On the End-User Productivity of a Pseudo-Constrained Parallel Data Refinement Method for the Advancing Front Local Reconnection Mesh Generation Software

Kevin Garner[1], Polykarpos Thomadakis[2], Christos Tsolakis[3], Thomas Kennedy[4], and Nikos Chrisochoides[5]

Center for Real-time Computing, Old Dominion University, Norfolk, VA 23529, USA

Preliminary results of a long-term project entailing the parallelization of an industrial strength sequential mesh generator, called Advancing Front Local Reconnection (AFLR), are presented. AFLR has been under development for the last 25 years at the NSF/ERC center at Mississippi State University. The parallel procedure that is presented is called Pseudo-constrained (PsC) Parallel Data Refinement (PDR) and consists of the following steps: (i) use an octree data-decomposition scheme to divide 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. Parallelism was achieved by implementing Pseudo-constrained Parallel Data Refinement AFLR (PsC.AFLR) on top of a runtime system called PREMA. During run time, the PsC.AFLR method exposes data decomposition information (number of subdomains waiting to be refined) to the underlying runtime system. In turn, this system facilitates work-load balancing and guides the program's execution towards the most efficient utilization of hardware resources. Preliminary results, on the mesh refinement operation, show that the end-user productivity (measured in terms of elements refined per second) increases as the number of cores in use are increased. When using approximately 16 cores, PsC.AFLR outperforms the serial AFLR code by about 2.5 times. PsC.AFLR also maintains its stability by generating meshes of comparable quality. Although it offers good end-user productivity, PsC.AFLR suffers in its capability to generate meshes with the same level of density or quality as that of the serial AFLR software due to the constraints set by subdomain boundaries that are required to successfully execute AFLR. These constraints demonstrate that it is not ideal to use AFLR in a black box manner when parallelizing the software. Its source code must be modified to a non-trivial extent if one wishes to remove these constraints and maximize the end-user productivity and potential scalability. Future work includes a PDR implementation similar to work done previously in the parallelization of the TetGen meshing software and the integration of PDR.AFLR in the context of a Telescopic Approach meant to achieve scalability in a large-scale framework.

[1] Research Assistant, Center for Real-time Computing, AIAA Member.
[2] Research Assistant, Center for Real-time Computing, AIAA Member.
[3] Research Assistant, Center for Real-time Computing, AIAA Member.
[4] Lecturer, Center for Real-time Computing.
[5] Richard T. Cheng Chair Professor of Computer Science, AIAA Member.

I. Introduction

Mesh generation software is used in many industries where high fidelity simulations are required, such as in health-care, defense, and aerospace. For the last 25 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), which is one of the top, industrial strength, mesh generators [1]. AFLR has not been parallelized to utilize large-scale supercomputing hardware due to the geometric and numerical challenges imposed by the nature of mesh generation complexity. The Center for Real-time Computing (CRTC) at Old Dominion University (ODU) has proposed the telescopic approach (see Fig. 1) [2] [3], a framework that will leverage the concurrency 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. The long-term goal of this effort is to integrate AFLR with the PDR layer, which in turn will be implemented on top of the PO layer in future efforts.

