INTEGRATION OF PARALLEL DATA REFINEMENT METHOD WITH ADVANCING FRONT LOCAL RECONNECTION MESH REFINEMENT SOFTWARE

Kevin Garner
Department of Computer Science
Old Dominion University
Norfolk, VA 23529, USA
kgarn006@odu.edu

Thomas Kennedy
Department of Computer Science
Old Dominion University
Norfolk, VA 23529, USA
tkennedy@cs.odu.edu

Nikos Chrisochoides
Department of Computer Science
Old Dominion University
Norfolk, VA 23529, USA
nikos@cs.odu.edu

Abstract

A procedure is presented for parallelizing an industrial-strength sequential mesh generator called Advancing Front Local Reconnection (AFLR) which has been under development for the last 25 years at the NSF/ERC center at Mississippi State University. AFLR is currently used at NASA (including NASA/LaRC) and other government agencies as well as in the aerospace industry such as Boeing. The parallelization procedure for AFLR presented in this paper is called Parallel Data Refinement (PDR) and it consists of the following steps: (i) use an octree data-decomposition scheme to break the original geometry into subdomains (octree leaves), (ii) refine each subdomain with the proper adjustments of its neighbors using the given refinement code, and (iii) combine all subdomain data into a single, conforming mesh. This approach is stable (i.e., guarantees the same mesh quality as the sequential mesh generation), it is robust (i.e., it can generate meshes for the same type of geometries AFLR can), and it can speed up execution time to a point where it can be 10 to 20 times faster on a 32-core node (PETTT Year 1 report sent to U.S. Department of Defense) (Kennedy, Chernikov, & Marcum, 2016). Finally, given earlier experience of an implementation of PDR with TetGen, a similar open source mesh generation code, the expectation is that one can modify about 5% of the AFLR code, achieving high code re-use. By the completion of this project, the robustness, stability, and end-user productivity aspects of the parallel mesh generation will be delivered.

1. Introduction

Simulations that involve complex geometries and physics are used in the health-care, transportation, and defense industries. Large meshes are required to produce accurate numerical results in simulations, such as in computational fluid dynamics applications. There are several sequential mesh generation and refinement procedures currently in use in these industries. Advancing Front Local Reconnection (AFLR) is one of them (Marcum & Weatherill, 1995). Although this software produces meshes of the highest quality, it has not been parallelized yet. When generating large meshes (consisting of billions of elements), it is imperative to utilize parallel meshing algorithms. The Center for Real-time Computing (CRTC) team at Old Dominion University (ODU) developed Parallel Delaunay Refinement (Chernikov & Chrisochoides, 2006).

Parallel Delaunay Refinement maintains a fixed level of concurrency while parallelizing the refinement process and guarantees four essential requirements – stability, robustness, code re-use, and scalability. Stability ensures that a mesh generated in parallel maintains a level of quality comparable to that of a sequentially

generated mesh. This quality is determined by the number and shape of the elements. Robustness guarantees that the parallel software produces the same geometries as its sequential counterpart. The input would be decomposed into many concurrently processed subdomains, so operator intervention is both highly expensive and infeasible. Code re-use essentially means that the parallel algorithm should be designed in such a way that it can be replaced and/or updated with minimal effort, regardless of the sequential meshing code it uses. This is the only practical approach due to the fact that sequential codes are constantly evolving to accommodate the functionality requirements from the wide ranges of applications and input geometries. Rewriting new parallel algorithms for every sequential meshing code can be highly expensive in time investment. Scalability compares the runtime of the best sequential implementation to the runtime of the parallel implementation, which should achieve a speedup. Non-trivial stages of the computation must be parallelized if one is to leverage current architectures that contain millions of cores.

