PARALLEL MESH GENERATION AND ADAPTATION FOR CAD GEOMETRIES

Saurabh Tendulkar

Senior Software Engineer, Simmetrix Inc., USA

Mark Beall

President, Simmetrix Inc., USA

Mark S. Shephard

Professor, Rensselaer Polytechnic Institute, USA

Kenneth Jansen

Professor, University of Colorado, USA

THEME

Emerging Issues - High Performance Computing

KEYWORDS

high performance computing, parallel meshing, mesh adaptivity

SUMMARY

Large scale parallel simulations are dealing with meshes with millions to billions of elements. As the numbers of elements in these meshes continue to increase, it is becoming necessary to deal with the generation and control of the mesh in parallel. This paper overviews a set of procedures that can accept a CAD model as input and generate initial meshes in parallel with 100's of millions of elements. Adaptive control of the mesh is supported by parallel mesh modification procedures that can perform parallel anisotropic mesh adaptation, including maintaining boundary layer meshes, on meshes with billions of elements. These procedures have been integrated with multiple parallel finite element analysis codes to support the execution of large-scale simulations where all steps in the process are executed in parallel.

PARALLEL MESH GENERATION AND ADAPTATION FOR CAD GEOMETRIES

1: Introduction

Performing accurate simulations in many areas, such as fluid flow or electromagnetics, requires meshes with many millions of elements. In such cases the only practical means to perform the analysis is the application of parallel computing methods. As mesh sizes continue to grow, and the level of geometric complexity of the domains increases, the application of serial mesh generation becomes a bottleneck in the process. To eliminate this bottleneck we have developed parallel mesh generation and adaptation procedures that operate in both multithreaded and distributed parallel environments.

The ability to perform any meshing operations in parallel requires a mechanism to understand how the mesh is distributed onto the parallel computer. The mechanism of a mesh partition, discussed in Section 2, is used for this purpose since it provides the ability to support both distributed memory parallelism based on message massing and single address space parallelism based on multithreading.

A key challenge to the construction of a truly automatic parallel mesh generator is devising a means to distribute the mesh generation tasks. Requiring the user to decompose the geometry does not eliminate the performance bottleneck. The techniques outlined in section 3 automatically decompose the work to be done at each step in the meshing process, so that the entire meshing process can be automatically executed in parallel starting from a CAD model.

To support the ability for users to apply adaptive analysis, the parallel mesh adaptation procedure described in Section 4 has also been developed. The input to the mesh adaptation is an anisotropic mesh metric field associated with the existing distributed mesh that is constructed using error indications based on the current analysis results (e.g., using the Hessian of the solution field (Zhou, et al., 2010a)). Parallel mesh adaptation is executed using mesh modification operations that easily include local solution transfer which is important to maintain the convergence of the non-linear iterations in the next solve step. Recent developments in the mesh adaptivity procedures include the ability to perform adaptivity of boundary layers for CFD simulations.

All these procedures operate with a functional interface to the CAD representation of the domain (Beall, et al., 2004). To support those situations where the level of detail of the CAD model is high, procedures to interact with the geometry in a parallel, distributed manner are also being developed as discussed in section 5.

Example application of the parallel mesh generation and adaptation processes are included in the sections that overview them.

Figure 2: Simple model meshed in parallel. The coloring. indicates which process meshed each model face.

2: Partitioned Mesh

In order to do parallel operation on a mesh, it is necessary to be able to distribute the mesh in parallel and have well defined parallel operations on that mesh. MeshSim (Simmetrix 2010) supports a partitioned mesh representation that allows distributing the mesh over the memories of the nodes of a parallel computer. Each local portion of a mesh is called a *part* and the boundaries of a part keep information that relates the entities on the boundary to their corresponding entities on the boundaries of other parts (Figure 1).

