5B.1

Parallel Mesh Adaptation for Highly Evolving Geometries with Application to Solid Propellant Rockets

Damrong Guoy[1], Terry Wilmarth[1], Phillip Alexander[1], Xiangmin Jiao[2], Michael Campbell[1], Eric Shaffer[1], Robert Fiedler[1], William Cochran[1], Pornput Suriyamongkol[1]

[1] Center for Simulation of Advanced Rockets, Computational Science and Engineering Program, University of Illinois at Urbana-Champaign
[2] College of Computing, Georgia Institute of Technology

Summary. We describe our parallel 3-D surface and volume mesh modification strategy for large-scale simulation of physical systems with dynamically changing domain boundaries. Key components include an accurate, robust, and efficient surface propagation scheme, frequent mesh smoothing without topology changes, infrequent remeshing at regular intervals or when triggered by declining mesh quality, a novel hybrid geometric partitioner, accurate and conservative solution transfer to the new mesh, and a high degree of automation. We apply these techniques to simulations of internal gas flows in firing solid propellant rocket motors, as various geometrical features in the initially complex propellant configuration change dramatically due to burn-back. Smoothing and remeshing ensure that mesh quality remains high throughout these simulations without dominating the run time.

1 Introduction

Many physical systems involve material interfaces with dynamically changing geometries. Examples include the gas in the vicinity of parachutes, airfoils, helicopter blades, turbine engines, and burning propellants in rockets. Several approaches to discretizing the gas (fluid) and solid (structural) domains in problems of this type may be adopted (embedded boundary method, immersed boundary method, Chimera overset structured grids, ghost fluid method, etc.), but since an accurate representation of the behavior of the solution near the interface is often vitally important to the evolution of the solution throughout the entire fluid domain, body-fitted grids seem to be the most appropriate choice. However, maintaining adequate mesh quality for a body-fitted grid as the domain boundary deforms significantly with time can be a very challenging problem, particularly in large-scale 3-D parallel simulations. In this paper, we describe our multi-tiered, dynamic, parallel mesh modification strategy and demonstrate its effectiveness for two real-world rocket problems. It should be noted that our meshing strategy is broadly applicable to many types of problems with highly evolving domains.

In simulations of physical systems with deforming geometry, the location of the material interface is often part of the solution (e.g., parachutes), rather than a surface whose motion is prescribed as a boundary condition (e.g., a rigid sphere moving through the air). While it may be possible to describe the initial configuration of a parachute using a CAD model, once the chute begins to deform due to its interaction with the air, the only available description of its geometry is the deformed surface mesh of the chute. Surface propagation driven by a discretized velocity field is in general a very challenging problem, which we address using the Face-Offsetting Method.

Generating a new mesh at advanced simulation times requires the ability to derive (locally) a smooth representation from the discrete surface mesh [1]. When a new mesh is needed, we perform surface and volume mesh generation using a package from Simmetrix, Inc. Partitioning the new mesh for parallel execution of the simulation may be accomplished using METIS or ParMETIS, but especially for very large meshes, our recently developed and integrated hybrid geometric partitioner provides superior performance and robustness.

Unfortunately, generating a new mesh and transferring the numerical solution from the old mesh to the new one is both time-consuming and introduces interpolation errors. Therefore, in our simulations we delay remeshing by frequently applying Mesquite from Sandia National Laboratories to smooth and maintain mesh quality without changing mesh topology. Our parallel implementation applies a serial version of Mesquite in a concurrent manner. Interpolation of the solution is not necessary when the mesh is smoothed, allowing us to smooth every few time steps without decreasing solution accuracy.

These remeshing, data transfer, partitioning, and smoothing capabilities have all recently been integrated in the *Rocstar* rocket simulation package to enable us to perform large-scale simulations of physical systems with highly evolving domain boundaries without greatly increasing the wall clock time (compared to simulations of problems with fixed domains). We present results for two rockets with complex changing internal geometries. Experience with these real-world applications has helped us find better criteria for determining when to remesh.

Related work: A number of researchers have worked on mesh generation for dynamically changing geometries. Here we mention several of the methodologies most closely related to our approach. Baker [2] proposed to perform smoothing, coarsening, and refinement at each time step during the geometric evolution. The paper demonstrated effectiveness of the meshing technique without a physics solver. Folwell et al. [3] showed that mesh smoothing can delay abort-time of a computational electromagnetic code TAU3P. Wan et al. [4] successfully applied local mesh modification to metal forming simulations and compared the results with remeshing. Cardoze et al. [5] used bezier-based curved elements with refinement, coarsening, and edge smoothing for dynamic blood flow simulation in two dimensions. Dheeravongkit and Shimada [6] performed pre-analysis to adjust the input mesh for the anticipated deformation. The method was demonstrated successfully for forging simulations in two dimensions. Giraud-Moreau et al. [7] described a geometrical error estimate for adaptive refinement and coarsening of triangular and quadrilateral surface meshes. It was applied to a thin-sheet metal forming simulation using ABAQUS. Compared to the previous works, we focus on large-scale parallel 3-D simulations with emphasis on software integration of various meshing tools.

2 System Integration Overview

The *Rocstar*[8] simulation suite consists of a large number of dynamically loaded software modules coupled together to solve fluid/structure interaction problems including moving and reacting interfaces. Each module encapsulates a physics solver or some service needed by the simulation. Figure 1 depicts the *Rocstar* architecture. This parallel code uses MPI (or AMPI[9]) and typically runs on hundreds to several thousand processors in a batch environment. All module interactions, including the communication of data and function calls, are mediated by a general integration interface. The integration framework is designed to allow each module to retain its own data structures, representation, and parallelism.

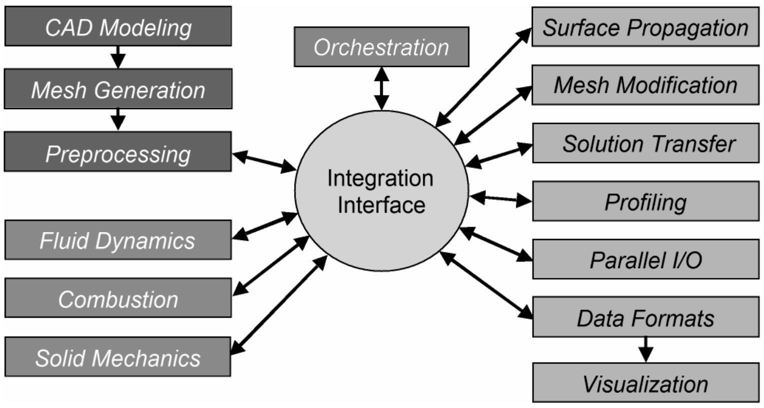

Fig. 1. *Rocstar* simulation component overview.

