Fine-grained Speculative Topological Transformation Scheme for Local Reconnection Methods

Fotis Drakopoulos, Christos Tsolakis and Nikos P. Chrisochoides
Old Dominion University, Computer Science Department, Norfolk, VA

A parallel method for topological transformations for local reconnection methods is presented. The proposed scheme combines known parallel techniques like data over-decomposition and load balancing with widely used topological transformations also known as flips or swaps. In contrast to most mesh generation methods, the proposed optimizes the connectivity in parallel throughout the mesh generation procedure. The speculative scheme is evaluated on a variety of aerospace configurations. Early results indicate that the high quality and performance attributes of this method see substantial improvement over existing state-of-the-art technology.

I. Introduction

The long term goal of this project is to achieve extreme-scale adaptive CFD simulations on the complex, heterogeneous High Performance Computing (HPC) platforms. To achieve this goal, a telescopic approach (see Figure 1) is proposed in [1, 2]. The telescopic approach is critical in leveraging the concurrency that exists at multiple levels in parallel anisotropic and adaptive simulations. At the chip level the telescopic approach deploys a Parallel Optimistic (PO), sometimes here and elsewhere is also called speculative approach, which explores concurrency at a fine-grain (element) level using data decomposition. In this paper, we focus on the implementation of the speculative layer, having however in mind that the code should be modular enough in order to be used throughout the telescopic approach.

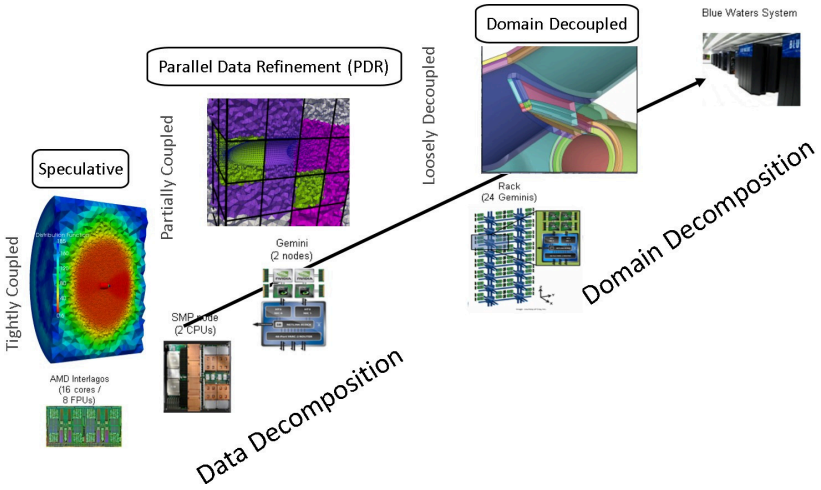


Fig. 1 Telescopic Approach for Extreme-Scale Parallel Mesh Generation [2]

The mesh generation kernel of the telescopic approach for CFD applications is essentially a combination of Advancing Front type point placement, direct point insertion, and parallel multi-threaded connectivity optimization schemes. The proposed speculative method repetitively reconnects the mesh using tightly-coupled topological transformations. Simple hill-climbing is employed to maximize the quality of the worst local element. The local reconnection scheme is based on: (i) data over-decomposition, (ii) atomic operations to avoid data races and maintain a valid mesh throughout the procedure, and (iii) load-balancing to redistribute work-units among the threads.

The proposed method and its implementation is designed to be a module of the CDT3D mesh generation software which focuses on five important aspects of parallel mesh generation and adaptation:

1) **Stability**. The quality of the mesh generated in parallel must be comparable to that of a mesh generated sequentially. The quality is defined in terms of the shape of the elements (using a chosen space-dependent metric) and the number of the elements (fewer is better for the same shape constraint).