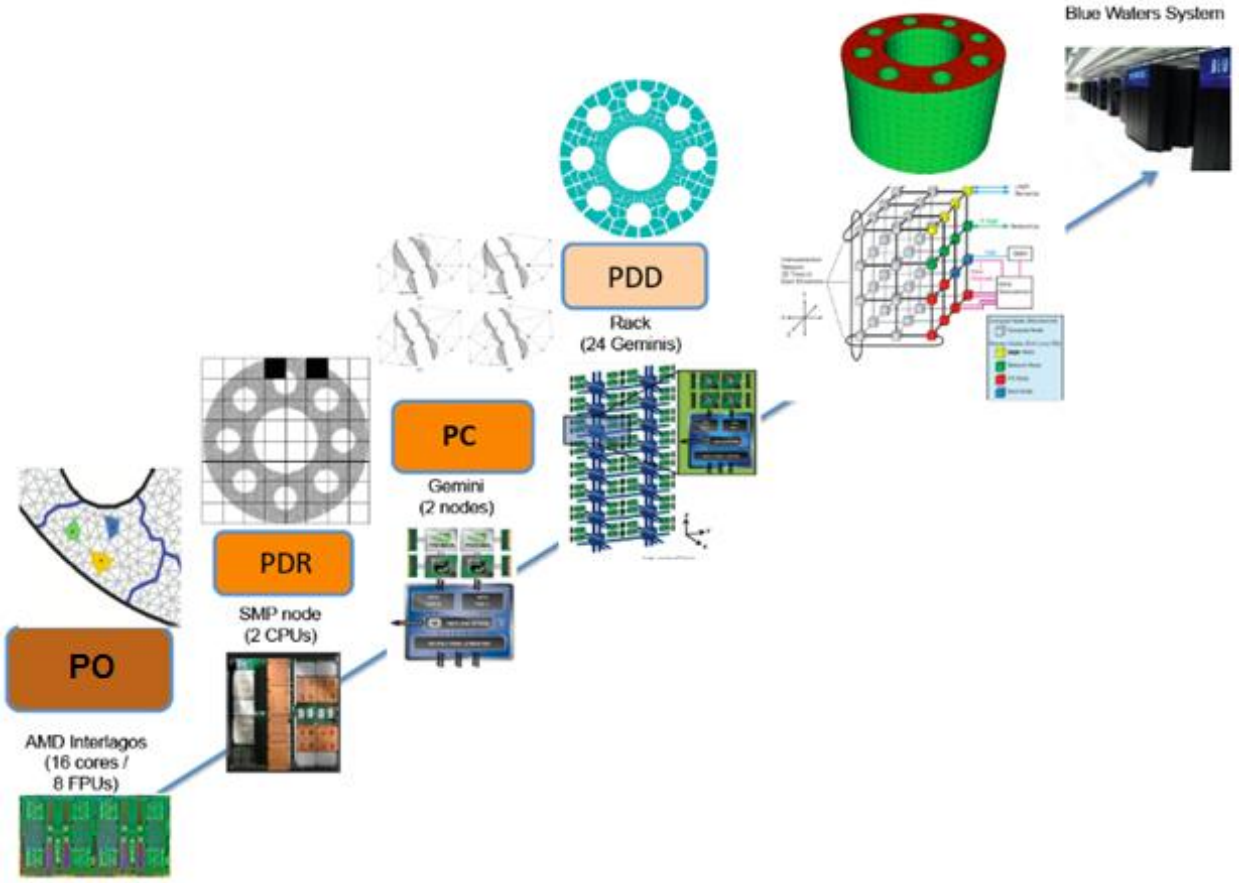

Fig. 1 Telescopic Approach to Parallel Mesh Generation. It covers the complete spectrum of hardware that spans from the Chip (bottom left) to a complete machine like Blue Waters (top right). The PO level is on the left and targets the chip level. The PDR approach is second from the left and targets hardware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

PDR maintains a fixed level of concurrency while parallelizing the refinement process. Its methodology is theoretically proven to maintain stability and robustness for parallel isotropic Delaunay-based mesh generation and has been experimentally verified for isotropic meshes [4] [5] [6]. It is also designed to allow for the utilization of any sequential mesh generator while offering guarantees for the following five requirements for parallel mesh generation [7]:

1) **Stability** ensures that a mesh generated in parallel maintains a level of quality comparable to that of a sequentially generated mesh. This quality is defined in terms of the density and shape of the elements evaluated in the metric field, and the number of the elements (fewer is better for the same level of metric conformity).
2) **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.
3) **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.
4) **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.
5) **Reproducibility** requires that the sequential mesh generator, when executed with the same input, produces either identical results (termed *Strong Reproducibility*) or those of the same quality (*Weak Reproducibility*) under the following modes of execution: (i) continuous without restarts, and (ii) with restarts and reconstructions of the internal data structures. 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 [8] 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. 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. The general idea of PDR is to concurrently refine the octree leaves while maintaining mesh conformity. The main concern when 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 (AFLR in the implementation presented here) on the subdomain within that leaf. Figure 2 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).

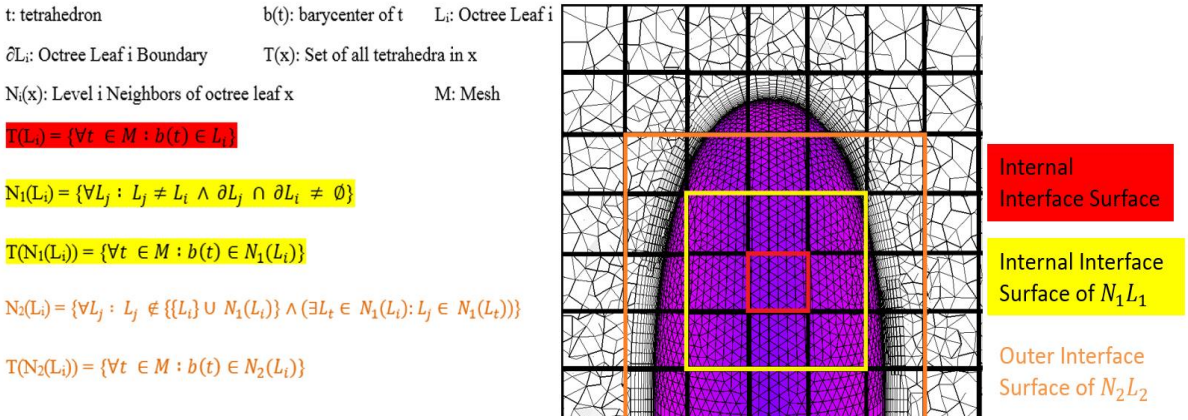

Fig. 2 Shown is a 2-D example of the upper portion of a data-decomposed rocket mesh where the red-boxed leaf is the primary leaf under refinement. The level 1 neighbors are those inside the yellow box (excluding the red leaf) and the level 2 neighbors are those inside the orange box (excluding all leaves 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).

