STABILITY OF ADVANCING FRONT LOCAL RECONNECTION FOR PARALLEL DATA REFINEMENT

Kevin Garner, Thomas Kennedy, Christos Tsolakis, and Nikos Chrisochoides*

Department of Computer Science

Old Dominion University

Norfolk, VA 23529, USA

Abstract

The status of a long-term project entailing the parallelization of an industrial-strength sequential mesh generator, called Advancing Front Local Reconnection (AFLR), is presented in this paper. AFLR 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 parallel procedure that is presented is called Parallel Data Refinement (PDR) and 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. AFLR is shown to maintain weak reproducibility (i.e., able to produce results of the same quality when executed recurrently using the same input) as required in parallel mesh generation. The data-decomposed AFLR implementation is shown to be stable (i.e., guarantees sufficient mesh quality that is comparable to that of the original AFLR software). By the completion of this project, PDR.AFLR will be robust (i.e., generate meshes for the same type of geometries that AFLR can) and will be the first fully functional unstructured mesh generation/refinement application that will be capable of maintaining good parallel efficiency at 10^6 concurrency levels in order to improve end-user productivity.

1. Introduction

Mesh generation software is used in many industries where high fidelity simulations are executed, such as in health-care, defense, and aerospace. For the last 30 years, these sequential software codes were typically developed and optimized without any thought towards scalability. One such code is called Advancing Front Local Reconnection (AFLR), one of the top, industrial strength, mesh generators that is currently used by NASA, the DoD, DoE, and a number of aerospace industry top research groups [1]. AFLR has not been fully parallelized to properly utilize large-scale supercomputing hardware due to the geometric and numerical challenges imposed by the nature of mesh generation complexity. Given the high performance computing (HPC) platforms that the aforementioned organizations use, one long-term goal is to achieve large-scale adaptive Computational Fluid Dynamics (CFD) simulations. The large-scale parallelization of AFLR will help realize this goal. The Center for Real-time Computing (CRTC) at

*{kgarner, tkennedy, ctsolakis, nikos}@cs.odu.edu

Old Dominion University (ODU) has proposed the telescopic approach (see figure 1) [2] [3], a framework that will leverage the concurrency that exists at multiple levels in parallel grid generation. At the chip and node levels, the telescopic approach deploys a Parallel Optimistic (PO) layer and Parallel Data Refinement (PDR) layer, respectively. AFLR will be integrated with the PDR layer, which in turn will be implemented on top of the PO layer in future efforts.

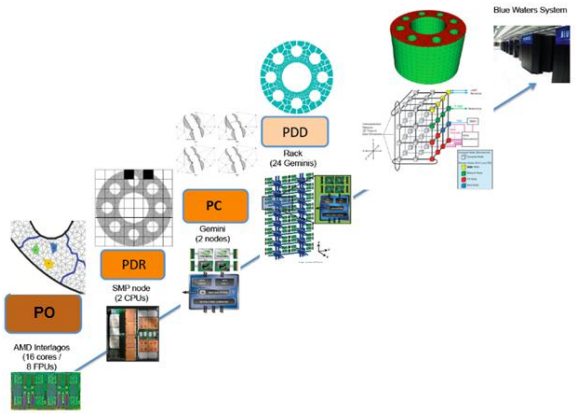

Figure 1. Telescopic Approach to Parallel Mesh Generation. It covers the complete spectrum of hardware that spans from the Chip (bottom left) to a complete machine like Blue Waters (top right). The PO level is on the left and targets the chip level. The PDR approach is second from the left and targets hardware at the node level.

PDR maintains a fixed level of concurrency while parallelizing the refinement process. Its methodology is theoretically proven, for isotropic meshes, to maintain stability and robustness for parallel isotropic Delaunay-based mesh generation and has been experimentally verified [4] [5] [6]. It is also designed to allow for the utilization of any sequential mesh generator while guaranteeing the following four requirements for parallel mesh generation: stability, robustness, scalability, and code re-use. 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 is able to correctly and efficiently process any input data. Operator intervention into a massively parallel computation is not only highly expensive, but most likely infeasible due to the large number of concurrently processed sub-problems. 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. 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 a 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. The code re-use approach is only feasible if the sequential mesh generator satisfies the reproducibility criterion. Reproducibility requires that the sequential mesh generator, when executed with the same input recurrently, produces either identical results or those of the same quality. Elements within a mesh may undergo refinement more than once when in parallel, so it is imperative that the sequential mesh generator satisfy this requirement. Previous work involving the

integration of the mesh generator TetGen with PDR shows that if the mesh generator fails to meet the reproducibility criterion in distributed memory, then the complexity of such state-of-the-art codes inhibits their modifications to a degree that their integration with parallel frameworks like PDR becomes impractical [7]. The original, sequential AFLR code was determined to be a suitable mesh generator to integrate with PDR, as it was tested and shown to maintain weak reproducibility, as seen in figures 2 and 3. Examples of refinement for two geometries are shown. The dihedral angle quality of the output mesh from the initial refinement is given. This is compared to the quality of the output meshes generated from refining the output volume and from refining the surface taken from the initial output.