Our multi-tiered approach to mesh modification involves the steps shown in Figure 2. First, the surface is propagated according to the solution of the physical system and the surface mesh is smoothed to maintain quality. Next, we smooth the volume mesh to maintain its quality, and update the solution field variables. Eventually, the change in geometry becomes so severe that the smoothing operations cannot maintain acceptable mesh quality, and global remeshing is triggered. More details on surface propagation, smoothing, and remeshing are presented in subsequent sections of this paper.

1. Propagate surface according to physical solution
2. Smooth surface mesh
3. Smooth volume mesh
4. Physics solvers update solution
5. If mesh and solution are acceptable then continue (Step 1)
6. If mesh or solution quality is too low, trigger remeshing (Step 7)
7. Global remesh of geometry at last good checkpoint
8. Transfer solution from old mesh to new and generate new checkpoint
9. Restart simulation from new checkpoint and continue (Step 1)

Fig. 2. Integrated surface propagation, solution update, and meshing procedures.

When triggered, the remeshing module reads the simulation's last restart disk files (checkpoint) and writes a full set of simulation restart files. This allows our remeshing software to access the mesh and solution data of any domain in an identical manner regardless of the source, and also ensures that we remesh from a known good geometry and solution. Thus, the result of a remeshing phase (trigger, remesh, transfer solution, generate restart files) looks like a normal full simulation restart. This is done in such a way that the batch job does not terminate.

We have explored several strategies for deciding when to remesh. Our first simple strategy is geometry-based triggering. In this strategy, we examine the mesh after each time step and make a decision based on some mesh quality metric such as minimum dihedral angle or aspect ratio of an element. This strategy is easily implemented since it does not rely on any solution data.

In our experience, geometric triggering is insufficient because the mesh may become distorted in such a way that its quality is not bad per se, but the physics solver can no longer get an accurate solution in some part of the mesh. A trivial way to deal with this is to trigger remeshing at fixed intervals short enough so that the mesh is not expected to deform significantly. This strategy is easily implemented and works well for maintaining a good quality mesh, but may be unnecessarily expensive in terms of wall clock time.

We also allow our physics solvers to initiate remeshing based on their own internal decision-making process, whatever that may be. One of our fluid solvers triggers remeshing based on a threshold for the minimum internal timestep. This solver ensures that its explicit time step does not exceed the Courant limit (the shortest time for a signal to cross any cell). This strategy is slightly better than geometry-based triggering, since the local fluid solution is given some weight in the decision-making process.

We believe that the best possible remeshing triggering strategy would be one based on an analysis of the solution accuracy in each domain. This would require internal solver error estimators which are currently under development. We plan to add this capability in the near future.

3 Surface Propagation

Dynamic moving surfaces arise in many scientific and engineering applications, such as crystal growth, dendritic solidification, microchip fabrication, multiphase flows, combustion and biomedical applications. Because singularities and topological changes may develop during the evolution of the surface, propagating it numerically poses daunting challenges in developing accurate, efficient, and robust techniques.

To address the problem, we have developed a novel method for surface propagation, called the *face-offsetting method* (*FOM*) [10]. FOM propagates an explicit surface mesh without requiring a volume mesh, but unlike traditional Lagrangian methods, FOM propagates the faces of the mesh and then reconstructs the vertices from the faces by performing an eigenvalue analysis locally at each vertex to resolve the normal motion (for obtaining surface geometry) and tangential motion (for maintaining mesh quality) simultaneously. This new approach enables us to obtain accurate physical solutions for both advective and wavefrontal types of motion, even at singularities, and to ensure the integrity of the surface as it evolves.

The steps of the face-offsetting method are outlined in Figure 3. The first step “offsets” each individual face by integrating the motion of its vertices or quadrature points, and hence the method is named “face offsetting.” After offsetting, the faces may no longer be connected to each other, and the second step reconstructs each vertex to be the “intersection,” in a least squares sense, of the offsets of its incident faces. The displacement $\mathbf{d}_i$ of the ith vertex is then estimated as the vector from the vertex to the intersection, which essentially propagates the vertex to the point that minimizes a weighted sum of the squared distances to the offsets of its incident faces. This computation results in a linear system $\mathbf{Ad} = \mathbf{b}$ at each vertex, where $\mathbf{A}$ is a 3×3 matrix and $\mathbf{d}$ and $\mathbf{b}$ are 3 vectors. Robustness of the computation is achieved through an eigenvalue analysis of the matrix $\mathbf{A}$.

We then update the tangential motion of the vertices to maintain good mesh quality (step 4). More specifically, we restrict the motion to be within the null space of the matrix $\mathbf{A}$, as it tends to introduce minimal errors. Redistributing the vertices in this manner is referred to as *null-space smoothing* and was proposed in [12]. To maintain a mesh that is free of self-intersections, we next determine the maximum propagation time step $\alpha\Delta t$ that prevents

1. Propagate each face by time-integration of motion at its vertices or quadrature points
2. Reconstruct vertices to estimate displacement $\mathbf{d}_i$ at each vertex $\mathbf{v}_i$
3. For wavefrontal motion, correct vertex displacement $\mathbf{d}_i$ based on Huygens' principle[11]
4. Compute tangential motion $\mathbf{t}_i$ for each vertex $\mathbf{v}_i$ to maintain mesh quality
5. Compute maximum time step $\alpha \Delta t$ that prevents mesh self-intersection for $\alpha \leq 1$ and move each vertex by $\alpha(\mathbf{d}_i + \mathbf{t}_i)$

Fig. 3. Outline of the face-offsetting method.

mesh folding for $\alpha \leq 1$, and then move each vertex $\mathbf{v}_i$ to $\mathbf{v}_i + \alpha(\mathbf{d}_i + \mathbf{t}_i)$ (step 5).