Parallel Delaunay Refinement decomposes an input geometry by introducing an octree, where each node of the octree may contain a subdomain of the geometry that is established by a node (or leaf) boundary. The boundaries of the nodes do not become a part of the mesh because this would only add constraints to the meshing problem and reduce the available optimization space. A Delaunay refinement algorithm is used, which has been proven to produce a mesh with guaranteed bounds on radius-edge ratio and on the density of elements. Delaunay refinement replaces poor quality elements with those of better quality by inserting Steiner points inside the circumdisks of the poor quality elements, thus eliminating them. There is a risk of losing stability and robustness when concurrently inserting points into two separate subdomains (or leaves of the octree) due to a possible data dependency (Chernikov & Chrisochoides, 2004). A buffer zone must be established around a leaf when it is scheduled for refinement. When one thread refines a part of the mesh associated with a leaf of the octree, no other thread may refine the parts of the mesh associated with the buffer zone around that leaf. This eliminates any data dependency risks and avoids fine grain synchronization overheads caused by concurrent point insertions. A thread refines a leaf by running a sequential refinement code on the subdomain within that leaf.

The implementation of the new Parallel Data Refinement (PDR) presented in this paper uses the Advancing Front Local Reconnection method to refine individual leaves. AFLR accepts an input geometry with an established boundary triangulation. A Delaunay triangulation criterion is used to construct an initial boundary conforming tetrahedral mesh. Each initial boundary point is assigned a value, by a point distribution function, representative of the local point spacing on the boundary surface. This function is used to control the final field point spacing. All elements are initially made active, meaning that they need to be refined. If the edge points of an element satisfy the point distribution function, the element is made inactive and does not need to be refined. The advancing front method is used on an active element. A face of the element that is adjacent to another active element is selected. A new point is created by advancing in a direction, normal to the selected face, a distance that would produce an equilateral element based on an appropriate length scale (using the average point distribution). If a new point is too close to an existing point or another new point, it is rejected and removed. Accepted points are inserted into the existing grid by subdividing their containing

elements. For example, if an edge point is inserted, then all elements sharing that edge are split. If a face point is inserted, then both of the elements sharing that face are split into three elements. All elements modified by point insertion, or any that undergo reconnection, are classified as active. A local reconnection scheme is used to optimize the connections between points (or edges). Edges are repeatedly reconnected, or swapped, until their containing elements satisfy a desired quality criterion. Finally, all active elements undergo three quality improvement passes, which consist of sliver removal and further reconnection.

2. Integration of AFLR with PDR

Minimal modifications were made to AFLR to ease its integration into PDR.AFLR. PDR.AFLR is designed to use any refinement code on a leaf/subdomain. This requires an API. Two functions were created within AFLR for this purpose – a function that constructs only the initial volume mesh and passes this data back to PDR.AFLR (to construct the octree and assign data to the octree leaves) and a function that calls the AFLR refinement code using the subdomain data given as parameters.

As stated earlier, AFLR accepts an input geometry that has an established boundary triangulation. The subdomain boundary surface must satisfy several rules. At this point, we focus on a manifold genus zero input geometry. The points of these surface triangles must have consistent, right-hand rule ordering and there must not be any one edge that is shared by more than two triangles. The boundary must be a closed surface. Since the input geometry is subdivided among the leaves of the octree, this means that no leaf will initially have a closed boundary.