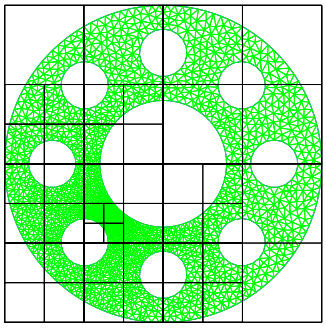

Figure 4. PDR octree (in 2-D).

Parallel Data Refinement decomposes a meshing problem by using an octree consisting of numerous leaves, or subdomains, that each hold a part of the mesh (2-D example shown in figure 4). The general idea of PDR is to concurrently refine the octree leaves while maintaining mesh conformity. The main concern when

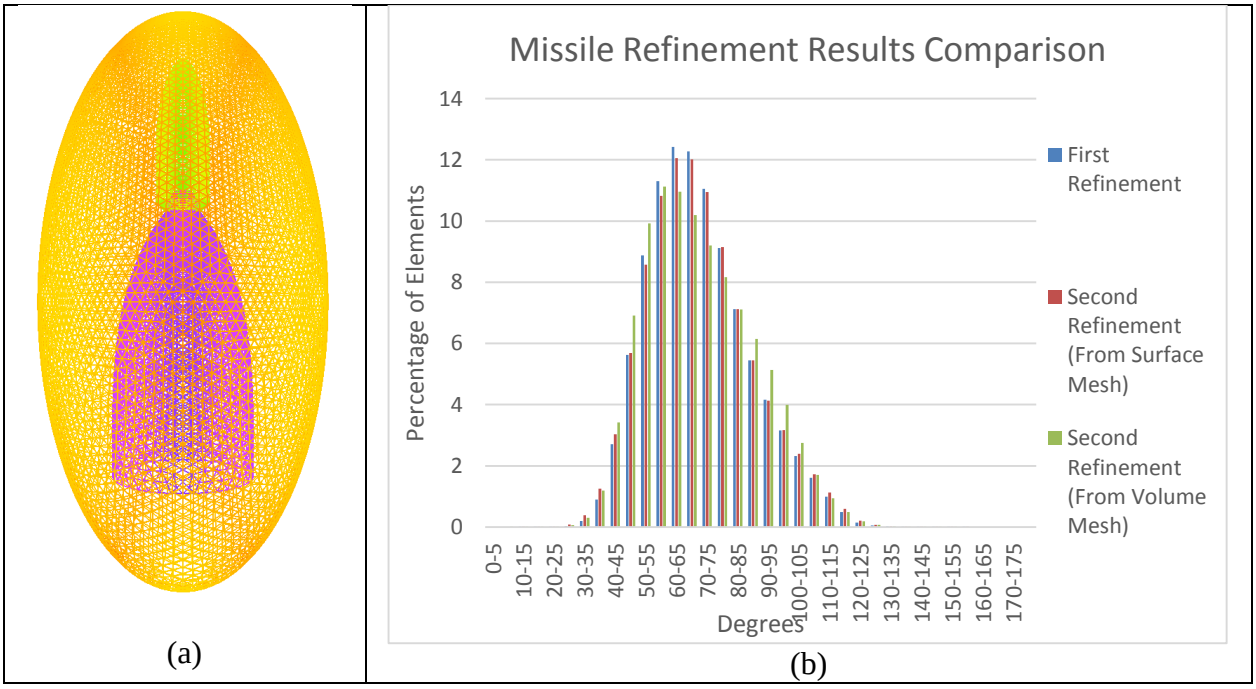

Figure 2. (a) shows the missile geometry. (b) compares the dihedral angle distributions of the output mesh from the first refinement, to the meshes output from refining the initial output volume and from refining the surface taken from the initial output volume.