If the motion is wavefrontal, such as in burning or etching, the estimated displacement from step 2 yields a poor approximation, and therefore an additional step is needed to correct the vertex displacements. This is done by adjusting the face offsets at expansions based on Huygens' principle[11] (step 3), so that the solution erodes expanding features. The effectiveness of the face offsetting method can be seen in Figures 10, 12, and 13.

4 Parallel Mesh Smoothing

We perform mesh optimization in *Rocmop*, a module which improves the quality of unstructured volume meshes through nodal repositioning, a process referred to as mesh smoothing. The purpose of smoothing is to delay degradation of mesh quality (and therefore the need for remeshing) due to the geometric evolution of the domain boundary as the simulation progresses. Because our fluid and structural dynamics equation solvers are formulated on moving meshes (i.e., Arbitrary Lagrangian-Eulerian [ALE] formulation), no interpolation of the solution is required when the mesh is smoothed. Instead, the equations of motion include the contribution to the change in the solution due to mesh motion. This allows us to smooth the mesh as often as desired without decreasing solution accuracy.

Nodes on the domain boundary move (at a nominal rate of about 1 cm/s in solid propellant rockets) due to the physics of the problem. Our fluid dynamics solver relies on *Rocmop* to move interior nodes, in particular those close to the surface, in order to avoid rapid generation of highly skewed elements there. *Rocmop* must be called at least every few time steps to maintain the same mesh quality as a run with smoothing performed every fluid time step. The main limitation appears to be stability of the numerical solution: if smoothing is not called often enough, the smoother makes more drastic changes to the nodal locations when it finally is invoked, which can cause the numerical solution to blow up.

An important feature of any module that may be called many times during a simulation is efficiency. In an early implementation of *Rocmop*, we noticed that more computational time was spent in mesh smoothing than in the fluid solver. After careful performance tuning, if called once every 3 fluid steps, *Rocmop* now uses around 30 percent of the total wall clock time. One lesson learned was that the execution time to evaluate the mesh quality measure (maximum dihedral angle) is significant compared to one time step of the physics solver. We wanted to check at every call whether this measure was below some specified threshold, and to perform additional smoothing if improvement was needed, but that turned out to be quite impractical. We now normally avoid computing the quality measure altogether.

Rocmop calls Mesquite [13, 14, 15], a powerful sequential mesh-smoothing package from Sandia National Laboratories. We apply Mesquite concurrently to each mesh partition independently, and then use a simple averaging scheme to realign the nodes shared by neighboring partitions. We were concerned that this averaging would degrade mesh quality, and were computing mesh quality metrics at substantial computational cost to ensure that mesh quality remained high. This turned out to cost far more than it was worth. We saw little improvement when we repeated the process of smoothing the mesh partitions independently and then averaging the coordinates of partition boundary nodes.

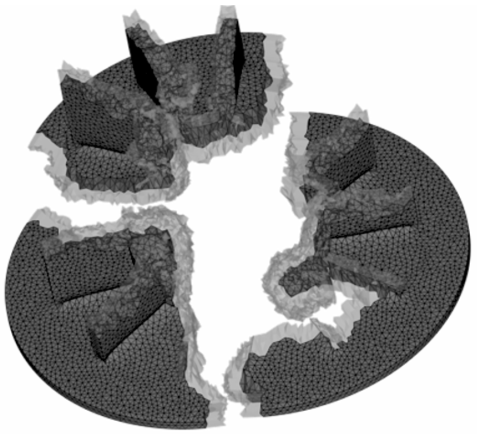

Fig. 4. Three mesh partitions with their ghost cells.

Note that Mesquite does not move nodes on surface patches or partition boundaries. As mentioned in Section 3, the surface mesh is smoothed frequently by our surface propagation module. Unlike most surface patches, partition boundaries do not often have a very regular geometry, especially when using a tetrahedral mesh. We have found that it is extremely important to allow Mesquite to relocate nodes on partition boundaries freely, as if they were interior nodes. We accomplish this by passing to Mesquite the real elements (local to a partition) plus the *ghost* cells (non-local elements) that share at least one node with a real element on the partition. The ghost nodes are used in the fluid solver's parallel implementation to store local copies of quantities needed from neighboring partitions. Figure 4 shows an example of three mesh partitions with their ghost cells. Utilizing these expanded mesh partitions, Mesquite can move the nodes on the partition boundary to their optimal locations.

5 Remeshing and Solution Data Transfer

We have developed a suite of remeshing and solution transfer tools called *Rocrem* to carry out a complete on-line remeshing process to improve the surface and volume quality of 3-D unstructured tetrahedral meshes. This approach is utilized when other methods, such as smoothing and local mesh repair, fail to improve mesh quality sufficiently for the continued use of the mesh (i.e. by a physics solver). *Rocrem* performs two key tasks: the generation of a better quality mesh and the transfer of solution data from the old mesh to the new mesh.

5.1 Surface Remeshing

Because of the use of commercial off-the-shelf (COTS) software for the mesh generation component of *Rocrem*, the first step is the serialization of the partitioned mesh. The surface mesh, like the physical domain boundary that it represents, is organized into *patches*, each having a particular boundary condition (BC). Each surface mesh partition may include regions ("*panes*") from various patches. The panes on each partition are first "stitched" together into a single surface mesh with patch information preserved by including it with the data for each triangular element. These surface partitions are then sent to the rank zero processor, and stitched together to form the serial surface mesh.

This serial mesh is then imported to Simmetrix's Simulation Modelling Suite (SMS). The SMS constructs a *discrete model* from the surface mesh. The patches and BCs are preserved by specifying each patch to Simmetrix as a *model face*, and assigning each triangular element to the appropriate model face.

At this point, mesh sizing prescribed by user-specified parameters is imposed, ranging from absolute sizing to fine control of the sizing relative to the discrete model. A new surface mesh is then generated according to the discrete model and these sizing parameters.

There are a few special situations that must be handled at the surface remeshing stage that are of particular importance in simulations with evolving mesh geometries. One is the disappearance of geometric features, and the other is the far more complicated disappearance of entire patches. The techniques for addressing these situations are notable in that they expand the functionality of our remeshing tools (beyond merely improving the quality of a mesh) into the realm of deviating slightly from the discrete model to properly continue the mesh evolution during a simulation. We describe each of these situations below.