An algorithm was added to PDR.AFLR (seen in figure 1) which generates a temporary boundary, connecting it with the input geometry external boundary found within that leaf. If the leaf is internal (completely inside the geometry), a full boundary is generated around the entire leaf. The current version of the implementation accepts only manifold genus zero input geometries, thereby guaranteeing that there are no holes in the connectivity of tetrahedra. With this assumption, there will always be either external boundary triangles or tetrahedra surrounding a leaf. The connectivity between the external boundary triangles is closed and the connectivity between surrounding tetrahedra is closed. Every tetrahedron immediately located outside an octree leaf boundary is connected to a tetrahedron inside that boundary. If the faces shared between these adjacent tetrahedra are collected, a closed boundary is created. Since every tetrahedron has a neighboring tetrahedron or triangle adjacent to each of its faces, a closed boundary surface is guaranteed. Once refinement is complete, the portion of the boundary created temporarily for AFLR is removed.

```
procedure Gather_Boundary (numTets, tetrahedra, tetrahedraNeighbors, numTrias, surfTrias)
Input: numTets is the number of tetrahedra contained by this leaf
       tetrahedra is the list of tetrahedra contained by this leaf
       tetrahedraNeighbors is the list of neighbors for each tetrahedron in the list of tetrahedra
       numTrias is the number of surface triangles contained by this leaf
       surfTrias is the list of surface triangles contained by this leaf
Output: surfTrias contains more triangles and numTrias has been increased as this leaf now has
a closed boundary
1      for i := 1 to numTets
2          for j := 1 to 4
3              if tetrahedraNeighbors_j is not a triangle and is contained by a different leaf
4                  then P1 := points of tetrahedra_i
5                       P2 := points of tetrahedra_j
6                       T := empty list of points for new triangle
7                       for k := 1 to 4
8                           for m := 1 to 4
9                               if P1k = P2m then add P1k to T
10                      Rearrange points in T to establish right-hand ordering
11                      Add T to surfTrias
12                      numTrias := numTrias + 1
```

Figure 1 Boundary Gathering Algorithm

An example of a mesh divided into octree leaves can be seen in figure 2. One of these leaves is shown with a closed boundary in figure 3. The mesh consists of a rocket body, engine, and nozzle that are all enclosed in an outer boundary that is relatively close to the rocket, keeping the domain size small. After refinement has been completed for all leaves, all of the leaf data is combined to create one mesh which is finally output.