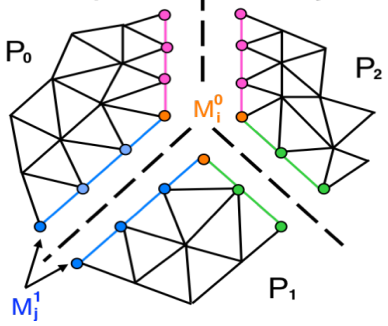
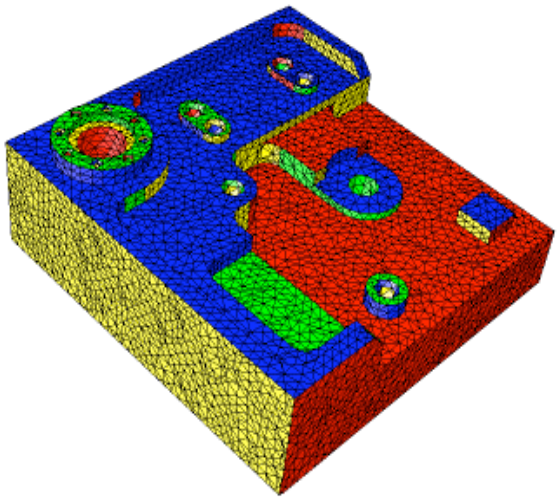


Figure 1: A mesh partitioned into three parts

All of the interprocessor operations are supported such as part-to-part communications and migration of mesh entities between parts. At the level of a single part, the mesh is equivalent to a serial mesh, so code that does completely local operations is the same as serial code. The availability of the part-to-part communications and mesh entity migration allows the development of parallel mesh generation and adaptation procedures that are not influenced by the manner in which the mesh is distributed. See Seol and Shephard (2006) for more information.

The distributed mesh structures are easily interfaced with parallel load balancing procedures (e.g., ParMETIS and Zoltan (2010)) to modify the mesh partitions as needed for the analysis program, including changing the numbers of processors to be used in the simulation. For example, we have generated a mesh on 64 cores and dynamically load balanced it to 16K cores for the analysis process (Zhou, et al., 1010).



3: Parallel Mesh Generation

In surface meshing, the model faces provide a natural decomposition of the work to be done which is divided up between processors. Each processor meshes multiple faces to maintain overall load balance (Figure 2). Which faces are meshed by a processor is

automatically determined by the mesh generator. Although this process does not guarantee a good load balance of the work – it is certainly possible that there are fewer model faces than processes or that a single model face may require more work than others – in practice there are generally many more model faces than processes and additional work to optimize the work balance here does not have a high payoff.

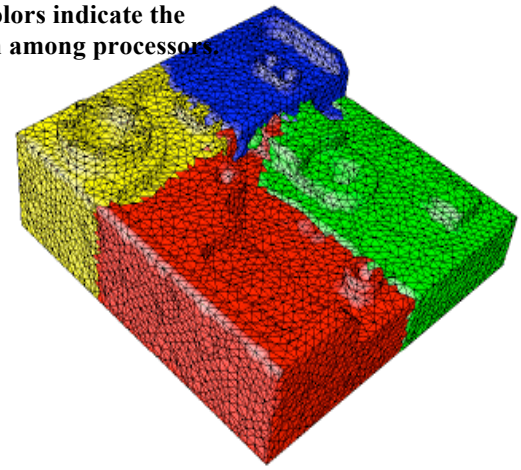


Figure 3: Final distribution of the mesh after volume meshing. The colors indicate the partitioning of the mesh among processors.

For volume meshing a spatial decomposition of the domain that takes into consideration the work to be done is used (de Cougny and Shephard, 1999). The initial decomposition targets getting a balanced amount of local mesh generation per part. The volume mesh for the portion of each part entirely local to that part is performed in parallel. A repartitioning is performed to get remaining unmeshed portions, which are the portions that were between parts in the initial partition, as completely onto single parts as possible and the process repeated. After a second step of that there are only isolated regions remaining unmeshed. These are completed after a final repartitioning. Figure 3 shows a simple example meshed on 4 processors.