The level 1 neighbors of a leaf are considered to be the buffer zone of that leaf (again, no leaf in the buffer zone may undergo refinement while the primary leaf undergoes refinement).

AFLR accepts an input geometry with an established boundary triangulation. A Delaunay-based method 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 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 active elements. 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 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, to satisfy a desired quality criterion. A min-max (minimize the maximum angle) criterion is primarily used which maximizes the minimum element edge weight, thereby producing high overall grid quality and eliminating most field sliver elements. All active elements undergo a final optimization phase, which consists of three quality improvement passes (sliver removal, grid coordinate smoothing, and further reconnection) [1].

II. Integration of AFLR with PDR

Ideally, the sequential mesh generator should be considered a black box when integrating it with PDR; however, AFLR has a number of requirements which not only impose constraints on PDR, but also require that modifications be made within the AFLR code itself. Due to the constraints set on PDR (which will be further explained later), this implementation will henceforth be referred to as Pseudo-constrained Parallel Data Refinement AFLR (or PsC.AFLR). In order to make the necessary modifications and properly integrate AFLR into the parallel framework, an intricate understanding of the underlying data structures and methodologies used for the initial grid generation, point insertion, element edge swapping, and optimization was required. The following steps outline the general process of PsC.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. Extract a 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 (executed in parallel).
11. Merge all data and call AFLR to perform final optimization.
12. Output the final mesh.

After an octree is constructed, data are assigned to octree leaves based on element barycenter. If the barycenter for an element falls within a leaf, that element is assigned to that leaf. Once data assignment is complete, it is possible that some leaves will have disconnected partitions (such as in Fig. 3), that is, elements (or groups of elements) connected to another element (or group) by only a point or edge. This is not acceptable for AFLR. Every tetrahedron must be face-connected to another (assuming that there is more than one tetrahedron of which AFLR is attempting to refine); otherwise, the results generated can vary greatly and will most definitely be incorrect (such as that on the right in Fig. 3). A method was created which checks for this problem before the refinement of a subdomain. If any disconnected tetrahedron is found, neighboring leaves are examined to find a tetrahedron with a matching face to the disconnected tetrahedron. Once found, the tetrahedron in question is reassigned to that particular leaf and removed from the leaf that is about to undergo refinement. This process is repeated until all connectivity issues are resolved.

In PsC.AFLR, a boundary must be created for a subdomain before any refinement of its elements can begin. This introduces overhead to the overall process due to the requirement that a smooth, simply-connected surface must be created for AFLR. The set of tetrahedra within a leaf is examined in isolation (as if the 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, while its neighboring tetrahedra were assigned to the primary 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 manifold boundary, acceptable for AFLR, is extracted.

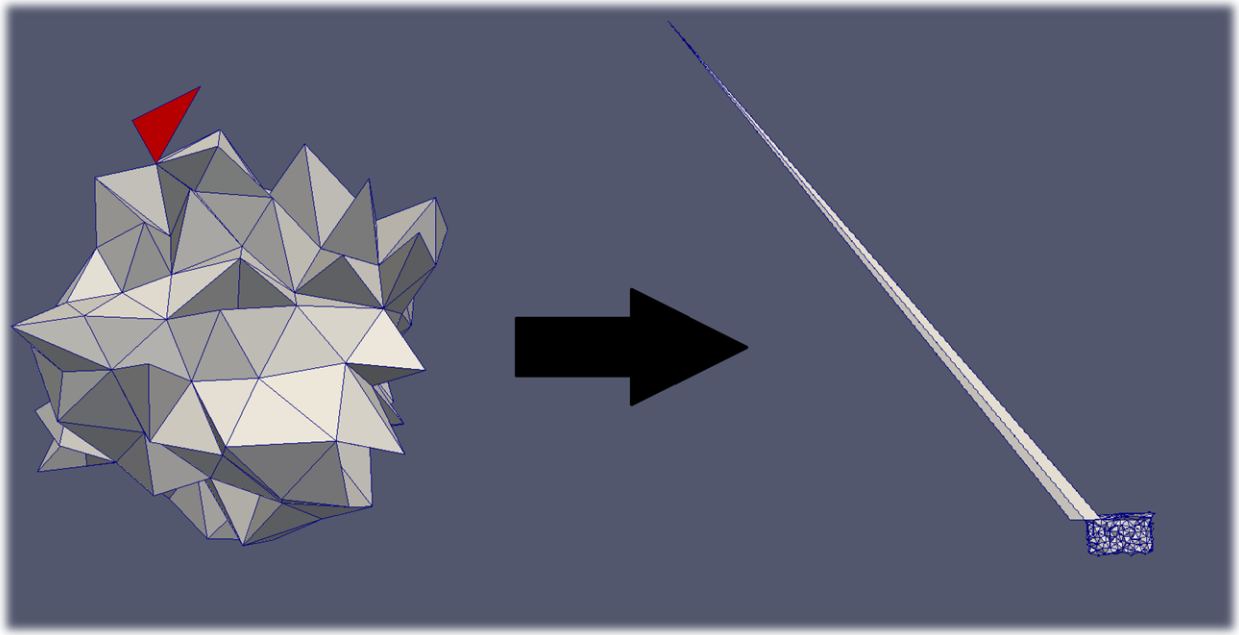

Fig. 3 The left side shows a subdomain of tetrahedra with one tetrahedron (in red) vertex-connected to the others (therefore considered to actually be "disconnected"). The right side shows a result generated after refining this subdomain (including the vertex-connected tetrahedron) in AFLR.