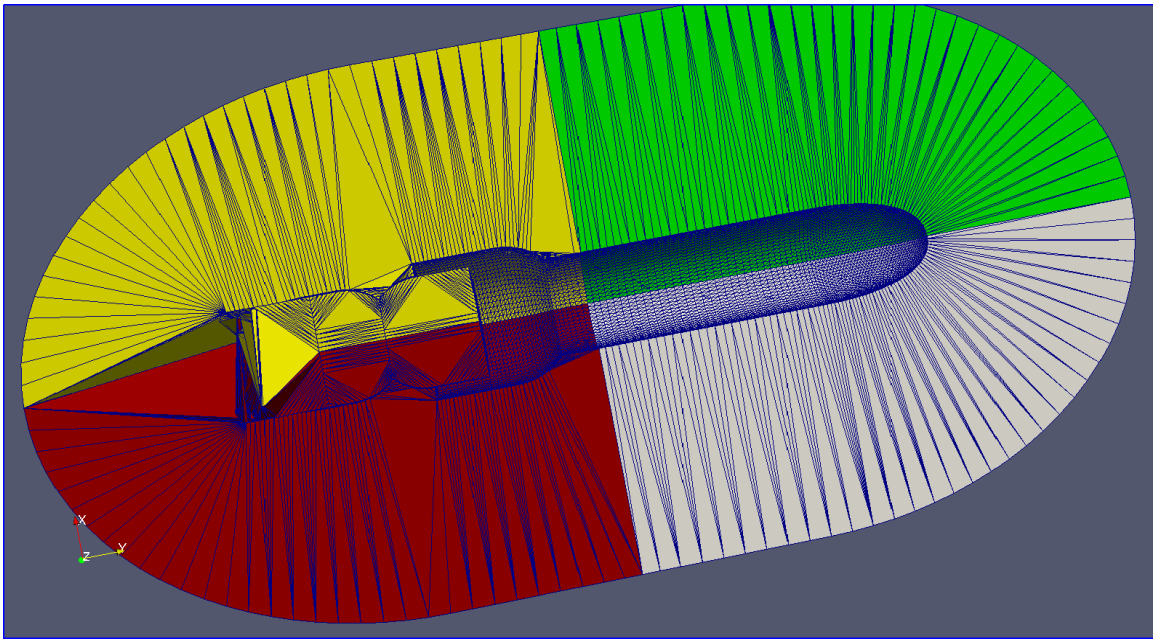

Figure 2 Initial Volume Mesh of Rocket Divided into Octree Leaves/Subdomains

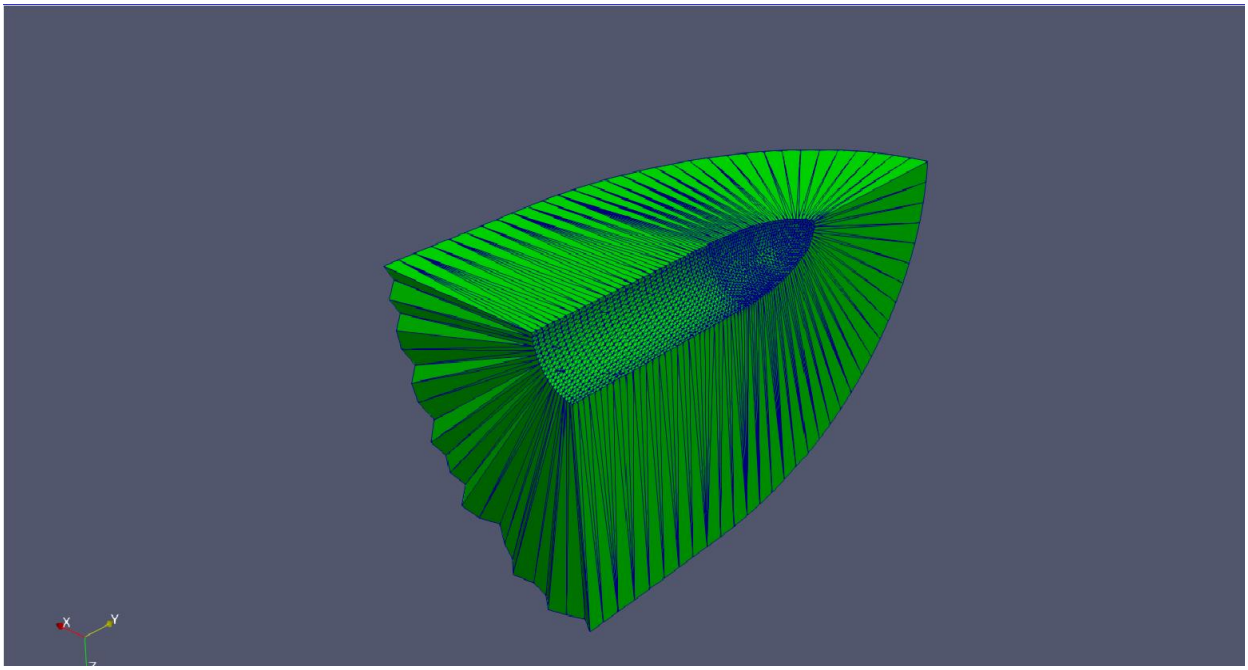

Figure 3 Octree Leaf Subdomain of Rocket Initial Volume Mesh with Closed Boundary

3. Future Work

There is still much to be accomplished for the completion of PDR.AFLR. In the current implementation, elements near the boundary are not refined in order to maintain connectivity between the subdomains, which affects stability. Since these elements do not undergo refinement, they retain poor quality in the final mesh. Figure 4 shows the distribution of the low quality elements in PDR.AFLR compared to the serial AFLR. The next implementation will allow the refinement of elements near the boundary while still preserving leaf connectivity. AFLR currently does not refine past the specified boundary triangulation of the domain. In the next implementation, point insertion will be modified to occur on the boundaries themselves. The buffer zone around a leaf will also be used to establish a secondary boundary for AFLR, in which it will allow local reconnection to occur outside the initial boundary but inside the secondary boundary. These changes will help to eliminate any poor quality elements and will establish much stability to rival that of the serial AFLR.

