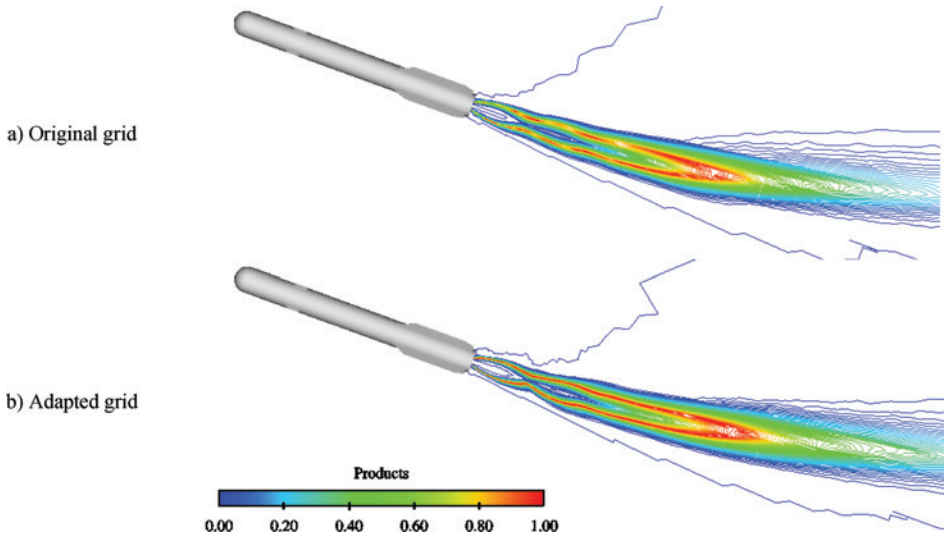


**Fig. 18 Missile exhaust product distribution in the symmetry plane.**

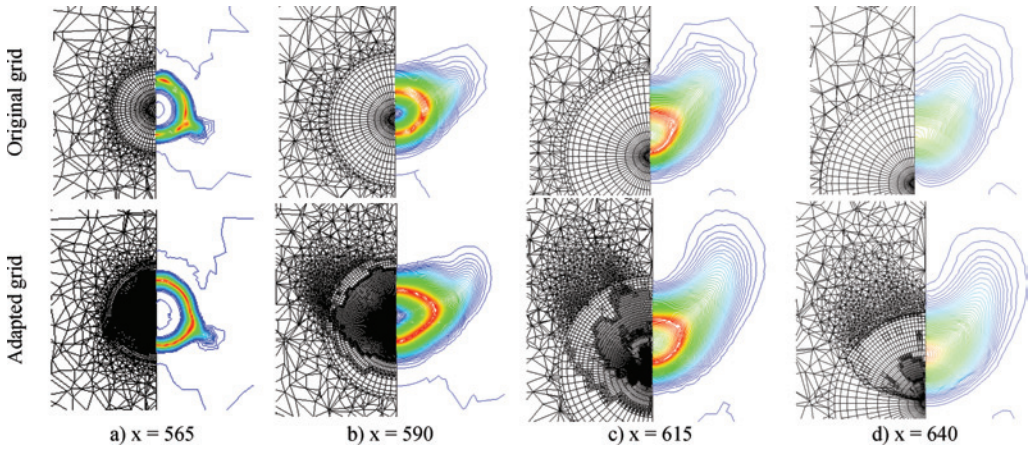


**Fig. 19 Product species comparison at selected axial locations.**

Figure 20 depicts the changes in the symmetry plane mesh and interprocessor boundaries. Two levels of refinement are seen in the hexahedral region of the mesh, along with refinement across topologies into the tetrahedral region as the exhaust plume is deflected by the freestream. Also note the coarsening of the tetrahedral mesh above and below the missile. The interprocessor boundaries, shown in red, are shifted each time the mesh is rebalanced. While a fairly

**Table 3 Adaptation statistics for missile exhaust case, 16 processors.**

| | Original grid | First adaptation | Second adaptation |
|---|---|---|---|
| Vertices | 226,569 | 601,876 | 1,849,645 |
| Tetrahedra | 272,984 | 1,145,087 | 2,903,460 |
| Pyramids | 3,600 | 206,624 | 618,964 |
| Prisms | 267,178 | 391,878 | 1,180,356 |
| Hexahedra | 36,000 | 142,427 | 576,409 |