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 internal interface surface is 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 between level 1 and level 2 neighbors remains frozen, so as to eliminate the need of updating level 2 neighbors during refinement (maintaining PDR's original method of concurrency). It is possible to have duplicate points when merging the two sets of data (leaf and its level 1 neighbors) because a neighboring leaf may contain a tetrahedron that has a point located in the primary leaf, or vice-versa. If each set of data were examined in isolation, both sets would contain the same internal interface points (due to the fact that they are subdomains which conform to one another). When these sets of data are merged, the duplicate points are removed and any tetrahedron or triangle that references these points are updated to use the same indices (all tetrahedra and faces use integer-based indices to denote which points they contain, so if two tetrahedra contain the same point, they will use the same index to reference that point). 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 undergone swapping, a tetrahedron will 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 leaf, that contains a tetrahedron with a matching face, and that is a level 1 neighbor of both the level 2 leaf and of the primary leaf.

A function was also added to AFLR which 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 function to be optimized sequentially (so that PsC.AFLR performs a final optimization step over the entire domain just as serial AFLR does).

Parallelism was achieved by fully integrating PsC.AFLR onto a runtime system called the Parallel Runtime Environment for Multi-computer Applications (PREMA) 2.0 [9]. PREMA 2.0 is a parallel runtime system developed to support adaptive and irregular applications. It is capable of running in both shared and distributed memory. PREMA provides a globally addressable name-space, message forwarding, and data migration capabilities by using constructs called mobile objects and mobile pointers. Mobile objects are user-defined data objects that may encapsulate data not residing in contiguous memory (a leaf and its level 1 neighbors). A mobile pointer is a unique identifier created for each mobile object that can be used by the system even if the object has migrated to different ranks. This enables a rank to send a message to a specific mobile object and execute a user-defined function on it, regardless of its location. In PsC.AFLR, a master-worker model is used, where steps 1-4 of the above outline are executed by the master thread and steps 5-9 are executed by worker threads in parallel. During run time, the PsC.AFLR method exposes data decomposition information (number of leaves/subdomains waiting to be refined in the queue) to the underlying run-time system. PDR essentially informs PREMA that a leaf may undergo refinement if it and its level 1 neighbors are not currently under use in another leaf's refinement process. The master thread will send a message to the corresponding mobile pointer (representative of the leaf and its set of neighbors that are ready for refinement), essentially informing PREMA to execute a refinement function given the mobile object's data. PREMA 2.0 monitors the load of the system and performs migration (of the leaf and neighbor data) to an available worker without interrupting execution. Communication and execution are separated into different threads to provide asynchronous message reception and instant computation execution at the arrival of new work requests.

III. Results

PsC.AFLR is executed and tested on the Turing cluster at ODU [10]. Each node runs Red Hat Enterprise Linux Server release 6.10 with 32 cores per node. Each CPU is an Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz with 128 GB of memory. GCC version 6.3.0 and MPICH version 3.1.3 are used for compilation and execution.

All data presented are from the refinement time of both applications (PsC.AFLR and serial AFLR) and does not include initial volume mesh generation time or end optimization time. These two processes have introduced challenges in the parallel implementation that will be addressed in future work. Figure 4 shows a geometry of a horn bulb that is used for testing with both PsC.AFLR and serial AFLR. The horn bulb geometry contains 1,062,042 surface elements. The number of tetrahedra within the initial volume mesh used were 3,773,233 and 1,655,568 for PsC.AFLR and serial AFLR, respectively. The number of elements within the initial mesh for PsC.AFLR are greater due to preprocessing requirements for data decomposition. The nature of these requirements and their effect on the final mesh is explained later (and will be further explored in the future). The octree used in PsC.AFLR is at level 4 (containing 4,096 leaves, or subdomains). The final number of tetrahedra generated for PsC.AFLR is 13,006,043 and 116,130,365 for serial AFLR. Serial AFLR's refinement time is 16,101.51 seconds and its refinement speed is 7,212.39 elements/sec. Table 1 shows the performance achieved by PsC.AFLR, in addition to its refinement speed for each number of cores used.