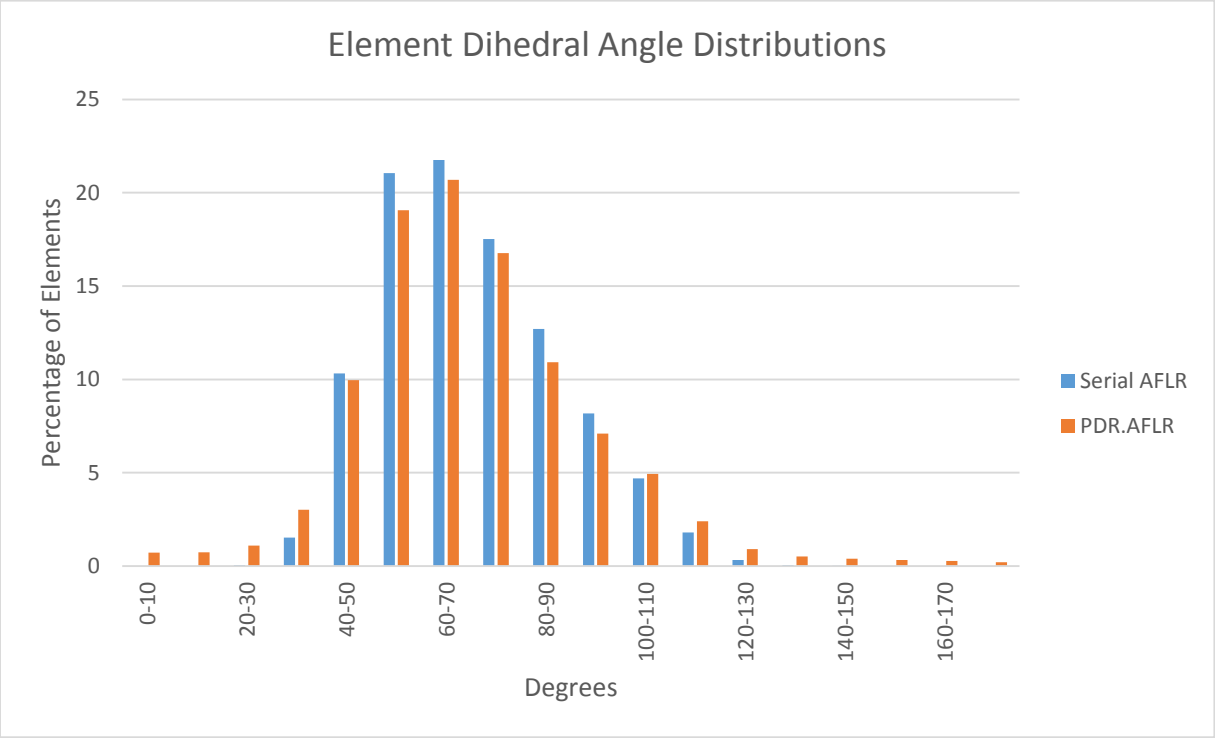

Figure 4 Element Dihedral Angle Distributions between Serial AFLR and PDR.AFLR

Pthreads will be used in the shared memory implementation of this software and three different distributed memory implementations will be developed. One will make use of MPI, another with OCR from Intel, and the other will use a subsequent project containing OCR's integration. This project is of a runtime system that is designed to be a multi-layered algorithmic and software framework for 3D tetrahedral anisotropic parallel unstructured mesh generation and adaptation to support state-of-the-art functionality. The CRTC team developed a flexible load balancing framework and runtime software system called PREMA (Parallel Runtime Environment for Multicomputer Applications) for supporting the development of adaptive applications on distributed-memory parallel computers (Barker, Chernikov, Chrisochoides, & Pingali, 2004). PREMA supports a global namespace, transparent object migration, automatic message forwarding and routing, and automatic load balancing. One particular CRTC team member, Polikarpos Thomadakis, has improved upon the framework by integrating into it MPI-3 and making modifications so that it can be easily integrated with another framework which would control the multithreaded communication layer of this runtime system.