Disappearing Geometric Features: Geometric feature information is determined by the SMS, and during surface remeshing those features are maintained by default. However, as the mesh evolves, certain features may become smaller and might eventually disappear, while others will arise and expand. As a feature disappears, its size will first become smaller than that of the edge sizing desired in the discretization. This will cause the appearance of poor quality elements, whose edges have a very high aspect ratio. These elements will invariably result in poor quality elements in the volume mesh. To remove such features and thereby recover higher quality in both the surface and volume meshes, it is necessary to deviate slightly from the discrete model that was determined from the original surface mesh. This is done by carefully selecting the parameters to the surface mesh modification function provided by the SMS. In particular, the SMS is instructed to remove poor quality elements that fall below a specified aspect-ratio threshold. This approach allows for the removal of a variety of features at the appropriate times during simulations with evolving geometries, such as the star-shaped region in some solid propellant rockets (see Section 7).

Disappearing Patches: A disappearing patch is a special case of a disappearing geometric feature. It arises when a geometric feature is shrinking and about to disappear, but the entire feature itself is a patch that was specified to the SMS discrete model as being a model face. While disappearing features on the same model face can be removed as described in the previous section, a model face that has a width of a single layer of very thin, poor quality elements cannot be removed without violating the model.

To handle this case, the mesh is examined to determine which patch is disappearing, and then the patch is reassigned to a neighboring patch. This allows for the removal of the thin elements using the same approach as in the previous section. The neighboring patch effectively takes over the space occupied by the reassigned patch.

In our rocket simulations, this situation typically arises when a burning surface neighboring an inhibited (non-burning) surface propagates in such as way as to eliminate the inhibited surface. When the inhibited surface is reduced to a thin ring and the solver cannot continue due to the poor element quality in that area, the inhibited surface elements are reassigned to the encroaching burning surface. Figure 5 illustrates this situation.

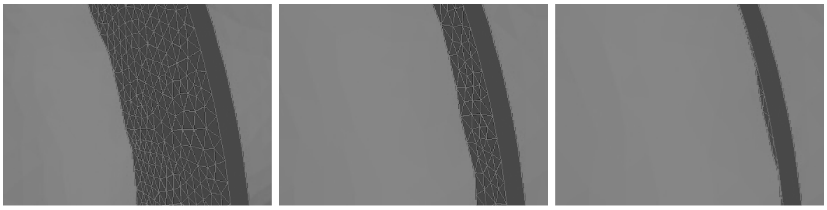
(a) 52.3s (b) 87.6s (c) 98.8s

Fig. 5. A geometric feature that is an entire patch in the RSRM joint-slot disappears over time.

Our current implementation requires user intervention in determining to which neighboring patch the disappearing patch should be reassigned. We are in the process of developing a means to detect this situation and automatically reassign the patch.

5.2 Volume Remeshing

Once the surface mesh is obtained, a volume mesh is generated. This typically results in a mesh with a coarser discretization deep in the interior. To control the interior mesh sizing and its gradient from the surface to deep in the interior, we set sizing parameters on the entire mesh and perform mesh adaptation using the SMS. The resulting mesh is smoothed and optimized, and the final product is then partitioned and ghost layers are added to it by ParFUM to facilitate communication and smoothing. For partitioning, we make use of either METIS- or ParMETIS-based[16, 17] partitioning provided by ParFUM[18], or our novel geometric partitioner (see Section 6). The new mesh is then ready for the solution transfer process.

5.3 Solution Data Transfer

We perform parallel solution data transfer using a conservative volume-weighted averaging strategy. Conservative volume-data transfer has been previously investigated in the literature, for example [19]. Our strategy begins by extruding the surface of the old mesh just enough to encapsulate the boundary of the new mesh. Cells generated by extrusion are assigned solution values from the cells whose faces were extruded to create them. Next, the new mesh is superimposed on the old mesh, and we determine which elements of the two meshes intersect. This is accomplished by using the parallel Charm++ Collision Detection library[20]. This library can be configured to determine lists of potentially colliding elements quite efficiently, carrying the associated element data to the new mesh in the process. These lists each contain entries for both source (old mesh) and destination (new mesh) elements. Each pair of source and destination elements is examined to determine their overlapping volume. The solution on the old element is then weighted by the percentage of the new element's volume that is overlapped, and this portion of the solution is accumulated on the new element. Preliminary tests verify that the solution transfer scheme is conservative to machine precision, and the accuracy is acceptable for this first order simulation. Further work will be devoted to detailed error quantification.

5.4 Future Directions in Remeshing

Efforts to parallelize the bulk of the remeshing phase are on-going. Approaches are selected based on how much the mesh quality has degraded. For example, a mesh with high surface quality but low volume quality is improved by remeshing the volume of each partition in parallel. This approach relies on the use of the geometric partitioner discussed in Section 6 to create well-formed partitions that the SMS can accept as valid meshes. This approach is complete and full integration of the new partitioner is underway.

Another approach we are taking to relieve the remeshing bottleneck addresses the problem of poor surface mesh quality. There is no mechanism to pass a mesh to the SMS in parallel and improve the surface. This phase will proceed as it currently does in *Rocrem*: the mesh is serialized, the surface is improved, and a new volume mesh is generated. However, one time-consuming phase involves the adaptation of the new volume mesh to a desired sizing. We intend to partition the mesh and perform this expensive step in parallel.

6 Parallel Hybrid Mesh Partitioner

As mesh-based simulations become larger and larger, the need for scalable, parallel mesh partitioners increases. Generally, mesh partitioners fall into two categories: topological and geometric. Each category has advantages and disadvantages relative to the other [21]. For instance, geometric partitioners can create disjoint partitions on non-convex domains. On the other hand, topological partitioners can create partitions with many neighboring partitions.

We seek to combine the advantages of each by developing a hybrid partitioner. The idea is to compute a coarse topological partitioning of an irregular mesh, segmenting large portions of the mesh that a geometric partitioner would have trouble partitioning into simpler domains. This coarse partitioning is then refined by a geometric partitioner.