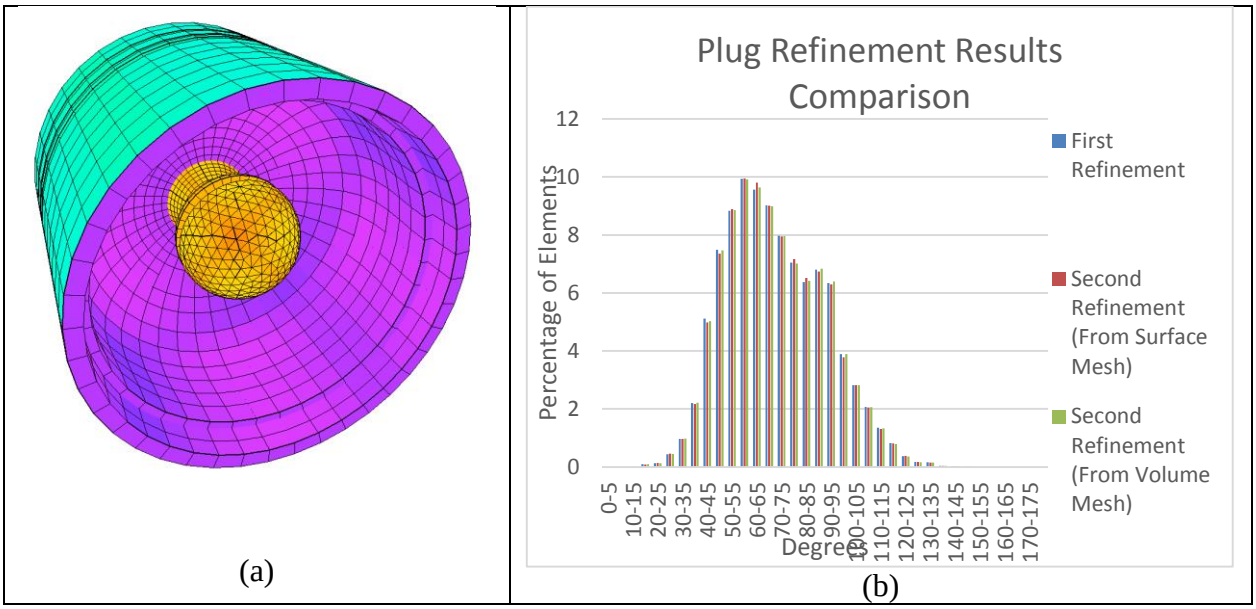

Figure 3. (a) shows the plug geometry. (b) compares the dihedral angle distributions of the output mesh from the first refinement, to the meshes output from refining the initial output volume and from refining the surface taken from the initial output volume.

parallelizing a refinement algorithm are the data dependencies between leaves caused by concurrent point insertions and the creation/deletion of elements in different octree leaves by multiple threads concurrently. PDR addresses this issue by introducing a buffer zone around each octree leaf. If a part of the mesh associated with a leaf is scheduled for refinement by a thread, no other thread can refine the parts of the mesh associated with the buffer zone of this leaf. This eliminates any data dependency risks and allows PDR to avoid fine-grain synchronization overheads associated with concurrent point insertions. A thread refines a leaf by running a sequential refinement code on the subdomain within that leaf.

The implementation of 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. All active elements undergo a final optimization phase, which consists of three quality improvement passes (sliver removal and further reconnection).

2. Integration of AFLR with PDR

Although PDR is designed to make the parallelization of sequential mesh generators more facile, several modifications had to be made to the original AFLR software code in order to ease its integration into the PDR framework (henceforth referred to as the data-decomposed AFLR). A large sum of time was invested in making these modifications due to the nature and complexity of the code. An intricate understanding of the data structures and methodologies used for the initial volume grid generation, point insertion, element edge swapping, and optimization is required in order to integrate these processes into a parallel framework. Additions were also made to the PDR code. The following steps outline the general process of the data-decomposed AFLR:

1. Accept an input geometry.
2. Generate an initial volume mesh.
3. Construct an octree.
4. Assign subdomains to octree leaves and insert leaves into refinement queue.
5. Remove a leaf from the queue. Create a temporary boundary for the leaf based on original element connectivity with neighboring subdomains.
6. Call AFLR to refine the leaf.
7. Merge all neighboring leaves, located within the buffer zone, with the newly refined leaf.
8. Call AFLR to perform local reconnection on the merged data.
9. Assign updated data to the necessary leaves.
10. Repeat steps 5-9 until there are no remaining leaves in the refinement queue.
11. Merge all data and call AFLR to perform final optimization.
12. Output the final mesh.

Tetrahedra are assigned to octree leaves based on where their barycenters fall. If a tetrahedron has an external boundary face, the face is assigned to the same leaf to which the tetrahedron is assigned. In step 5, a smooth, simply-connected boundary (as required by AFLR) must be extracted for the leaf under refinement [8]. The set of tetrahedra within this leaf is examined in isolation (as if this subdomain is the entire domain). Any face that is not shared between tetrahedra is considered to be a boundary face. It is possible to extract a boundary that contains an edge which is shared by more than two triangles. This is not acceptable for AFLR. This scenario occurs when there is a tetrahedron that has a barycenter just over the leaf boundary, causing it to be assigned to a neighboring leaf. Figures 5 and 6 show such a scenario. Figure 5 shows a green line, which is the

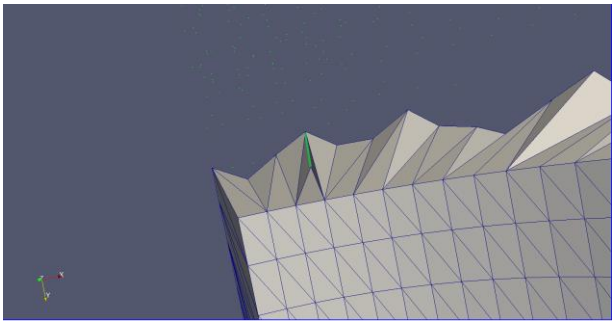

Figure 5. Top-down view of extracted leaf boundary that contains an edge which is shared by more than two faces.