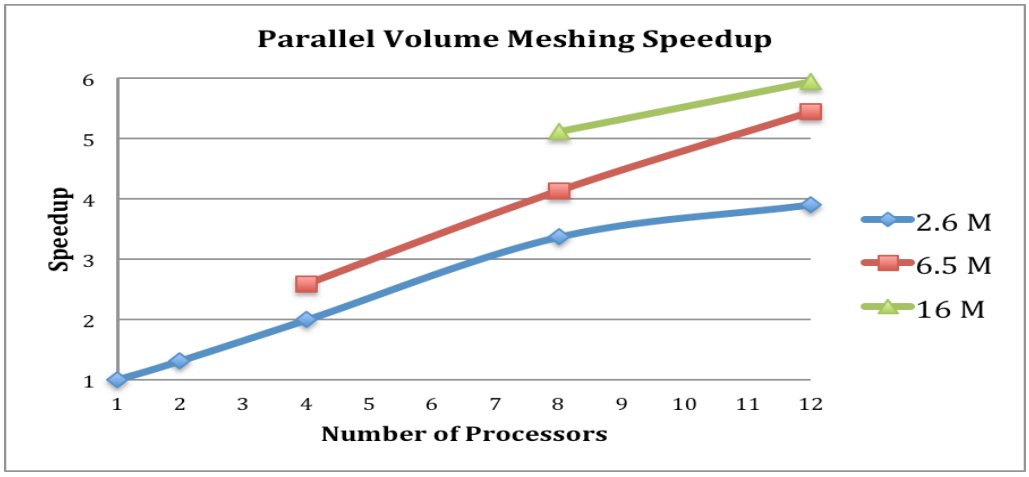


Figure 4: Parallel volume meshing speedup on up to 12 processors.

Obtaining good scaling with volume mesh generation is a difficult task as the process does not have a constant structure as does, for example, a linear solver. Figure 4 show the speedup of the entire volume meshing process for 3 different

mesh sizes run on a cluster of 6 dual processor 2.0 Ghz Opteron servers connected with gigabit ethernet. The smallest problem could be run on a single processor and thus that single processor performance is used to normalize all of the runs. As is expected, the strong scaling results drop as the number of elements on each processor decreases. However, for the generation of very large meshes, we are concerned more with being able to generate them in a reasonable amount of time rather than with theoretical scaling.

Figure 5 shows the results of a number of parallel meshing runs on a cluster consisting of 40 IBM x3755 servers with four dual-core 2.8 Ghz Opteron processors and an Infiniband interconnect. As can be seen, meshes of over 300 million elements were generated in several minutes.

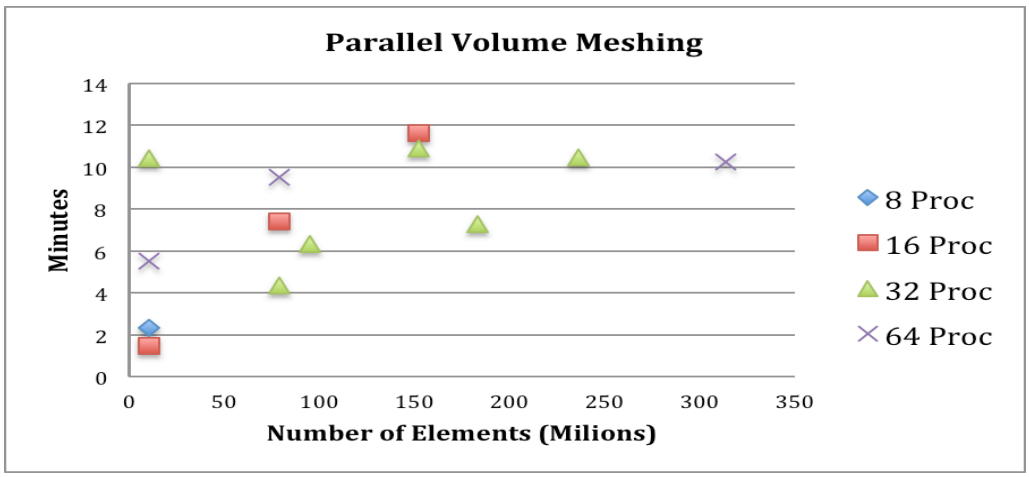


Figure 5: Parallel volume meshing results on up to 64 processors.