At present, we have implemented in parallel a nested dissection geometric partitioner [22]. It has proven scalable for meshes with tens of millions of cells on tens of thousands of compute nodes.

A novel topological partitioner is under development. The partitioner finds separators in the medial surface of the domain. The medial surface is computed in parallel with only a simple partitioning (Figure 6). Once computed, the medial surface is walked to find edges that bound large features. These features

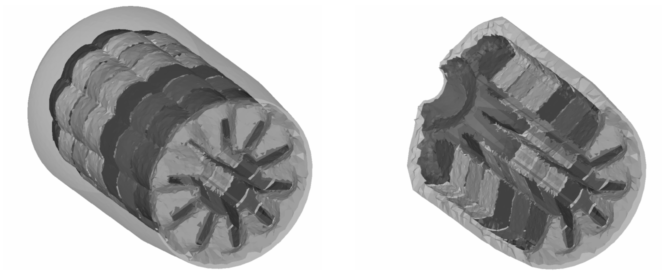

Fig. 6. Medial surface computed on 8 processors.

are broken off of the mesh and partitioned geometrically. Preliminary results of the hybrid partitioner show regular partitions with a bounded number of neighbors and no disjoint partitions (Figure 7).

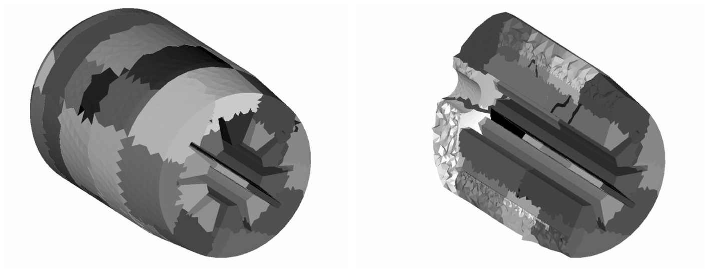

Fig. 7. Hybrid partition example.

7 Results

In this section, we present *Rocstar* simulations of two different rockets whose propellant is of significant geometrical complexity, a two-meter long test motor called StarAft and the 30-meter long Space Shuttle Reusable Solid Rocket Motor (RSRM).

Figure 8 shows the outer surface of the fluid domain in the StarAft rocket at four different simulation times during the test firing. The colors indicate the mesh partitions produced by METIS. The initial domain has a 6-pointed star-shaped cross section that tapers to a cylindrical profile near the head end (left hand side of Figure 8) and an inert nozzle at the aft end. As the propellant burns back to the cylindrical case, the fluid domain expands to fill the new fluid volume. When the propellant has burned completely, the fluid volume is over four times larger than it was initially. The number of

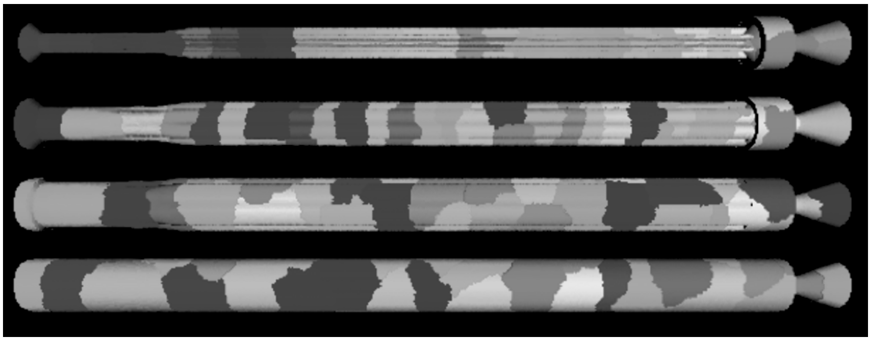

Fig. 8. Internal fluid domain of StarAft at (from top) 0.1, 3.4, 6.8, and 10.2 seconds.

tetrahedral elements increases from about 500K to as many as 1.7M. These computations were performed on up to 96 CPUs on various platforms. We used the Slow-timescale Acceleration technique [23] to speed up the burn-back rate by a factor Z from 64 to 1000. This reduces the number of time steps and mesh smoothings (performed every fluid time step) by a factor of Z, and therefore reduces by Z the ratio of the wall clock time for these operations compared to remeshing, whose invocation frequency and wall clock time are not affected by the acceleration technique. For $Z = 64$, the error in the fluid solution introduced by the acceleration is less than 2 percent; the percentages of wall clock time usage are: fluid solver 36, smoothing 41, and remeshing 23.

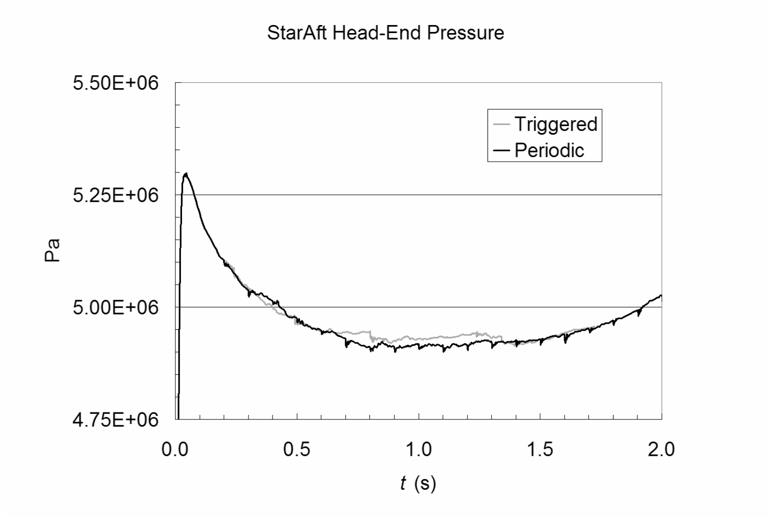

Fig. 9. StarAft pressure history for two different remeshing criteria.

Figure 9 shows the gas pressure at a particular location in the StarAft rocket for two simulations that differ only in their criterion for remeshing. The curve labeled "Periodic" (black) derives from a run in which remeshing was applied at 0.1 s intervals. One can see that each remeshing introduces a small (less than 0.3 percent amplitude) transient in the pressure history that dies away in about 0.02 s.