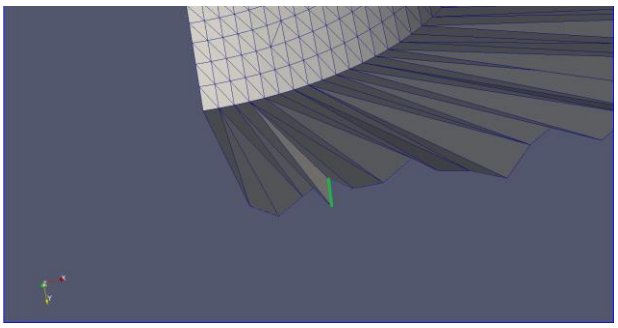

Figure 6. Top-down view of neighboring leaf that contains the sliver with the edge shared by the faces of the leaf in figure 2.

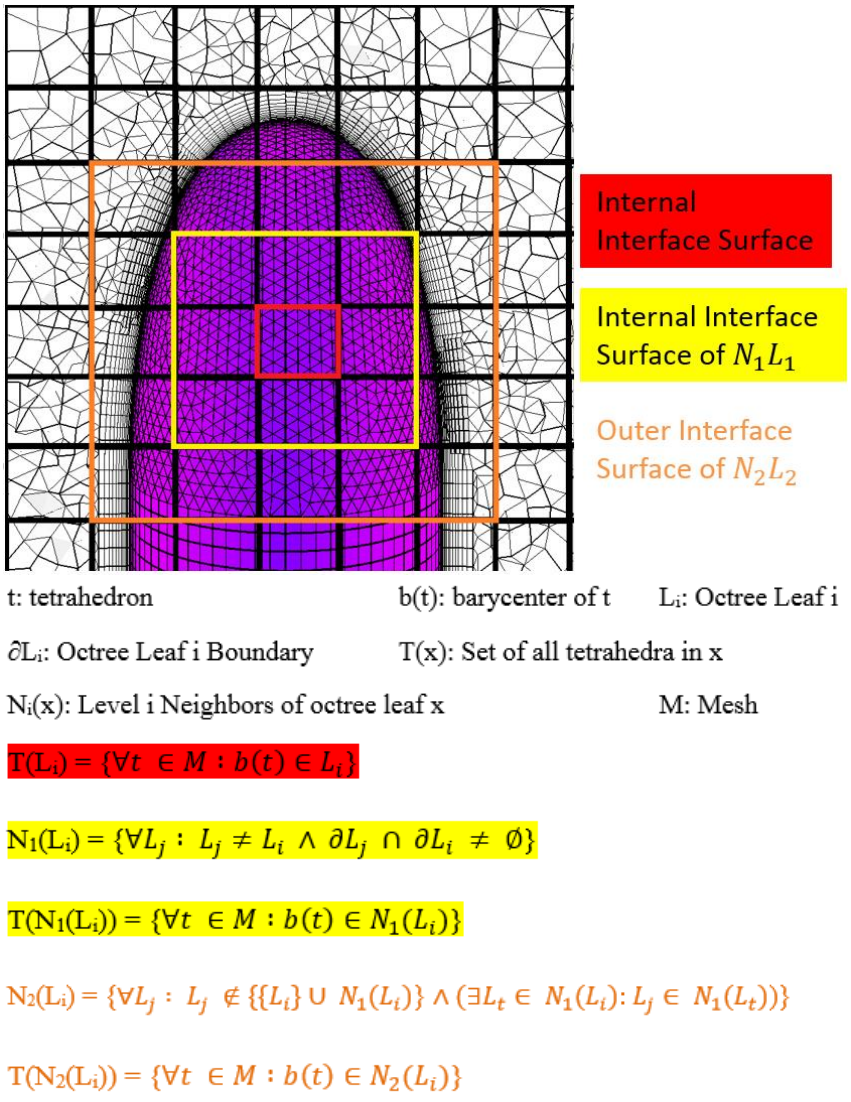

Figure 7. Shown is a 2-D example of the upper portion of a data-decomposed rocket mesh where the red-boxed leaf is the primary leaf under refinement. The level 1 neighbors are those inside the yellow box (excluding the red leaf) and the level 2 neighbors are those inside the orange box (excluding all leaves inside the yellow box). The formulas give mathematical representations that denote the tetrahedra within leaves and the sets of neighboring leaves (with matching colors showing what is contained within each surface).

edge that is shared by more than two triangles. Figure 6 shows the sliver that shares this edge, but was assigned to a neighboring leaf. When such an edge is found in the extracted boundary, the corresponding tetrahedron is located within the neighboring leaf, removed from that leaf, and added to the primary leaf. The boundary is extracted and examined again. This process repeats until a smooth boundary, acceptable for AFLR, is extracted.

Figure 7 shows a 2-D example of PDR's data decomposition and the assignment of data generated from the upper portion of a rocket geometry. Mathematical formulas are given for the different levels of neighboring leaves around the primary leaf under refinement (in red). The level 1 neighbors of a leaf are considered to be the buffer zone of

that leaf (no leaf in the buffer zone may undergo refinement while the primary leaf undergoes refinement).