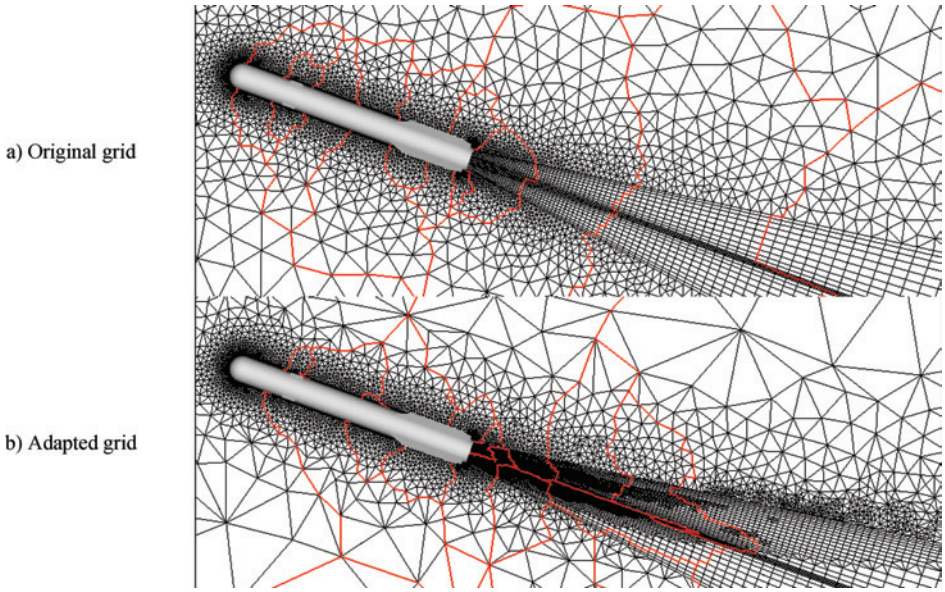


**Fig. 20 Symmetry plane mesh and interprocessor boundaries for missile exhaust.**

straightforward geometry and flowfield, this case successfully demonstrates the parallel refinement and rebalancing methods described earlier.

### B. Hydrogen Flow Control Valve

The second demonstration is a gaseous hydrogen valve, depicted in Fig. 21. In this scenario, the plug is set at a 52% open position, and an inlet pressure of 4400 psi is applied. The pressure drop across the valve is 2600 psi, which produces an underexpanded, annular jet at the nozzle throat between the plug and valve seat. The complex flowpath leads to formation of a large secondary flow and highly turbulent region in the exhaust duct downstream of the valve seat.

Figure 22 illustrates the mixed-element mesh topology. The purple, orange, and green sections of the mesh denote the tetrahedral, prismatic, and hexahedral regions, respectively. Hexahedral domains were constructed around the