Figure 6 shows a 1/8th portion of a 180 million element mesh of an linear accelerator model generated in 8 minutes on 64 processors. The close-ups indicate a bit more of the object's shape and existence of interior structure.

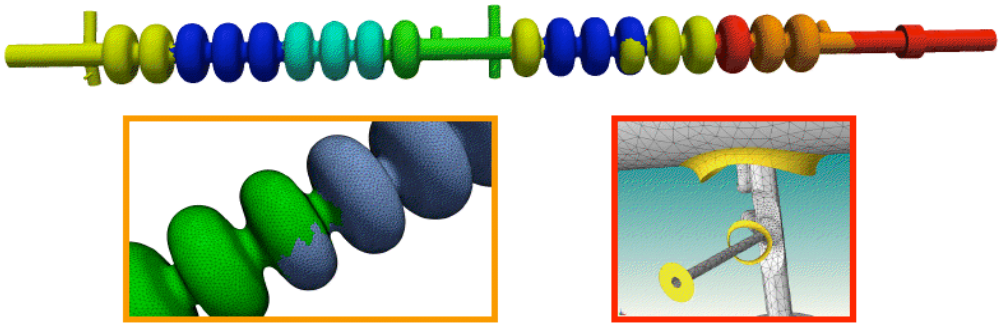


Figure 6: Image shows 1/8th of a 180 million element mesh generated on 64 processors.

In addition to distributed parallel meshing, we have also used the same ideas to provide multithreaded parallel volume meshing. This turns out to be an even more challenging problem as it has all of the same scaling issues associated with distributed parallel meshing, but adds additional issues related to avoiding contention for shared resources and, very likely, memory bandwidth issues. Figure 7 shows typical current results for multithreaded volume meshing with speedups approaching about 1.5 on 4 cores. We are currently working on improving the efficiency of this process with the goal of a speedup of 1.5 on 2 cores and above 2.0 on 4 cores.

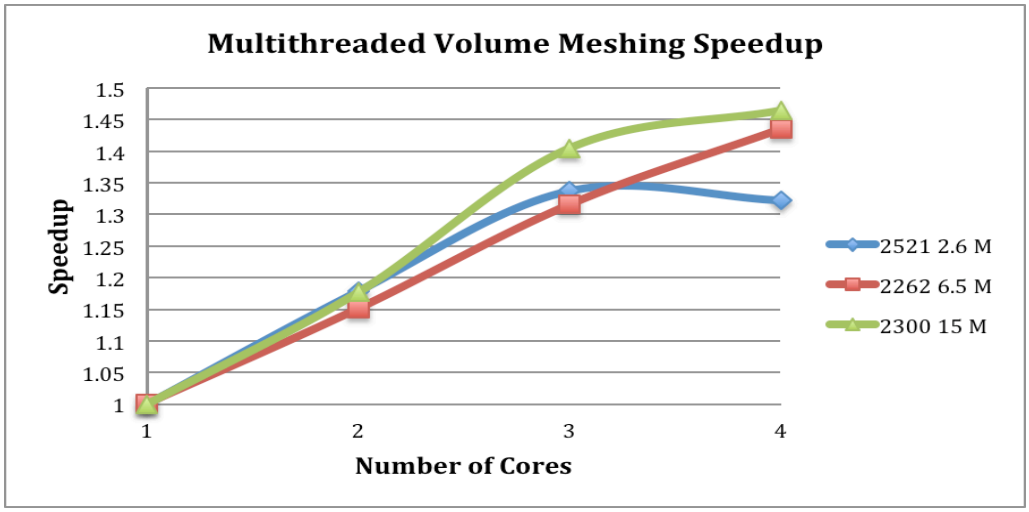


Figure 7: Multithreaded volume meshing speedup

4: Parallel Mesh Adaptivity

Parallel mesh adaptation is critical to performing adaptive analyses on parallel computers. Without this capability it would be necessary to migrate the entire mesh to a single processor, adapt it and then repartition it again. Apart from this being very inefficient, it is unlikely that a single node of a parallel computer would have enough memory to hold the entire mesh for a large simulation.