Another implementation challenge, caused by data decomposition, is allowing subdomain boundary elements to undergo refinement. If a boundary face, shared by two leaves, undergoes refinement, then the corresponding elements within both leaves must be updated (which adds dependencies and increases overall runtime due to the required communication between the corresponding threads). Otherwise, the connectivity between the subdomains will be incorrect and the final mesh will be non-conforming. Subdomain boundary refinement is preferred so that boundary elements do not retain poor quality by the end of refinement. To solve this issue, AFLR was modified to not only accept a single set of data (points, triangles, and tetrahedra) for one leaf, but to also accept a second set of data – the set of all of its level 1 neighboring leaves. The faces of the primary leaf's internal interface surface are kept frozen in step 6, meaning that point insertion is not allowed on the leaf boundary. AFLR refines the individual leaf (advancing front point placement/insertion and local reconnection) but does not make any optimizations/quality improvement as the serial AFLR would. Instead, the newly refined leaf is merged with its level 1 neighbors into a super-subdomain and local reconnection is performed over the super-subdomain (thereby allowing the optimization of the primary leaf's boundary elements). The internal interface surface of the level 1 neighbor leaves remains frozen, so as to eliminate the need of updating level 2 neighbors during refinement (and maintaining PDR's original method of concurrency). It is possible to have duplicate points when combining these sets of data because a neighboring leaf may contain a tetrahedron that has a point located in the primary leaf, or vice-versa. Each set of data will contain that same point, so the duplicate point is removed and any tetrahedron or triangle that references this point is updated to use the same index (all tetrahedra and faces use integer-based indices to denote which points they contain). The removal of duplicate points is necessary as they are not permitted by AFLR.

Once local reconnection over the super-subdomain has completed, this refined data is returned to PDR and is assigned to octree leaves. No points are deleted during refinement, so only new points are added to leaves. All previous tetrahedra data within the leaf and its level 1 neighbors are deleted. The new tetrahedra are assigned to leaves. Having underwent swapping, a tetrahedron will have a different volume and will therefore have a different barycenter. It is possible for the barycenter to move just enough to be assigned to a level 2 neighbor. If a level 2 neighbor must be updated during refinement, then this limits parallelism and conflicts with PDR's method of concurrency. A thread should only refine a leaf and its level 1 neighbors without allowing any changes to propagate beyond the level 1 region. If this situation occurs, the tetrahedron is assigned to a level 1 neighboring leaf that is a neighbor of the level 2 leaf. Consider the following.

Let the octree leaf under refinement be L_1. The level 2 neighbor $n \in N_2(L_1)$ is where the new barycenter of the tetrahedron falls. Examine the level 1 neighbors, $N_1(L_1)$, and find a leaf that is also included in $N_1(N_2(L_1))$. When a matching leaf is found, assign the tetrahedron to that leaf instead. Although the tetrahedron is assigned to an inaccurate location, it will still reference the correct points (using the appropriate indices). If this causes a boundary extraction problem (as described above) during the refinement of another leaf, then the tetrahedron will be moved accordingly. The goal is to avoid making any alterations to a level 2 neighbor. If that tetrahedron is moved to another leaf, then it will be moved during a phase of refinement for that leaf, in which case the altering of that leaf's data is acceptable because it will be a part of another refinement process occurring after the current leaf has completed refinement. This removes the need to add new data to a level 2 neighbor during refinement.

Finally, a function was added to AFLR which simply accepts a set of data and performs quality improvement/optimization on it (sliver removal and local reconnection). After all leaves have undergone refinement, their data are combined into a single set and passed into this optimization function.