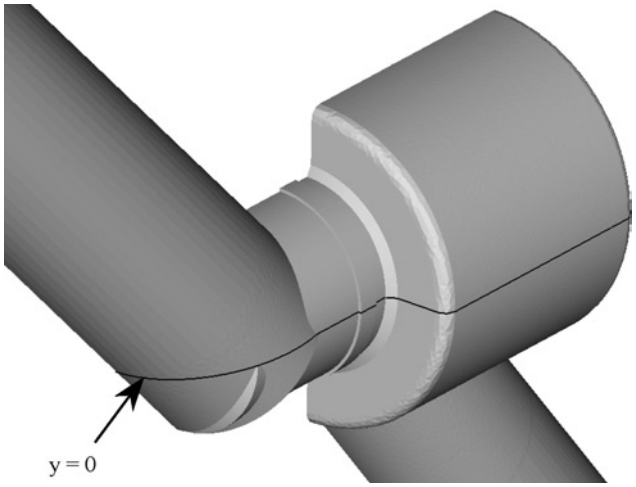


**Fig. 21 Outer surface of gaseous hydrogen valve depicting selected slice.**

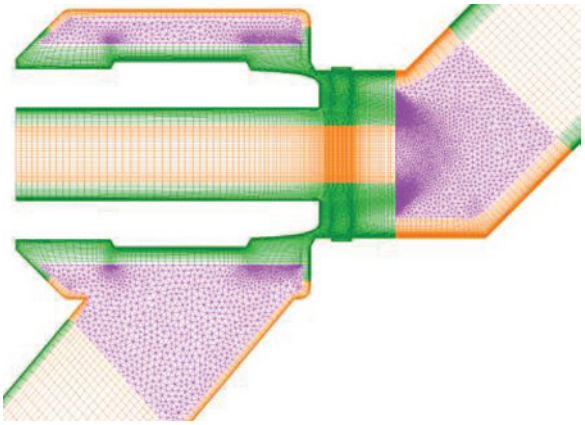

**Fig. 22 Symmetry plane with overview of mixed-element mesh topology.**

plug and in the narrow clearances of the seat region. The inlet and exhaust pipes were modeled with extruded hexahedra and prism cells. These portions of the mesh were connected to the domains in the plug and seat region using tetrahedra. The mesh was decomposed on 64 processors.

Adaptation has a notable impact on the computed flowfield, evident in the 3-D streamribbons shown in Fig. 23. The streamribbons are colored by the local Mach number. With the original mesh, nozzle flow entering from above is immediately turned to the bottom of the duct, producing a tight vortical flow, while a second, weaker vortical structure forms downstream in the exhaust duct. The flowfield upstream of the valve seat is rather incoherent. In contrast, the solution on the adapted grid exhibits several differences. Upstream of the seat, the flow negotiates the