The two common approaches to mesh adaptation of conforming unstructured finite element meshes are remeshing and mesh modification. Although remeshing would have the appeal to being quite general, it has poor computational performance and in the case where solution fields have to be transferred between meshes, yields a transferred field that has poor local conservation properties. The alternative is to perform local mesh modification operations, which if fully generalized, including accounting for interactions with the CAD representation of the domain, are as flexible as remeshing. The mesh adaptation procedures presented here are based on generalized mesh modification (Li, et al., 2005).

Given a geometric domain, a current mesh and a mesh size field defined in a piecewise manner over that mesh, the mesh modification steps are executed to convert the given mesh into one that satisfies the given size field. The mesh modification operations supported include; entity splits for refining the mesh, entity collapse to coarsen the mesh, swap operations to improve the shape of elements and various compound operations that couple multiple single step modifications to provide the desired flexibility. To ensure the adapted mesh will properly satisfy the required size field, the mesh modification algorithms are carefully constructed and executed in three stages of (i) mesh coarsening to eliminate short edges, (ii) shape correction to provide better structured mesh for the third stage, and (iii) intelligent refinement that includes operations for ensuring the proper geometric approximation of the mesh and proper shape to match the mesh size field.

Since the mesh modification operations only affect small portions of the mesh, most of the mesh modification operations can be executed in parallel directly using on-processor operations. However, not all operations can be properly executed in this manner and a parallel algorithm is needed to ensure that all operations can be carried out such that the resulting mesh properly fits together and all mesh modifications required to satisfy the requested mesh size field have been executed. Since all refinement operations can be understood at the level of single elements they can always be performed in parallel on part. The only parallel operation required is a round of communication to be up-date the interporcesser communication links between the new mesh entities introduced on part boundaries. In the case of collapse and swap operations, the mesh entities involved in that modification may be resident on multiple parts. For example the left image of Figure 8 shows the elements involved with a 2D edge collapse that are distributed over four parts. In these cases, the parallel execution of this operation first determines the mesh entities involved with the modification. Those mesh entities are then migrated to a single part (middle image of Figure 8) and the mesh modification executed on that part (right image of Figure 8).

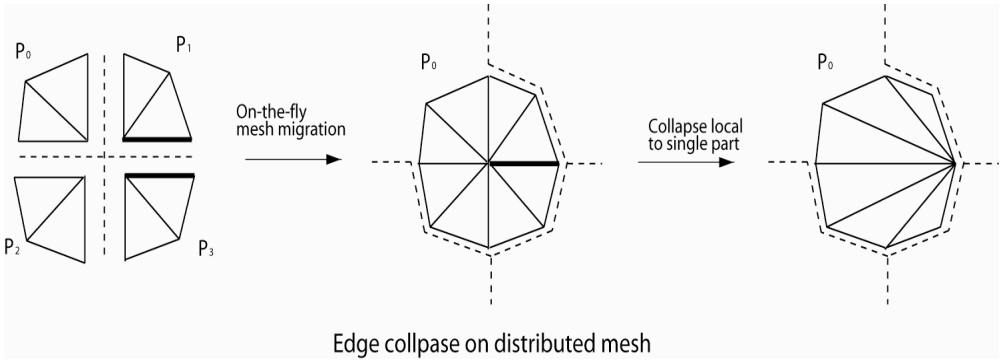


Figure 8: Steps on the parallel execution of a collapse operation.

In the execution of an adaptive analysis, error estimators are used to set the next mesh size field. By its nature, how the mesh size fields changes over the domain is non-uniform where portions of the domain will require a substantial amount of refinement in the mesh and others coarsening, while some portions of the domain may require little or no change in the mesh size.

When doing the adaptive mesh modifications it is necessary to consider both load and memory balance as well as ending up with a final partitioning of the mesh that is suitable for the next solver step. At the same time it is necessary to minimize communication. Thus the steps use to execute this process are (de Cougny and Shephard, 1999a):

- Estimate the overall work to be done on each part based on the new mesh size field. This associates weights with the highest dimension mesh entities.
- Use those weights to do predictive load balancing which will migrate mesh entities such that the work to be done during mesh adaptation is fairly balanced and the result of the adaptivity will give good memory balance.
- Execute the parallel mesh modification operations.
- Perform a final load balancing to provide an optimal mesh partition for the next analysis step.