3. Qualitative Results

Preliminary results from the initial implementation of the sequential, data-

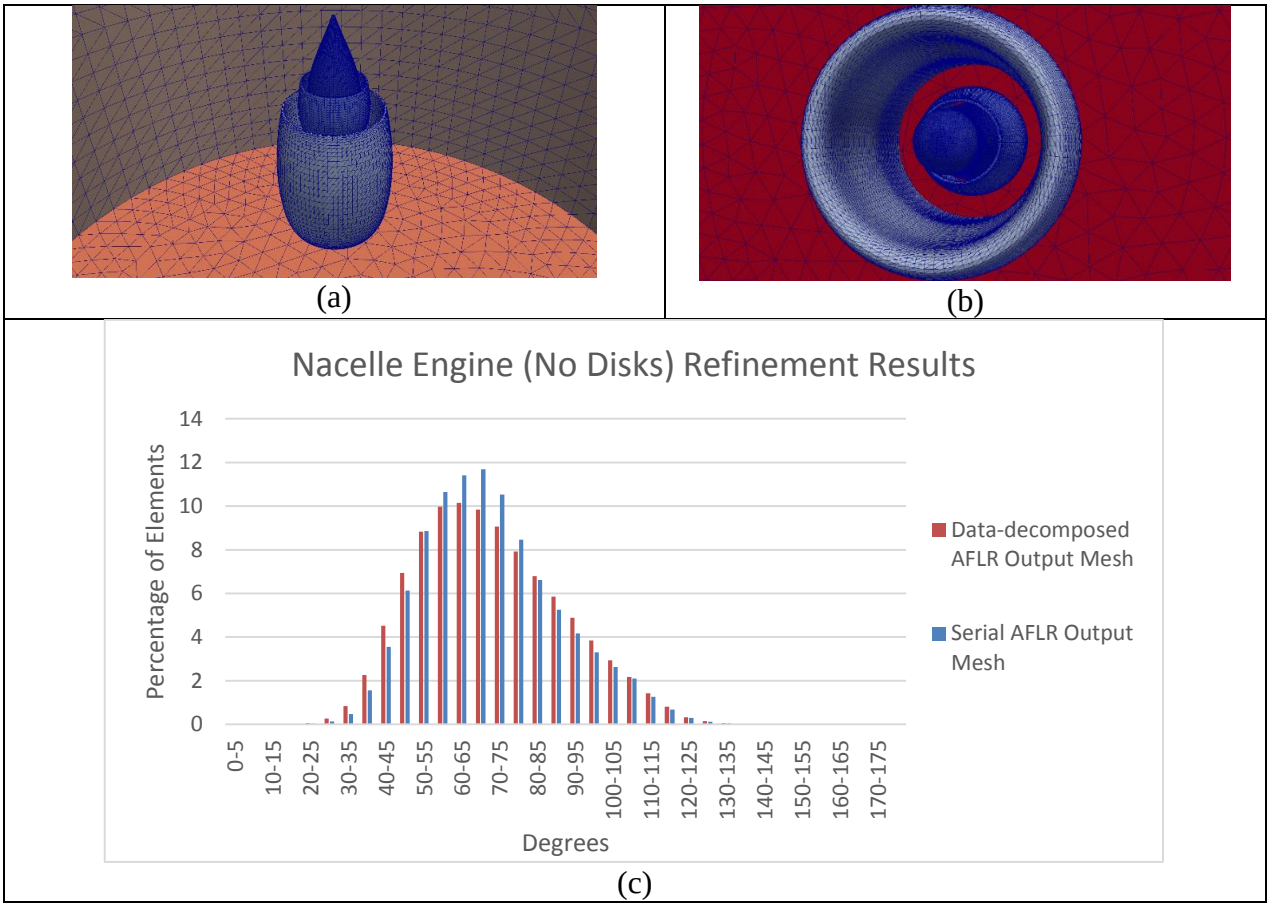

Figure 8. The Fan and Turbine Disk surfaces were removed from the nacelle engine geometry. (a) and (b) show different viewpoints of the geometry. (c) compares the dihedral angle distributions of the output meshes between the serial AFLR code and PDR.AFLR.