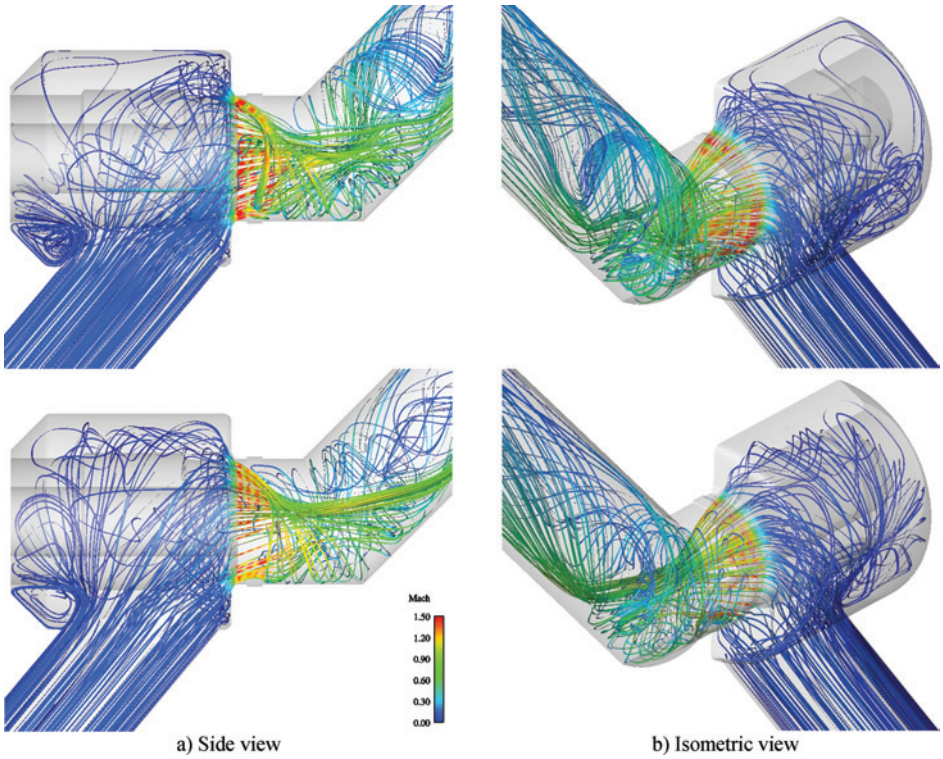


**Fig. 23 Streamribbon comparisons for original (top) and adapted (bottom) meshes.**

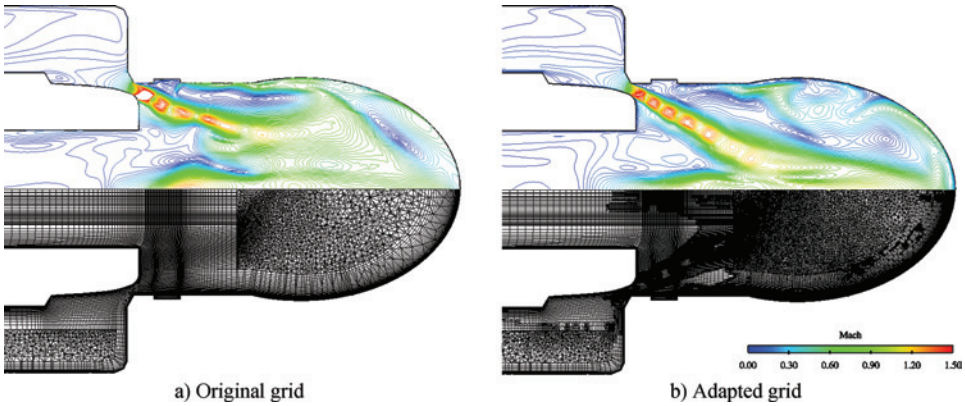


**Fig. 24 Mach number comparison in $y = 0$ plane.**

**Table 4 Adaptation statistics for hydrogen valve case, 64 processors.**

| | Original grid | First adaptation | Second adaptation |
|---|---|---|---|
| Vertices | 1,338,624 | 3,318,622 | 4,821,025 |
| Tetrahedra | 828,820 | 5,248,062 | 6,639,388 |
| Pyramids | 10,650 | 820,213 | 1,291,500 |
| Prisms | 437,626 | 878,241 | 1,560,404 |
| Hexahedra | 936,290 | 1,727,572 | 2,486,150 |

plug more effectively, entering the throat region more directly. This, combined with added resolution in the annular jet region, contributes to the greater degree of penetration observed. The vortical flow in the exhaust duct is also altered, producing a more coherent, tighter vortex in the upper section of the duct, and relaxing the lower vortex where the flow first impinges on the wall. Figure 24 further illustrates the improvement in jet definition with mesh adaptation. A slice through the mesh at the $y = 0$ plane is depicted, along with Mach number contours. After two adaptations, five distinct shock cells are observed in the annular jet. On the original mesh, these structures are diffuse and even exhibit some unsteady character.

Table 4 summarizes the mesh statistics for this case. Note the increase in the number of tetrahedra and pyramids, which is partly due to hanging node closure for the prisms and hexahedra. The flow solution was obtained on this sequence of meshes in 52 hours using an IBM SP with 375 MHz Power3 processors. Each adaptation required