The resulting mesh adaptation procedures have been applied to a wide variety of applications being interfaces to both commercial and research finite element and finite volume codes. Figure 9 shows an adaptively refined mesh for an interesting two-phase flow problem (Galimov, et al., 2010).

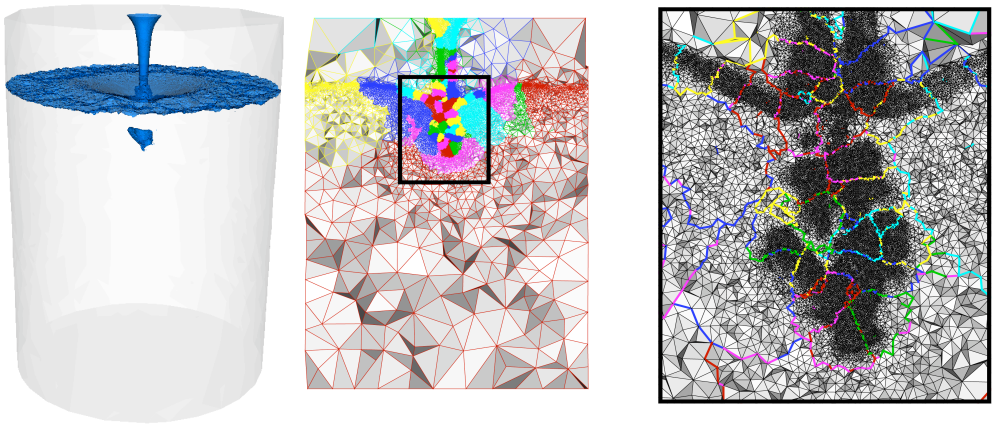


Figure 9: Simulation of air entrapment of a plunging jet. 13 M elements in 128 parts, colors indicate different parts.

Since many application areas of interest are characterized by directionally dominate solution features, it is highly desirable to be able to support anisotropic mesh configurations. In such cases a mesh metric tensor can be used where the eigenvalues of the 3x3 metric matrix correspond to the desired

edge lengths while the eigenvectors define the principal directions for those edges. The Hessian of a solution parameter of interest is one common means used in the definition of the mesh metric field over a given mesh (Alauzet, et al. 2006; Zhou et al., 2010a). The mesh adaptation procedures (Simmetrix 2010) fully support the mesh modification to an anisotropic mesh metric field.

Figure 10: Edge split in the boundary layer.

In cases where highly anisotropic meshes are needed, it is often desirable that a carefully structured set of elements are constructed need selected portions of the domain boundary. MeshSim (Simmetrix 2010) supports a full range of such boundary layer mesh generation capabilities.

To be able to properly maintain these structured boundary layers during mesh adaptation, the mesh modification procedures have been extended to decompose the mesh modifications in the direction normal to the boundary from the other two directions and ensure that the modification goes through the entire stack and is properly reflected in any transition elements and the interior mesh (Sahni, et al., 2008). Figure 10 demonstrate this for a simple edge split operation. The procedures also include adjustment to the height of the layers.