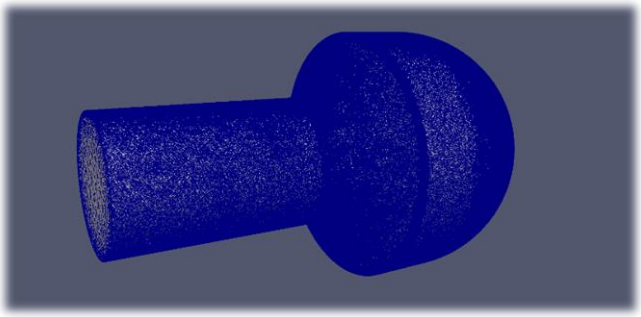

Fig. 4 Horn Bulb Geometry with 1,062,042 Surface Elements

Table 1 PsC.AFLR Performance Data and Refinement Speed based on the horn bulb geometry defined by 1,062,042 surface elements where PsC.AFLR generates about 13 million tetrahedra and serial AFLR generates about 116 million tetrahedra

# of Cores	1		2	4	8	16	32	64
Runtime (sec)	16,101.51 (for serial AFLR)	6,185.53	3,144.18	1,626.48	1,040.45	749.88	681.74	654.03
Refinement Speed (Elements per sec)	7,212.39 (for serial AFLR)	2,102.66	4,136.55	7,996.44	12,500.40	17,344.17	19,077.72	19,886

Based on the results in table 1, we observe that the end-user productivity increases as the number of cores are increased and outperforms serial AFLR (in both total runtime and refinement speed). PsC.AFLR is capable of generating a larger number of elements per second than the serial code on just 4 cores. It also outperforms serial AFLR by about 2.5 times on 16 cores. Although PsC.AFLR produces a final mesh with fewer elements than the serial software, its final mesh maintains satisfactory quality in comparison as shown in Fig. 5. The minimum dihedral angle of an element in the final mesh is 3.47 degrees while the maximum is 172.58 degrees with PsC.AFLR (as opposed to serial AFLR having a minimum of 7.3 degrees and a maximum of 164.57 degrees). The quality of the mesh generated by PsC.AFLR is within operational limits of CFD solvers such as FUN3D and SU2 [11] [12]. The final number of elements between the two applications is different for two reasons:

1) PsC.AFLR extracts a surface triangulation for a subdomain based on the initial volume elements and these surface elements remain frozen throughout the refinement of that particular leaf. The surface elements undergo local reconnection once the newly refined leaf data is merged with the elements within the surrounding level 1 neighbor leaves, but they do not undergo point insertion (which would create more elements) in order to maintain conformity between the subdomains.
2) Serial AFLR was run in its default state (no parameter adjustments). This is not satisfactory for PsC.AFLR when refining subdomains because it uses an initial mesh generated by TetGen (due to initial volume mesh generation challenges with AFLR) and therefore utilizes the point distribution of the subdomain surfaces (from the coarse TetGen mesh) rather than the external dense surface of the geometry (which serial AFLR uses). This point distribution directly affects the number of elements created. AFLR's point distribution function was abstracted to be used before refinement within PsC.AFLR so that it can utilize this value based on the external surface. The point distribution is explicitly set to this value as a parameter for all subdomains. Henceforth, PsC.AFLR is able to generate a denser mesh of higher quality although it extracts coarse surfaces based on its initial TetGen-generated mesh. Even with this adjustment though, these constraints do not allow PsC.AFLR to generate a mesh as dense as that generated by serial AFLR; however, PsC.AFLR still maintains satisfactory quality.

The final meshes generated by PsC.AFLR and serial AFLR cannot easily be compared because the serial AFLR-generated mesh is about 9 times larger. AFLR does not perform as well within the context of PsC.AFLR due to the constraints set by its own requirements. AFLR's boundary requirement reduces the number of elements that AFLR creates in PsC.AFLR due to the coarse boundary extracted from the TetGen-generated mesh. Adjusting the point distribution improves the subdomain density, but ultimately does not allow PsC.AFLR to generate a mesh as dense as serial AFLR, which also affects the final dihedral angle quality of its elements.

The percentage of runtime taken by PsC operations, AFLR, PREMA (data dissemnination decisions), and load balancing (packing, unpacking, and migration of data) requires further study, as the isolation of runtime for each of these processes presents some challenge since they are executed in parallel. The average time for each PsC operation was gathered, compared to the total time spent executing PsC-specific operations, in Fig. 6. This data was gathered by taking the average times among the master and worker processes for each of 100 runs, executed for each number of cores (100 runs for 1 core, 100 runs for 2 cores, etc.). L2 Tet Removal and Addition reference the temporary removal of tetrahedra from a subdomain that contain level 2 points, and the later re-assignment of these tetrahedra back to their respective leaves after refinement. This is necessary as level 1 neighbor boundary elements may contain points that are assigned to level 2 neighbors, in which case these points are not packed and migrated to the node where the refinement process is about to begin. Only a leaf and its level 1 neighbor data (that is, data within level 1 neighbor

leaves) are packed for migration, which follows PDR's method of concurrency. The temporary removal of these tetrahedra is acceptable because these elements have either already undergone local reconnection (since they are boundary elements) or will do so in a later refinement process for another leaf. Partition Swap represents the time spent detecting disconnected partitions and re-assigning them to neighboring leaves. This is the most time-consuming PDR operation. Neighbor Extraction is the time spent merging all level 1 neighbor data into a single set of data to be passed as a parameter to AFLR and Data Assignment represents the time spent assigning data to octree leaves after refinement.