In the simulation corresponding to the curve labeled "Triggered" (pink), remeshing was triggered only when the time step computed by the fluid solver fell below a threshold value, presumably indicating the presence of sliver elements. Note the elevated pressures for times between 0.6 s and 1.3 s for this run. These elevated values occur because numerical errors in the solution increase as the mesh expands along with the fluid domain, reducing the spatial resolution. Whenever a new mesh is generated, additional interior nodes are inserted automatically to help preserve the initial local mesh spacing. Thus, the more frequent remeshings which occur in the periodically remeshed run better preserve both solution accuracy and mesh quality, although at a somewhat higher computational cost. In practice, we have found that periodic remeshing can also help prevent occasional simulation crashes due to large nodal displacements induced by the mesh smoother while attempting to improve a mesh of marginal quality.

Figure 10 shows the grid near the aft end of the StarAft rocket at four different times. The colors again indicate the partitioning pattern. The image sequence shows how the geometric features evolve. By 3.4 s, the concave star grooves have become sharp ridges, and the convex star tips have become several times wider. By 6.8 s, the annular surface at the aft end of the star has vanished, and part of the convex star tips have reached the cylindrical case. By 10.2 s, all features associated with the star have burned away.

When geometric features are about to change, nearby surface triangles often become very small, and remeshing is more frequently triggered by the fluid time step criterion. For this reason, our "periodically" remeshed runs enforce remeshing at both regular intervals and the fluid time step threshold.

Figure 11 indicates both the mesh quality and the frequency of remeshing in a run with only the fluid time step threshold (no periodic remeshing). At 0.1 s intervals, we computed the value of the maximum dihedral angle. The black dots represent times at which remeshing occurred, and line segments connecting the dots correspond to time intervals during which no remeshing was triggered. Some remeshing periods are so short that only one data point was used for that period. We can see that each remeshing yielded a mesh with a maximum dihedral angle of about 155 degrees. The maximum dihedral angle subsequently deteriorated to somewhere in the range of 170 to nearly 180 degrees before the next remesh was triggered. Note that frequency of remeshing varied from several tenths of a second at early times, to very often for times between about 6 to 8 seconds, and then several tenths of a second again towards burn out. This observation agrees with the fact that geometric features changed frequently between 6 s and 8 s, as mentioned above. Note

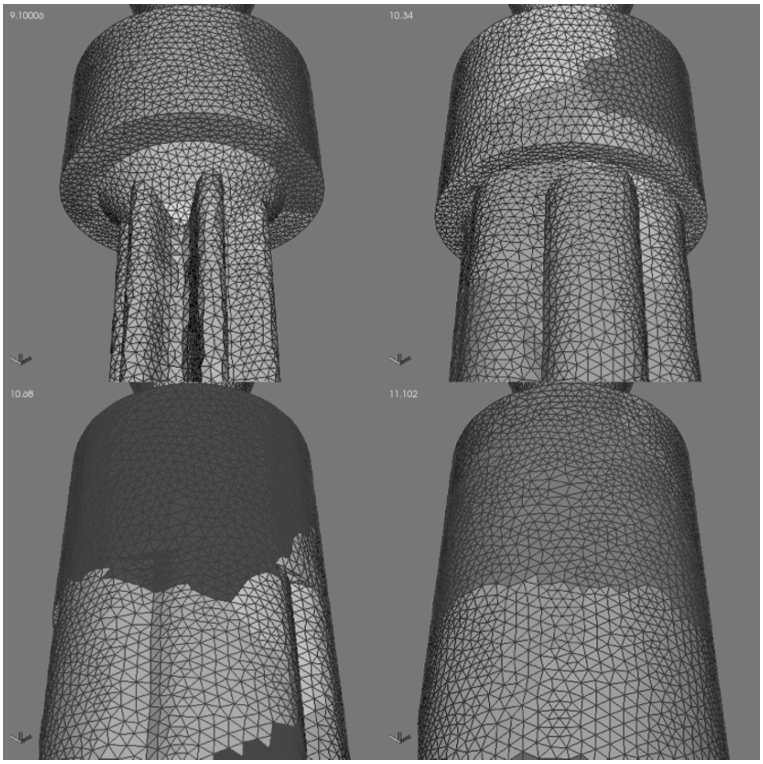

Fig. 10. Mesh of StarAft near the aft end at 0.1, 3.4, 6.8, and 10.2 seconds.

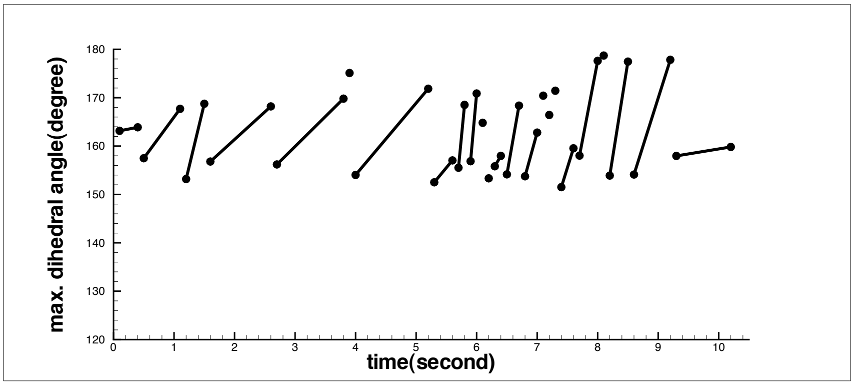

Fig. 11. Maximum dihedral angle of StarAft. Data points are connected together for each interval between remeshing.