As part of this project, two options were considered to control this layer – High Performance ParalleX (HPX-5), developed by the Center for Research in Extreme Scale Technologies (CREST) at Indiana University, and the Open Community Runtime (OCR) framework ("Open Community Runtime," 2016). OCR was selected as the better candidate, given its intended features and capabilities and because of the fact that it is under development by Intel. When integrated into the PREMA runtime system, OCR will control virtual nodes (i.e. the maximum number of hardware nodes for which OCR provides good performance) and processes in both distributed and shared memory. PREMA will be integrated with OCR to scale a number of virtual nodes, which is ideal since both of these systems' APIs are similar. PREMA will manage course-grain granularity while OCR will handle medium-to-fine-grain granularity. PREMA will be modified to maintain a queue of work units (remote handlers and data), where a unit is thought of as an OCR Event Driven Task (EDT). This gives OCR the responsibility of assigning tasks to nodes/threads. PREMA will essentially control how messages are passed between virtual nodes, in the context of remote unit mobility. This mobility will allow the CRTC's runtime system to relocate tasks and data to respond to any system failures, achieve a better balance of load among processes, and to optimize memory and energy consumption while exhibiting strong scaling performance. PDR.AFLR will be the ideal application to test on this new runtime system once it is complete.

4. Conclusion

The Parallel Data Refinement (PDR) application maintains a fixed level of concurrency while parallelizing the refinement process and guarantees stability, robustness, code re-use, and scalability. PDR.AFLR essentially decomposes an input geometry into subdomains and refines each subdomain using a sequential mesh refinement code, then combines all of the refined subdomain data into a single, conforming mesh. The Advancing Front Local Reconnection method was used for refinement in this implementation. Some modifications were made to both PDR and AFLR to accommodate each other. A new API was created within AFLR and a subdomain boundary creation method was developed within PDR.AFLR. More modifications will be made to AFLR in the

next implementation to increase not only the stability of the software, but create a significant speedup that will make PDR.AFLR much more preferable over using the serial AFLR. Developing four implementations for this software (shared memory, distributed memory with MPI, distributed memory with OCR, and distributed memory with the CRTC runtime system) will provide a wide range of options for users, allowing for fast, high quality mesh adaptation in an extreme-scale environment.

5. Acknowledgements

We would like to thank Christos Tsolakis and Fotis Drakopoulos for their valuable insight on mesh generation and refinement procedures. This work in part is funded by the Virginia Space Grant Consortium (VSGC) Graduate Research Fellowship, NSF grant no. CCF-1439079, NASA grant no. NNX15AU39A and DoD's PETTT Project PP-CFD-KY07-007.

6. References

Barker, K., Chernikov, A., Chrisochoides, N., & Pingali, K. (2004). A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems, 15(2)*, 183-192. doi: 10.1109/TPDS.2004.1264800

Chernikov, A. & Chrisochoides, N. (2006). Parallel guaranteed-quality delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing, 28(5)*, 1907-1926. doi: 10.1137/050625886

Chernikov, A. & Chrisochoides, N. (2004). Practical and efficient point-insertion scheduling method for parallel guaranteed-quality delaunay refinement. *18th ACM International Conference on Supercomputing*, 48-57. doi: 10.1145/1006209.1006217

Marcum, D. & Weatherill, N. (1995). Unstructured grid generation using iterative point insertion and local reconnection. *AIAA Journal, 33(9)*, 1619-1625. doi: 10.2514/3.12701

(2016). Open community runtime. Retrieved from https://xstackwiki.modelado.org/Open_Community_Runtime

United States Department of Defense. (2016). *Scalable software framework and algorithms for parallel mesh generation and adaptation* (Final Technical Report for PP-CFD-KY07-007-P3, DoD's User Productivity Enhancement, Technology Transfer, and Training (PETTT) Program High Performance Computing Modernization Program (HPCMP)). Washington, DC: Kennedy, T., Chernikov, A., & Marcum, D.