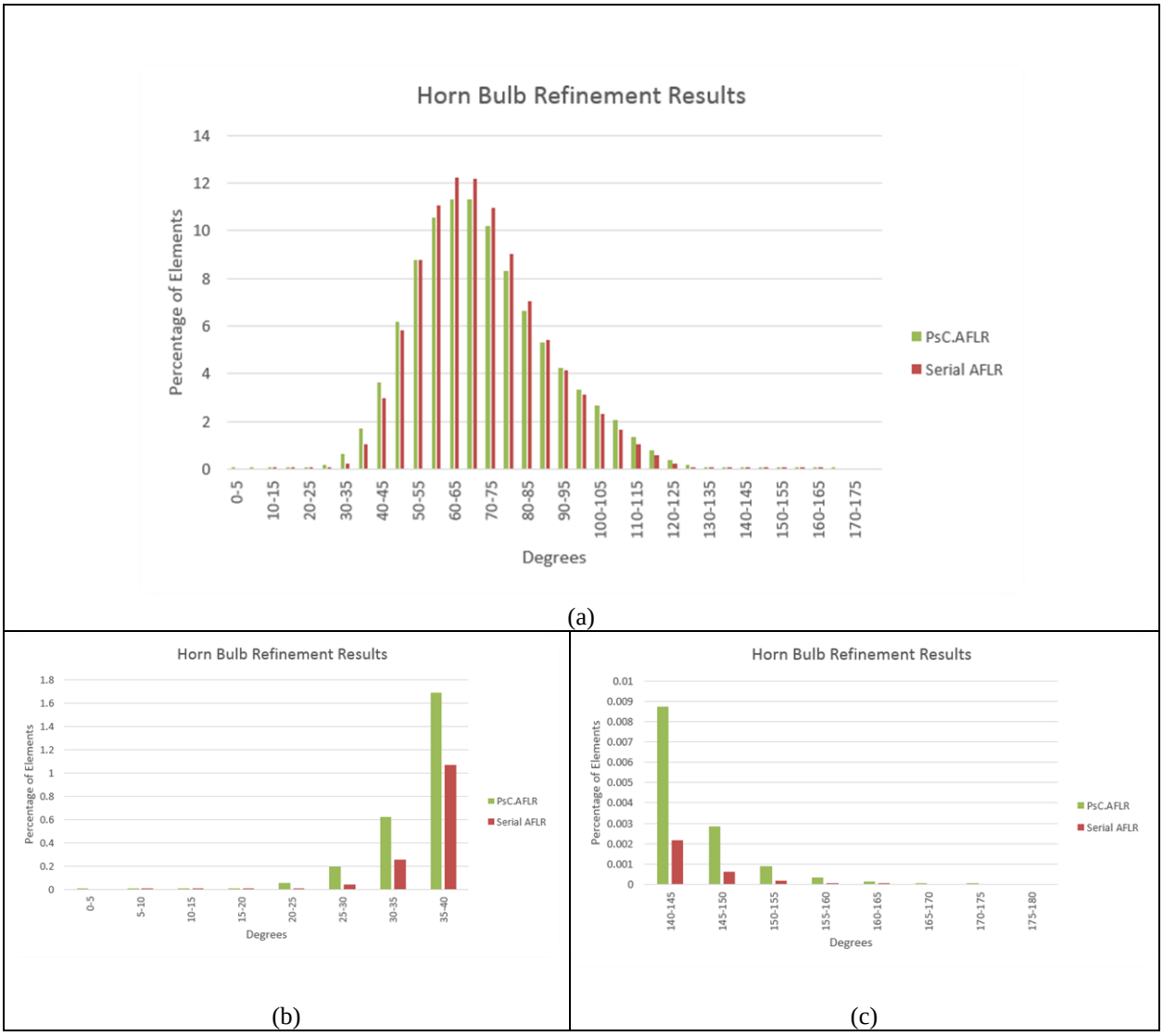

Fig. 5 The dihedral angle statistics of the final meshes generated by PsC.AFLR and serial AFLR are shown and compared in (a). (b) and (c) show the lower (0 to 40 degrees) and upper (140 to 180 degrees) dihedral angle statistics, respectively.