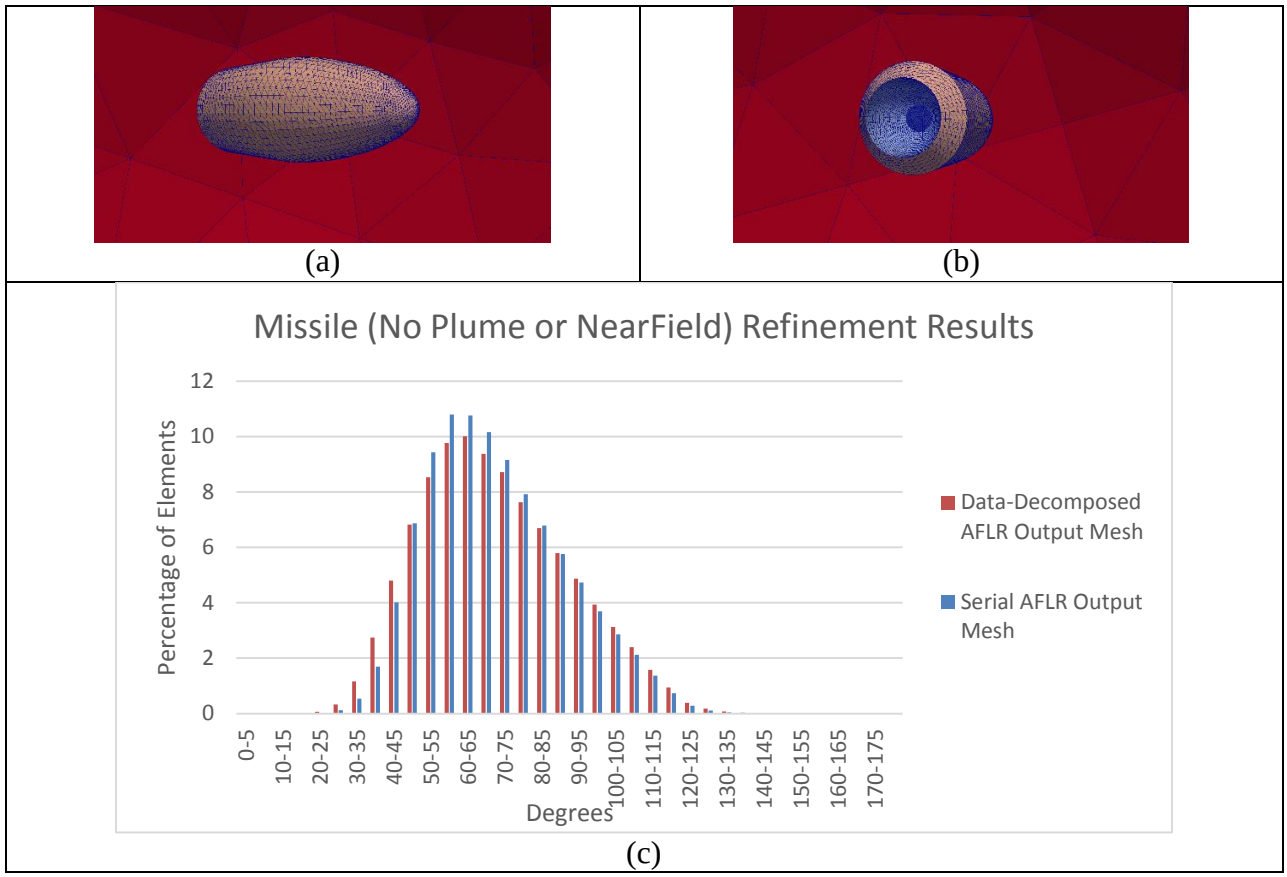

Figure 9. The Plume and NearField embedded surfaces were removed from the missile geometry. (a) and (b) show different viewpoints of the geometry. (c) compares the dihedral angle distributions of the output meshes between the serial AFLR code and PDR.AFLR.

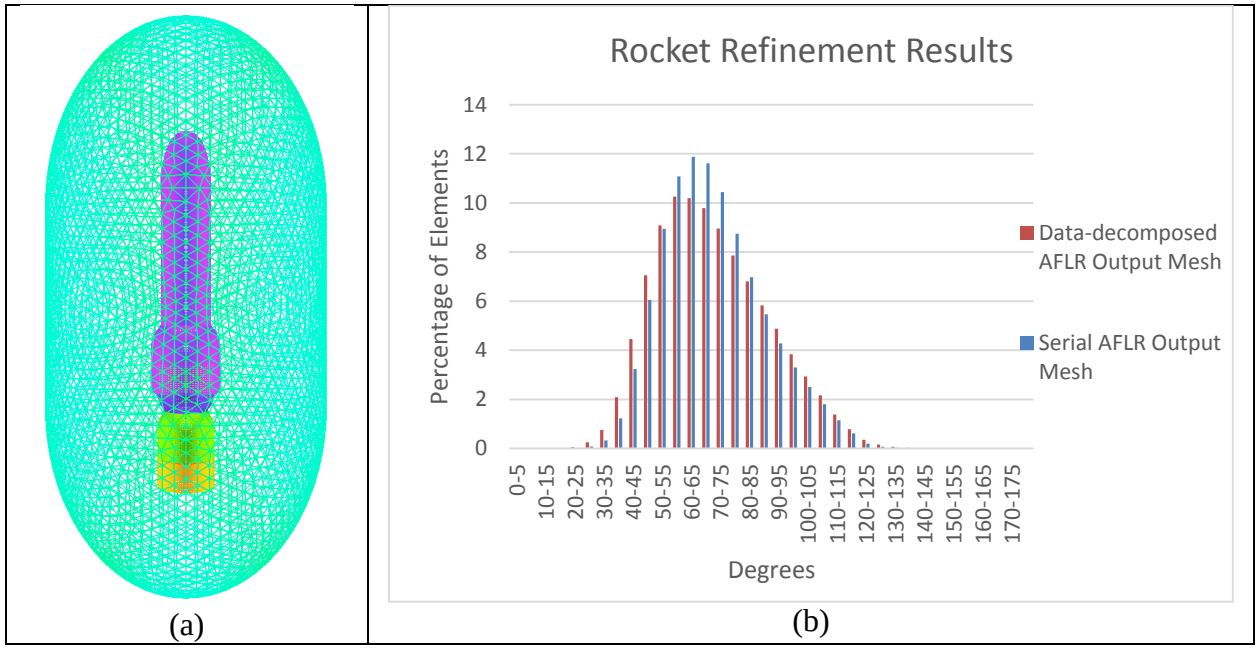

Figure 10. (a) shows the rocket geometry. (b) shows the dihedral angle distributions of the output meshes compared between the serial AFLR code and PDR.AFLR.

decomposed AFLR show that PDR's data decomposition does not hinder the quality of the output as it can be seen from the dihedral angle quality statistics of meshes in comparison to their quality when generated by the original AFLR code, in figures 8, 9, and 10. While the output meshes of the modified AFLR contain slightly more elements of lower quality (percentage of elements towards both ends of the charts), it maintains its stability with a close number of high quality elements to that of the original AFLR software output. This implementation is limited to the refinement of manifold, genus zero geometries that do not contain transparent/embedded surfaces. These limitations will be addressed in the final, complete implementation of PDR.AFLR. For simplicity, any transparent/embedded surfaces were removed from certain geometries while testing the stability of the data-decomposed AFLR (specified in the figures).