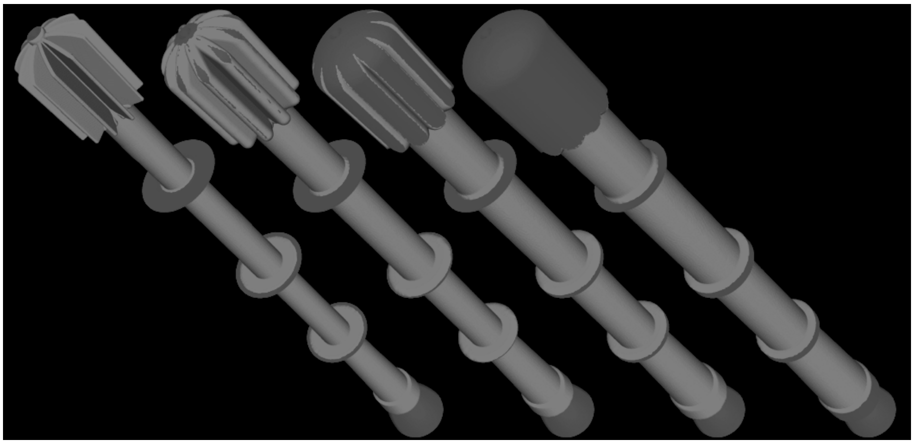

Fig. 12. Burning (red) and non-burning surfaces in the RSRM at 0.1 s, 22 s, 50 s, and 61 s.

that the remeshing interval was also very short near 4 s, when the star grooves became sharp ridges.

Next, we simulated near-complete propellant burn-out in the RSRM on up to 256 processors. Figure 12 illustrates the progression of propellant burn-back. Shortly after ignition (leftmost image), burning propellant covers most of the surface of the fluid domain, except for a hole for the igniter at the head end (upper left), the nozzle (lower right) and annular rings in the three joint slots where a coating inhibits burning. At 22 s, the star tips first reach the rocket case, which is essentially cylindrical with an ellipsoidal dome at the head end. At subsequent times, the star tips broaden and then merge. The burning surface in the joint slots expand radially until they reach the case, and then slide along the rocket axis. By 111 s, the propellant is entirely gone.

Figure 13 shows geometrical details and the surface mesh at four different times. The color scale corresponds to the gas pressure. Shortly after ignition, at 0.2 s, the pressure is still low, since the propellant has just begun to burn. There are several triangles across the concave grooves in the star-shaped region, and the convex star tips are thin. By 7.1 s, the concave grooves are just beginning to merge. The merging of features occurs over the course of several remeshings. During each remeshing, all triangles below a certain size are eliminated, but this process may remove only part of the corresponding geometrical feature; the rest of the feature remains in the model until the remaining triangles become small enough to be eliminated during a subsequent remeshing. By 15.1 s, the concave grooves have become ridges, and the wider bumps at the bases of the star tips are about to merge. By 23.1 s, the bumps have merged and the star tips are flattening where they have reached the case.

The mesh in this simulation had 1.7 M tetrahedra initially, and 4.1 M by 23.1 s. Remeshing was invoked every 0.5 s, and also triggered whenever

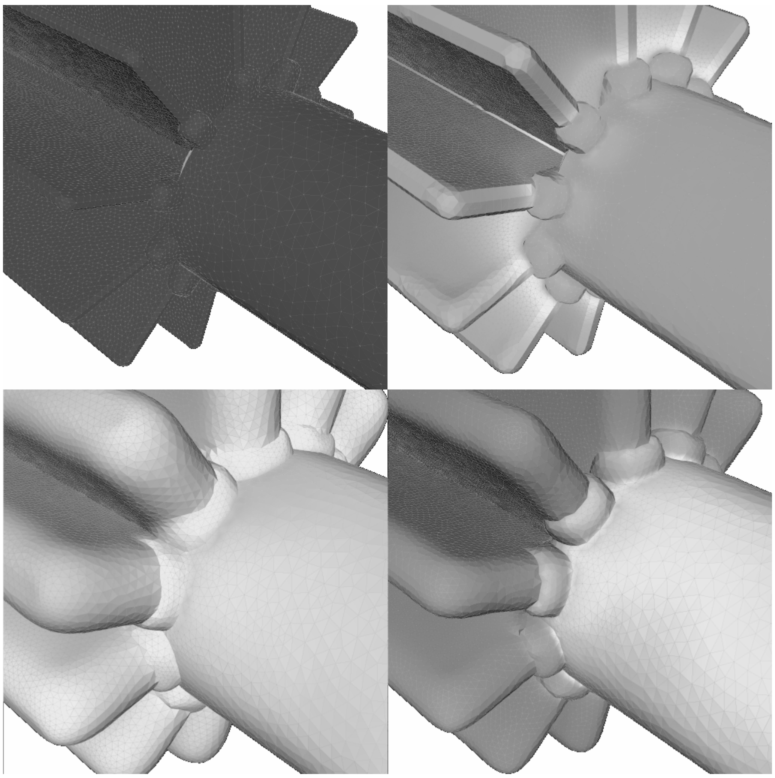

Fig. 13. Aft end of the star-shaped region in the RSRM. Simulation times are (clockwise from upper left) 0.1 s, 7.1 s, 15.1 s, and 23.1 s.

the fluid time step dropped below 1 microsecond. There were roughly 50,000 fluid steps between remeshings. The Slow-timescale Acceleration factor Z was 64. Each entire remeshing procedure takes about 46 minutes and produces a mesh with a maximum dihedral angle near 155 degrees. For a run that is not accelerated ($Z = 1$), in which we smooth the mesh (approximately 3 M elements) every 3 fluid time steps, the percentages of wall clock time usage are: fluid solver 70, smoothing 27, and remeshing 3. Accelerating the run by a large factor essentially eliminates the fluid and smoothing times (even if we smooth every fluid step), and therefore serial remeshing limits the maximum possible speedup (through Slow-timescale Acceleration) to a factor of about 33.

8 Conclusion and Discussion

The strategy for modifying body-fitted meshes described in this paper enables our integrated simulation package to obtain accurate numerical solutions for many classes of physical problems with moving domain boundaries. For the solid propellant rockets studied here, the nominal burn rate is quite slow compared to the Courant limit on the explicit time step that can be used in our unstructured-mesh fluid dynamics solver. As a result, even though we perform some remeshing stages using a sequential commercial package, these operations do not dominate the total simulation wall clock time. In contrast, frequent application of a sequential mesh smoother in a parallel manner is essential to minimizing the total computational cost of both smoothing and remeshing.

For problems with more rapid domain boundary motions, including those we produced by accelerating the burn-back times in our rockets, serial remeshing can become a bottleneck. Parallelizing all stages of mesh generation is a much more difficult problem than smoothing mesh partitions concurrently. Although there are parallel versions of some meshing tools, including Simmetrix's, we believe that a parallel remeshing strategy based on concurrent application of serial packages has important advantages, including ease of integration of alternative serial meshing tools.