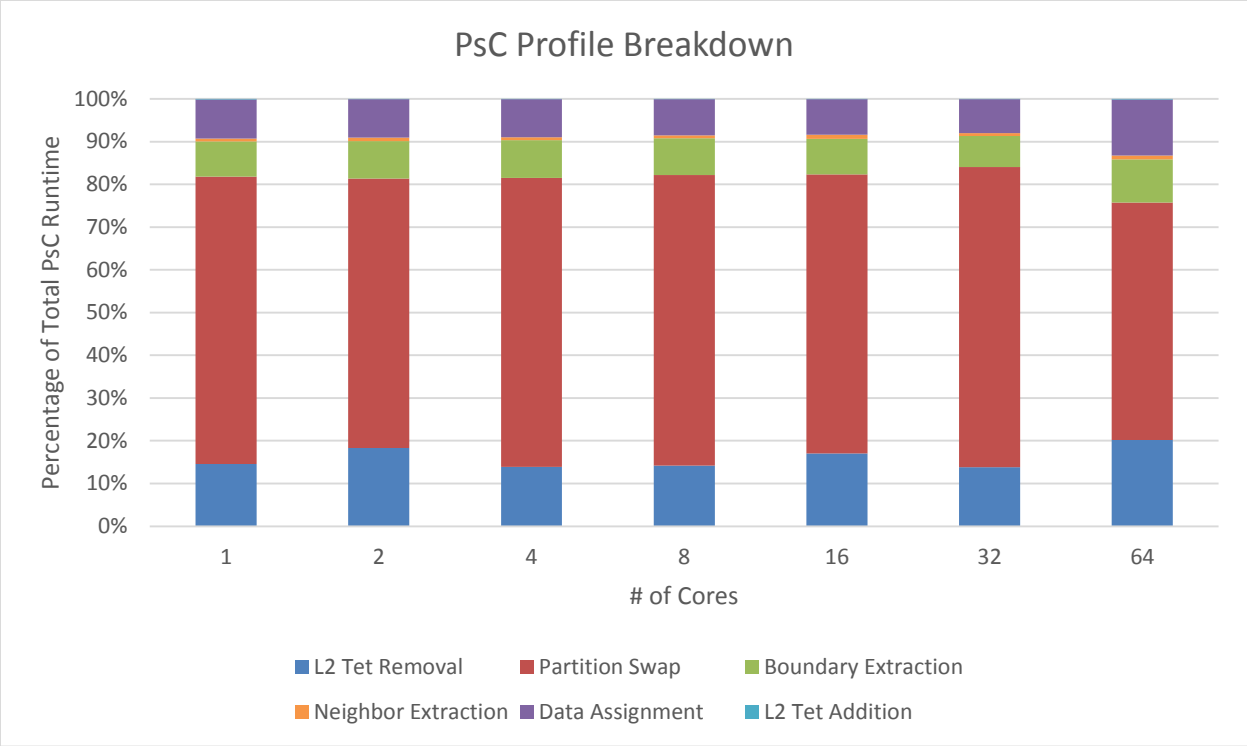

Fig. 6 A breakdown of PsC operations is presented. The average percentage of time spent in each operation in reference to the total amount of time spent performing PsC operations is shown.

IV. Future Work

There is still much to be accomplished in order to create a PDR.AFLR implementation. Based on past work and results [7], an unconstrained parallel data refinement implementation with AFLR is expected to be much faster and efficient than the pseudo-constrained PDR.AFLR. This will require a significant time investment, as many fundamental changes will need to be made to the source code. Successfully implementing these changes would eliminate several pseudo-constrained PDR processes, which would likely significantly increase end-user productivity.

The initial volume mesh is currently generated by TetGen for PsC.AFLR. PsC.AFLR requires an initial volume mesh that is dense enough to satisfy boundary requirements of individual octree leaves. Again, AFLR requires a smooth, simply-connected boundary when refining a domain. If the mesh is too coarse, there may be a tetrahedron that spans multiple leaves. In this scenario, no boundary can be extracted for each leaf if a face is spanning across them all. One solution would be to reduce the octree level (increasing the size of the leaves/subdomains) but this would reduce the overall number of leaves/subdomains, thereby reducing the amount of achievable concurrency. In order to maintain parallelism, the initial mesh must be dense enough so that faces can be extracted for every leaf containing tetrahedra. A problem observed with AFLR is that it does not always generate tetrahedra within a certain volume constraint unless the mesh undergoes a significant amount of refinement. This is counterintuitive for the purpose of generating an initial mesh. If the initial mesh undergoes too much refinement (in order to satisfy the density requirement), then more runtime will be spent at this stage rather than in parallel refinement (sometimes taking hours for larger geometries) and rendering the parallel refinement futile. TetGen is currently being used to generate an initial mesh due to its low runtime and because a volume constraint can be set easily when refining a geometry. The caveat, as described earlier, is that the final mesh generated by PsC.AFLR will not be as dense as that generated by serial AFLR and while the quality is satisfactory, it is not as good as that generated by the serial code. More research and collaboration with Dr. David Marcum (original developer of AFLR), of Mississippi State University, is required to establish a more reliable method of generating the initial mesh. This problem may be eliminated completely if the aforementioned modifications are made to AFLR for an unconstrained PDR.AFLR implementation (eliminating the need for a dense initial mesh if boundaries do not need to be extracted for each leaf).