4. Future Work

There is still much to be accomplished for the completion of PDR.AFLR. The current implementation of the data-decomposed AFLR accepts only manifold genus zero computational geometries. Robustness will be addressed by identifying leaves that contain disconnected volumes of a mesh (caused by hole(s) in the geometry) and these individual pieces will be refined independently of each other. A new methodology will also be developed to allow the data-decomposed AFLR to process geometries with transparent/embedded surfaces. Code Re-use will be addressed by further modifying the design of PDR and developing a universal API, one that is capable of handling different data types/structures of different mesh generators.

Scalability will be achieved by fully integrating PDR.AFLR onto a runtime system called PREMA 2.0 [9]. During run time, the PDR.AFLR method will expose data decomposition information (number of subdomains waiting to be refined) to the underlying run-time system. In turn, this system will facilitate work-load balancing and guide the program's execution towards the most efficient utilization of hardware resources. PREMA 2.0 is a parallel runtime system that supports asynchronous communication, global address space and load balancing for adaptive and irregular applications. Capable of executing applications in both shared and distributed memory, PREMA 2.0 alleviates the burden of monitoring data and computations in parallel and as such is an ideal system to support the execution of PDR.AFLR.

5. Conclusion

The Parallel Data Refinement method maintains a fixed level of concurrency while parallelizing the refinement process and guarantees stability, robustness, code re-use, and scalability. The data-decomposed AFLR essentially decomposes an input geometry into subdomains, refines each subdomain using the Advancing Front Local Reconnection mesh refinement code, and then combines all of the refined subdomain data into a single, conforming mesh. Several modifications were made to both PDR and AFLR in order to accommodate each other and reliably generate meshes with quality comparable to those generated by the original AFLR software. AFLR meets the reproducibility requirement and the data-decomposed AFLR maintains its stability. The data-decomposed AFLR will undergo further development and become fully integrated into PDR in order to meet the remaining parallel mesh generation requirements – robustness, code re-use, and scalability. By the completion of this

project, PDR.AFLR will be the first fully functional unstructured mesh generation/refinement application that will be capable of maintaining good parallel efficiency at 10^6 concurrency levels in order to improve end-user productivity.

6. Acknowledgements

We would like to thank Dr. David Marcum, of the Center of Advanced Vehicular Systems at Mississippi State University, for acting as our consultant for any questions we had regarding the modifications of AFLR. This research was sponsored by NASA's Transformational Tools and Technologies Project (grant no. NNX15AU39A) of the Transformative Aeronautics Concepts Program under the Aeronautics Research Mission Directorate. This work in part is funded by the Virginia Space Grant Consortium (VSGC) Graduate Research Fellowship and NSF grant no. CCF-1439079.

7. References

[1] D. Marcum and N. Weatherill, "Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection," *AIAA Journal*, pp. 1619-1625, 1995.

[2] N. Chrisochoides, "Telescopic Approach for Extreme-scale Parallel Mesh Generation for CFD Applications," in *AIAA Aviation*, 2016.

[3] N. Chrisochoides, A. Chernikov, A. Fedorov, A. Kot, L. Linardakis and P. Foteinos, "Towards Exascale Parallel Delaunay Mesh Generation," in *18th International Meshing Roundtable*, Salt Lake City, UT, 2009.

[4] A. Chernikov and N. Chrisochoides, "Parallel Guaranteed-quality Delaunay Uniform Mesh Refinement," *SIAM Journal on Scientific Computing*, vol. 28, no. 5, pp. 1907-1926, 2006.

[5] A. Chernikov and N. Chrisochoides, "Practical and Efficient Point-insertion Scheduling Method for Parallel Guaranteed-quality Delaunay Refinement," in *18th ACM International Conference on Supercomputing*, 2004.

[6] A. Chernikov and N. Chrisochoides, "Three-dimensional Delaunay Refinement for Multi-Core Processors," in *22nd ACM International Conference on Supercomputing*, Island of Kos, Greece, 2008.

[7] N. Chrisochoides, A. Chernikov, T. Kennedy, C. Tsolakis and K. Garner, "Parallel Data Refinement Layer of a Telescopic Approach for Extreme-scale Parallel Mesh Generation for CFD Applications," in *AIAA Aviation 2018 (Accepted)*, Atlanta, Georgia, 2018.

[8] K. Garner, T. Kennedy and N. Chrisochoides, "Integration of Parallel Data Refinement Method with Advancing Front Local Reconnection Mesh Refinement Software," in *Virginia Space Grant Consortium (VSGC) 2017 Student Research Conference*, Williamsburg, Virginia, 2017.

[9] P. Thomadakis, C. Tsolakis, K. Vogiatzis, A. Kot and N. Chrisochoides, "Parallel Software Framework for Large-scale Parallel Mesh Generation and Adaptation for CFD Solvers," in *AIAA Aviation 2018 (Submitted)*, Atlanta, Georgia, 2017.