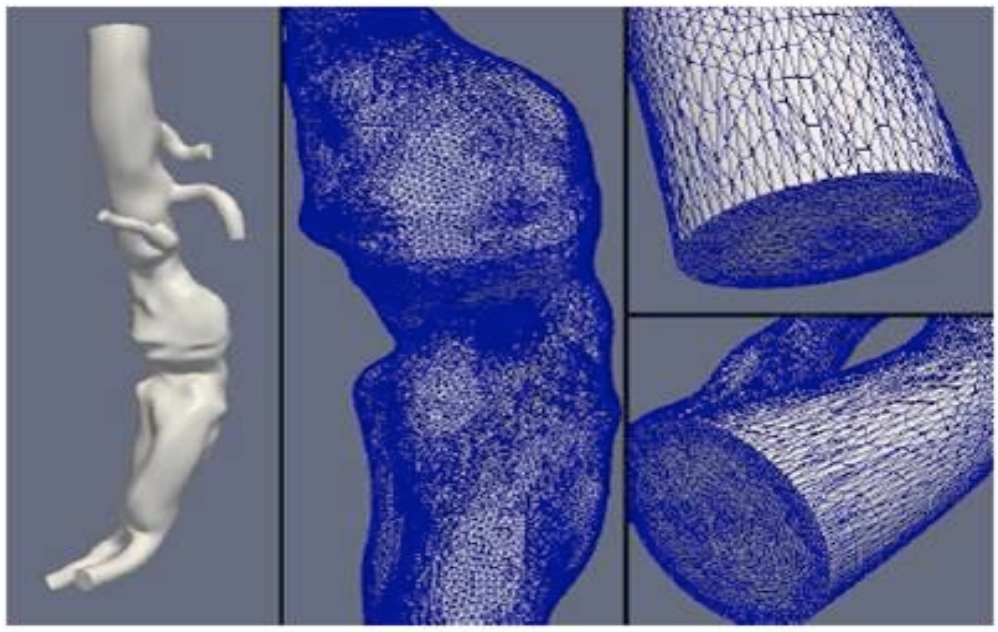


Figure 11: Anisotropically adapted boundary layer mesh example.

Figure 11 shows an example adapted boundary layer mesh used in the simulation of blood flow through an aortic aneurism (Zhou, et al. 2010a).

5: Distributed parallel geometry

For adaptive simulations on very large numbers of processors, keeping the CAD model geometry on each processor becomes problematic in terms of memory usage. To avoid this problem we are developing a distributed, parallel geometry representation. This will connect with the mesh adaptivity and load balancing routines to ensure that only the geometry needed on each processor is present and automatically migrate that geometry as needed during the adaptive analysis.

The basic idea behind distributing the model geometry is similar to a partitioned mesh, model entities can be migrated from processor to processor and maintain enough information to properly hook up with their adjacent entities when they are migrated. Figure 12 shows an example of a distributed model where the model geometry needed for part of the mesh is shown in grey along with the corresponding mesh part.