Another methodology must be developed to further refine subdomain boundary elements (actual point insertion) after they are frozen during leaf refinement (since these elements are extracted from the initial mesh and

play a significant role in the density of a subdomain). There are similar methods in other mesh generation programs, such as *EPIC* from Boeing and *refine* from NASA, which freeze boundary elements during subdomain refinement to maintain mesh conformity [13]. These elements are later shifted between subdomains for refinement. The methods used to shift and refine these elements will be examined so that a similar approach may be applied to PDR.AFLR.

As seen in Fig. 5, the partition swapping process takes a significant amount of time in comparison to the other PsC operations. This process of assigning data based on element barycenter will be modified to take advantage of a mathematically proven optimization-based geometry partitioning algorithm, which will eliminate the creation of disconnected partitions [14].

PsC.AFLR currently 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 PDR.AFLR to process geometries with transparent/embedded surfaces. An embedded surface must remain frozen during refinement, and because there may a volume on both sides of the surface, both sides can be considered as two separate subdomains. This is similar to the disconnected volume problem of a mesh with a genus greater than zero. Leaves with embedded surfaces will need to be partitioned further into additional subdomains, and these subdomains will be refined independently. Many geometries, such as meshes with turbulent flow around a rocket or missile, typically contain transparent/embedded surfaces. Once this functionality has been implemented, PDR.AFLR will be capable of refining these types of meshes. Code Re-use will be addressed by further expanding 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 also be improved by parallelizing the final optimization process. Additional timing functionality will be implemented so that an in-depth performance analysis may be gauged for PsC, AFLR, PREMA, and load balancing operations. This will also assist in future optimization efforts.

V. Conclusion

Due to the requirements and constraints set by AFLR, it is not possible to simply use it in a black box manner when attempting to parallelize the software. The point distribution function was abstracted in order to allow PsC.AFLR to generate dense meshes of good quality. Modifications were made to AFLR so that it would accept a second set of data (level 1 neighbors of a leaf) so that leaf boundary elements may undergo swapping and improve their quality since they remain frozen during point insertion. One must consider the overhead introduced simply from preparing a subdomain of data for refinement. Partition reassignment, boundary extraction, data merging (leaf and level 1 neighbors), and data assignment (prevention of tetrahedron assignment to level 2 neighbors in order to maintain concurrency) all play a significant role in both the success of refinement (generating correct results, i.e. a conforming mesh with good quality) and in the runtime. Although they produce overhead on the runtime, these operations are essential when parallelizing AFLR if one wishes to minimize the modifications needed for AFLR. PsC.AFLR has good end-user productivity given its refinement speed and is able to outperform serial AFLR with just 4 cores and by about 2.5 times on 16 cores. Still, a pressing issue is that meshes generated by PsC.AFLR are much less dense than those generated by serial AFLR. The two programs essentially produce two different final volume meshes, regardless of attempting to run AFLR within PsC.AFLR using the same settings as serial AFLR (point distribution). The aforementioned operations, although necessary, constrain the full capabilities of AFLR to generate dense meshes of high quality.

More work is needed to complete the PDR.AFLR implementation, including several additions to maintain the same level of functionality as the serial code and optimizations needed to improve the scalability. By the completion of this project, the aforementioned pseudo-constraints will have been removed and 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, further improving end-user productivity.

VI. Acknowledgement

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 Southern Regional Education Board (SREB) Doctoral Fellowship, Virginia Space Grant Consortium (VSGC) Graduate Research Fellowship, and NSF grant no. CCF-1439079.

VII. 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*, Atlanta, Georgia, 2018.

[8] H. Si, "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator," *ACM Transactions on Mathematical Software*, vol. 41, no. 11, 2015.

[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 Forum 2018*, Atlanta, 2018.

[10] Old Dominion University, "High Performance Computing," 15 September 2018. [Online]. Available: https://www.odu.edu/facultystaff/research/resources/computing/high-performance-computing. [Accessed 13 May 2019].

[11] R. Biedron, J.-R. Carlson, J. Derlaga, P. Gnoffo, D. Hammond, W. Jones, B. Kleb, E. Lee-Rausch, E. Nielsen, M. Park, C. Rumsey, J. Thomas, K. Thompson and W. Wood, "FUN3D Manual: 13.4," NASA, October 2018. [Online]. Available: https://fun3d.larc.nasa.gov/papers/FUN3D_Manual-13.4.pdf. [Accessed 7 May 2019].

[12] T. Economon, F. Palacios, S. Copeland, T. Lukaczyk and J. Alonso, "SU2: An Open-Source Suite for Multiphysics Simulation and Design," *AIAA Journal*, vol. 54, no. 3, 2015.

[13] C. Tsolakis, N. Chrisochoides, M. Park, A. Loseille and T. Michal, "Parallel Anisotropic Unstructured Grid Adaptation," in *AIAA SciTech Forum 2019*, San Diego, 2019.

[14] N. Chrisochoides, C. Houstis, E. Houstis, S. Kortesis and J. Rice, "Automatic Load Balanced Partitioning Strategies," Purdue University, West Lafayette, 1989.