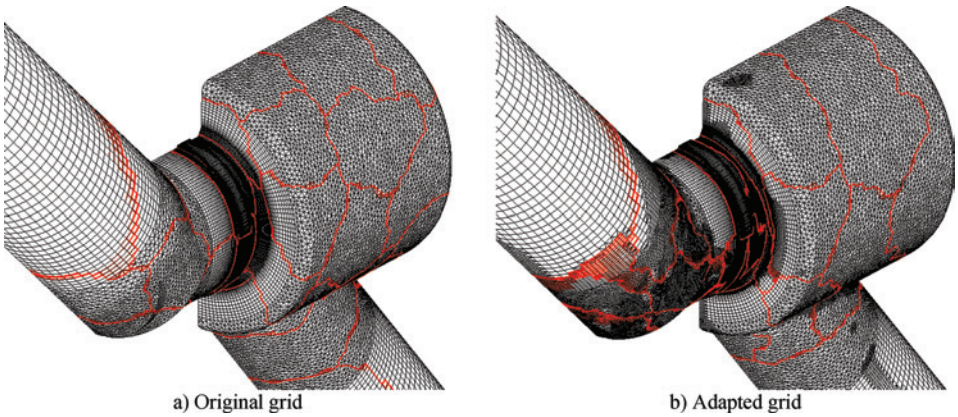


**Fig. 25 Surface mesh and interprocessor boundaries for hydrogen valve.**

approximately one hour of CPU time. The rebalancing of the adapted mesh among the 64 processors is illustrated in Fig. 25, along with the surface mesh. The interprocessor boundaries, shown in red, are shifted to maintain an approximately equal number of cells on each processor as the mesh is adapted. This case represents a practical application of the parallel methods in *CRISP CFD®* for a complex mesh of significant size, readily treating a mixed-element grid comprised of 3.3 million vertices and 4 million cells.

## VII. Conclusions

A parallel, unstructured mesh adaptation package, *CRISP CFD®*, has been developed to permit repeated refinement of mixed-element meshes comprised of any combination of tetrahedra, pyramids, prisms, or hexahedra. These recent developments in parallel, adaptive methods expand the ability of unstructured Navier-Stokes solvers to provide rapid, accurate detailed analyses and/or design assessments. Repeated adaptation allows the analyst to obtain grid-converged results for complex flowfields where the locations of relevant structures of interest are not known *a priori*. Validation studies were performed on selected problems illustrating the utility of the package for practical steady-state applications on multi-million cell models employing up to 64 processors. In addition to *CRUNCH CFD®*, the code presently supports the Air Vehicles Unstructured Solver (*AVUS*),[24] formerly known as $Cobalt_{60}$, *USM3D*[25] and the *FAST*[26] format. Filters for additional unstructured grid solvers may be readily developed for use with *CRISP CFD®*.

## Acknowledgments

Timothy Baker of Princeton University developed and provided the solution interpolation method used in this work. The Air Force Research Laboratory Computational Sciences Branch at Wright-Patterson AFB funded this research under contract F33615-02-C-3215, Phase II SBIR. Support for the hydrogen valve study was provided by NASA Stennis Space Center under contract NNS04AA08C, with Peter Sulyma as technical monitor.

## References

[1]De Keyser, J., and Roose, D., "Run-Time Load Balancing Techniques for a Parallel Unstructured Multi-Grid Euler Solver with Adaptive Grid Refinement," *Parallel Computing*, Vol. 21, No. 2, 1995, pp. 179–198.

[2]MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., and Packer, C., "PARAMESH: A Parallel Adaptive Mesh Refinement Community Toolkit," *Computer Physics Communications*, Vol. 126, No. 3, 2000, pp. 330–354.

[3]Soni, B. K., Koomullil, R., Thompson, D. S., and Thornburg, H., "Solution Adaptive Grid Strategies Based on Point Redistribution," *Computer Methods in Applied Mechanics and Engineering*, Vol. 189, No. 4, 2000, pp. 1183–1204.

[4]Jones, M. T., and Plassman, P. E., "Parallel Algorithms for Adaptive Mesh Refinement," *SIAM Journal on Scientific Computing*, Vol. 18, No. 3, 1997, pp. 686–708.

[5]Barry, W. J., Jones, M. T., and Plassman, P. E., "Parallel Adaptive Mesh Refinement Techniques for Plasticity Problems," *Advances in Engineering Software*, Vol. 29, No. 3–6, 1998, pp. 217–225.

[6]Oliker, L., Biswas, R., and Gabow, H. N., "Parallel Tetrahedral Mesh Adaptation with Dynamic Load Balancing," *Parallel Computing*, Vol. 26, No. 12, 2000, pp. 1583–1608.

[7]Flaherty, J. E., Loy, R. M., Shephard, M. S., Szymanski, B. K., Teresco, J. D., and Ziantz, L. H., "Adaptive Local Refinement with Octree Load Balancing for the Parallel Solution of Three-Dimensional Conservation Laws," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 2, 1997, pp. 139–152.

[8]Niekamp, R., and Stein, E., "An Object-Oriented Approach for Parallel Two- and Three-Dimensional Adaptive Finite Element Computations," *Computers and Structures*, Vol. 80, No. 3–4, 2002, pp. 317–328.