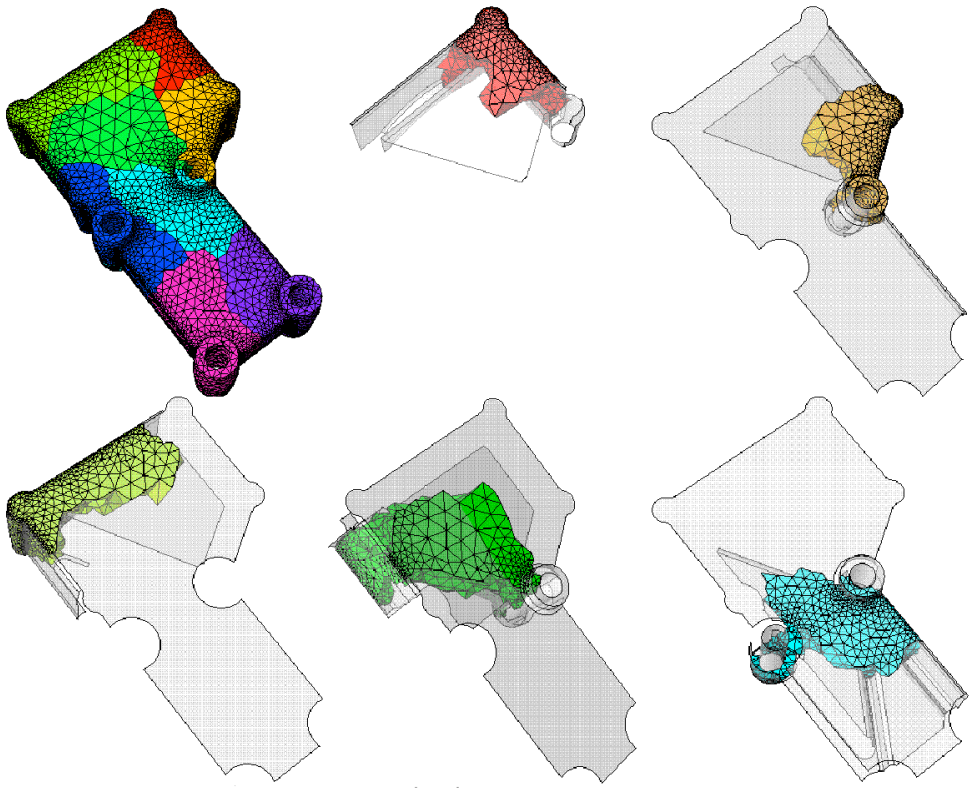


Figure 12: Distributed model geometry matching mesh partitioning

The specific geometric model entities that must be present on any part includes all those for which mesh entities on that part are associated. Although this forces duplication of model entities on multiple parts, as in the example shown in Figure 12, on a mesh with a large number of model entities and parts, there will need to be only a small fraction of the total geometry on each processor and many parts - those entirely interior to the model domain - will have none at all. Overall this will lead to substantial memory savings while maintaining the ability to interact with the CAD model in a scalable manner.

REFERENCES

Alauzet, F., Li, X., Seol, E.S. and Shephard, M.S. (2006) Parallel Anisotropic 3D Mesh Adaptation by Mesh Modification, *Eng.with Comp.*, 21(3):247-258.

Beall, M.W., Walsh, J. and Shephard, M.S (2004) "A comparison of techniques for geometry access related to mesh generation", *Eng. with Comp.*, 20(3):210-221.

de Cougny, H.L. and Shephard, M.S. (1999) "Parallel volume meshing using face removals and hierarchical repartitioning", *Comp. Meth. Appl. Mech. Engng.*, 174(3-4):275-298.

de Cougny, H.L. and Shephard, M.S. (1999a) "Parallel Refinement and Coarsening of Tetrahedral Meshes", *Int. J. Numer. Meth. Eng.,* 46:1101-1125.

Galimov, A.Y., Sahni, O., Lahey, Jr., R.T., Shephard, M.S., Drew, D.A. and Jansen, K.E. (2010) "Parallel Adaptive Simulation of a Plunging Liquid Jet", *Acta Mathematica Scientia*, 30B: 522-538.

Li, X., Shephard, M.S. and Beall, M.W. (2005) "3-D Anisotropic Mesh Adaptation by Mesh Modifications", *Comp. Meth. Appl. Mech. Engng*., 194(48-49):4915-4950.

Sahni, O., Jansen, K.E., Shephard, M.S., Taylor, C.A. and Beall, M.W. (2008) "Adaptive Boundary Layer Meshing for Viscous Flow Simulations" *Eng. with Comp.*, 24(3): 267-285.

Seol, E.S. and Shephard, M.S. (2006) "Efficient distributed mesh data structure for parallel automated adaptive analysis", *Eng. with Comp.*, 22(3-4):197-213.

Shephard, M.S., Flaherty, J.E., Jansen, K.E., Li, X., Luo, X.-J., Chevaugeon, N., Remacle, J.-F., Beall, M.W. and O'Bara, R.M. (2005) "Adaptive mesh generation for curved domains", *J. Applied Num. Math.*, 53(2-3)251-271.

Simmetrix (2010), http://www.simmetrix.com/

Zhou, M., Xie, T., Seol, S., Shephard, M.S.. Sahni, O. and Jansen, K.E. (2010) "Tools to Support Mesh Adaptation on Massively Parallel Computers", *Eng. with Comp.*, to appear.

Zhou, M., Sahni, O., Kim, H.J., Figueroa, C.A. Taylor, C.A. Shephard, M.S. and Jansen, K.E. (2010a) "Cardiovascular Flow Simulation at Extreme Scale, *Computational Mechanics*, 46:71-82.

Zoltan (2010) http://www.cs.sandia.gov/Zoltan/Zoltan_phil.html