Our experience with large-scale simulations with highly evolving domains suggests some future directions in meshing research. One research area is software integration with physics solvers and among meshing packages. Although many meshing packages can successfully solve the problems for which they were designed, integrating diverse packages into a large simulation remains difficult and time consuming. Various obstacles prevented us from using the native parallel versions of Mesquite and Simmetrix (which did not exist when we developed Rocstar in its present form). For example, Mesquite has an API that conforms to the TSTT specification, but Simmetrix does not. Another obstacle involves the discrete geometry of partitioned meshes. The mesh in a large-scale simulation is already partitioned when remeshing is needed, and the domain geometry can be defined only by a collection of surface patches distributed among a large number of processors. The surface patches carry additional attributes, such as boundary conditions, whose values must remain associated with the appropriate faces when the domain is remeshed and repartitioned.

Additional research is needed to improve the efficiency of mesh smoothing, which for evolving geometries can consume a significant fraction of the run time, even when performed concurrently. Our simulations would benefit from the use of a mesh quality measure that is both quick to evaluate and takes into account the element size distribution of the existing mesh. The smoothed mesh should be as similar as possible to the existing mesh to help prevent sudden large motions of nodes that could cause solution instability in the physics solvers. This constraint and the fact that the existing mesh is usually of reasonably good quality might be used to improve the rate of convergence to the optimal smoothed mesh configuration.

Acknowledgements

This work is supported by the Center for Simulation of Advanced Rockets (CSAR) at the University of Illinois at Urbana-Champaign. The CSAR research program is supported by the US Department of Energy through the University of California under subcontract B523819. The authors thank John Norris for the visualization software *Rocketeer*, Mu Sun for generating pictures and movies from simulation results, Mark Brandyberry for helping with initial mesh generation, Gengbin Zheng for the orchestration module *Rocman*, and Professor Michael Heath for general direction of the project.

References

1. Owen S J, White D R (2003) Mesh-based geometry. Int. J. Numer. Meth. Engrg. 58:375–395.
2. Baker T J (2001) Mesh movement and metamorphosis. In Proc. 10th Int. Meshing Roundtable, Newport Beach, California, pp. 387–395.
3. Folwell N, Knupp P, Brewer M (2003) Increasing TAU3P abort-time via mesh quality improvement. In Proc. 12th Int. Meshing Roundtable, Santa Fe, New Mexico, pp. 379–390.
4. Wan J, Kocak S, Shephard M S (2004) Automated adaptive forming simulations. In Proc. 13th Int. Meshing Roundtable, Williamsburg, Virginia, pp. 323–334.
5. Cardoze D E, Miller G L, Olah M, Phillips T (2004) A bezier-based moving mesh framework for simulation with elastic membranes. In Proc. 14th Int. Meshing Roundtable, San Diego, California, pp. 71–79.
6. Dheeravongkit A, Shimada K (2004) Inverse pre-deformation of finite element mesh for large deformation analysis. In Proc. 14th Int. Meshing Roundtable, San Diego, California, pp. 81–94.
7. Giraud-Moreau L, Borouchaki H, Cherouat A (2006) A remeshing procedure for numerical simulation of forming processess in three dimensions. In Proc. 16th Int. Meshing Roundtable, Birmingham, Alabama, pp. 127–144.
8. Dick W, Fiedler R A, Heath M T (2006) Building rocstar: Simulation science for solid propellant rocket motors. In 42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, AIAA-2006-4590.
9. Huang C, Lawlor O, Kalé L V (2003) Adaptive MPI. In Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958, College Station, Texas, pp. 306–322.
10. Jiao X (2007) Face offsetting: a unified framework for explicit moving interfaces. J. Comput. Phys. 220:612–625.
11. Baker B, Copson E (1950) The Mathematical Theory of Huygens' Principle. Clarendon Press, Oxford.
12. Jiao X, Alexander P (2005) Parallel feature-preserving mesh smoothing. In Computational Science and Its Applications - ICCSA 2005, vol. 3483/2005 of *Lecture Notes in Computer Science*, pp. 1180–1189.
13. Knupp P (2006) Mesh quality improvement for SciDAC applications. Journal of Physics: Conference Series 46:458–462.

14. Brewer M, Diachin L, Knupp P, Leurent T, Melander D (2003) The Mesquite mesh quality improvement toolkit. In Proc. 12th Int. Meshing Roundtable, Santa Fe, New Mexico, pp. 239–250.
15. Freitag L, Knupp P, Leurent T, Melander D (2002) MESQUITE design: Issues in the development of a mesh quality improvement toolkit. In Proc. 8th Int. Conf. Numer. Grid Gen. in Comput. Field Sim., Honolulu, Hawaii, pp. 159–168.
16. Karypis G, Kumar V (1998) Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 48:96 – 129.
17. Karypis G, Kumar V (1997) A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm. In Proc. of the 8th SIAM conference on Parallel Processing for Scientific Computing.
18. Lawlor O, Chakravorty S, Wilmarth T, Choudhury N, Dooley I, Zheng G, Kale L (2006) Parfum: A parallel framework for unstructured meshes for scalable dynamic physics applications. Engineering with Computers 22 3-4:215–235.
19. Grandy J (1999) Conservative remapping and region overlays by intersecting arbitrary polyhedra. J. Comput. Phys. 148 2:433–466.
20. Lawlor O S, Kalé L V (2002) A voxel-based parallel collision detection algorithm. In Proc. Int. Conf. in Supercomputing, ACM Press, pp. 285–293.
21. Hendrickson B (1998) Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract). In IRREGULAR '98: Proc. 5th Int. Symp. on Solving Irregularly Structured Problems in Parallel, Springer-Verlag, London, UK, ISBN 3-540-64809-7, pp. 218–225.
22. Vidwans A (1999) A Framework for Grid Solvers. Ph.D. thesis, Univeristy of Illinois, Urbana-Champaign, Urbana, IL.
23. Haselbacher A, Najjar F M, Massa L, Moser R (2006) Enabling three-dimensional unsteady srm burn-out computations by slow-time acceleration. In 42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, AIAA-2006-4591.