[9]Cavallo, P. A., Sinha, N., and Feldman, G. M., "Parallel Unstructured Mesh Adaptation Method for Moving Body Applications," *AIAA Journal*, Vol. 43, No. 9, September 2005, pp. 1937–1945.

[10]Cavallo, P. A., and Sinha, N., "An Adaptive Unstructured Mesh Approach for Aircraft/Store Compatibility Assessment," *Proceedings of NATO Symposium RTO-AVT-108*, Williamsburg, VA, June 7–10, 2004.

[11]Shipman, J. D., Cavallo, P. A., and Hosangadi, A., "Efficient Simulation of Aircraft Exhaust Plume Flows using a Multi-Element Unstructured Methodology," AIAA Paper 2001–0598, 39th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 8–11, 2001.

[12]Parthasarathy, V., and Kallinderis, Y., "Adaptive Prismatic-Tetrahedral Grid Refinement and Redistribution for Viscous Flows," *AIAA Journal*, Vol. 34, No. 4, April 1996, pp. 707–716.

[13]Cavallo, P. A., and Baker, T. J., "Efficient Delaunay-Based Solution Adaptation for Three-Dimensional Unstructured Meshes," AIAA Paper 2000–0809, 38th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 10–13, 2000.

[14]Biswas, R., and Strawn, R. C., "Tetrahedral and Hexahedral Mesh Adaptation for CFD Problems," *Applied Numerical Mathematics*, Vol. 26, No. 1–2, 1998, pp. 135–151.

[15]Mavriplis, D. J., "Adaptive Meshing Techniques for Viscous Flow Calculations on Mixed Element Unstructured Meshes," NASA CR-201675, 1997.

[16]Gridgen, Grid Generation Software Package, Ver. 15, Pointwise, Inc., Ft. Worth, TX, 2004.

[17]Marcum, D. L., and Gaither, J. A., "Mixed Element Type Unstructured Grid Generation for Viscous Flow Applications," AIAA Paper 99–3252, 14th Computational Fluid Dynamics Conference, Norfolk, VA, June 28–July 1, 1999.

[18]Pirzadeh, S., "Progress Towards a User-Oriented Unstructured Viscous Grid Generator," AIAA Paper 96–0031, 34th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 15–18, 1996.

[19]Baker, T. J., "Interpolation from a Cloud of Points," *Proceedings of the 12th International Meshing Roundtable*, Santa Fe, NM, September 14–17, 2003.

[20]Baker, T. J., "Generation of Tetrahedral Meshes Around Complete Aircraft," *Proceedings of the 8th International Conference on Numerical Grid Generation*, Pineridge Press, Swansea, UK, 1988, pp. 675–685.

[21]Schloegel, K., Karypis, G., and Kumar, V., "A Unified Algorithm for Load-Balancing Adaptive Scientific Simulations," Technical Report, 00-033, University of Minnesota, Department of Computer Science and Engineering, 2000.

[22]Hosangadi, A., Lee, R. A., Cavallo, P. A., Sinha, N., and York, B. J., "Hybrid, Viscous, Unstructured Mesh Solver for Propulsive Applications," AIAA Paper 98–3153, 34th Joint Propulsion Conference, Cleveland, OH, July 13–15, 1998.

[23]Hosangadi, A., Lee, R.A., York, B.J., Sinha, N. and Dash, S.M., "Upwind Unstructured Scheme for Three-Dimensional Combusting Flows," *Journal of Propulsion and Power*, Vol. 12, No. 3, May–June 1996, pp. 494–503.

[24]Strang, W. Z., Tomaro, R.F., and Grismer, M.J., "The Defining Methods of Cobalt$_{60}$: A Parallel, Implicit, Unstructured Euler/Navier-Stokes Flow Solver," AIAA Paper 99–0786, 37th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 11–14, 1999.

[25]Frink, N. T., "Tetrahedral Unstructured Navier-Stokes Method for Turbulent Flows," *AIAA Journal*, Vol. 36, No. 11, November 1998, pp. 1975–1982.

[26]Walatka, P. P., Clucas, J., McCabe, R. K., Plessel, T., and Potter, R., "FAST User Guide," NAS Technical Report RND-93-010, June 1993.