2) **Robustness**. The ability of the software to correctly and efficiently process any input data and to consistently generate high quality elements. Operator intervention into a parallel computation is not only highly expensive, but most likely infeasible due to the large number of concurrently processed sub-problems.
3) **End-User Productivity**. The ability of the software to process the input data faster compared to state-of-the-art codes. Scalability is defined as the ratio of the time taken by the best sequential implementation to the time taken by the parallel implementation. The speedup is always limited by the inverse of the sequential fraction of the software, and therefore all non-trivial stages of the computation must be parallelized to leverage the current multi-core architectures.
4) **Code Re-Use**. A modular design of the parallel software, such that it can be replaced and/or updated with minimal effort. Due to the complexity of mesh generation codes, this is the only practical approach for keeping up with the ever-evolving algorithms and computer architectures.
5) **Reproducibility**. Related to the stability notion is the one of reproducibility, were the mesh generator should first terminate and produce comparable quality no matter if it is initialized with a CAD- or image-driven model or a surface of a volume mesh generated by the mesher itself.

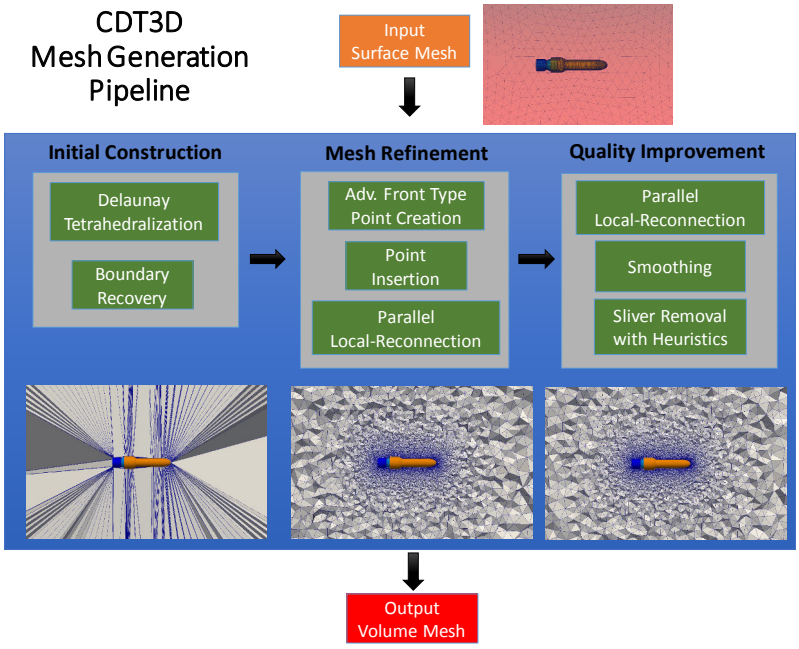


Fig. 2 High level mesh generation pipeline of the CDT3D software.

The focus in this paper is the generation of high quality isotropic meshes employing the speculative local reconnection approach. While boundary layer and metric-based anisotropic mesh generation are not supported yet, CDT3D's modular design allows for these methods to be integrated in the future.

CDT3D is inspired from state-of-the-art unstructured grid technology AFLR (Advancing Front-Local Reconnection) [3, 4] and is designed to be an alternative for industrial-strength extreme-scale parallel mesh generation. AFLR is directly incorporated in several industrial-strength systems. However, its software complexity, given that is designed and optimized to be super-efficient on single-core machines raised questions as to whether its parallelization can meet all five requirements stated above and it can be accomplished within the time constrains of its users.

Mesh generation codes typically improve the shape of the elements in a post-processing step, **on the contrary, the proposed method optimizes the connectivity throughout the generation procedure** to obtain a maximum quality. The connectivity is optimized using topological transformations coupled with a combined quality criterion : Delaunay in-sphere [5] and Min-Max type [6]. Topological transformations are fundamental operations in local reconnection. A topological transformation removes elements and replaces them with a different set of elements that occupy the same space. Transformations can be computationally expensive and time-consuming, mainly for two reasons: (i) a transformation does not always produce a geometrically valid connectivity, hence the orientation of the new elements needs to be verified, and (ii) a quality metric needs to be computed to decide whether the new connectivity is locally

optimal or not. However, transformations involving more than three elements may have more than one candidate solutions and the cost to compute the optimal solution is a-priori non-polynomial [7]. Also, a transformation does not always produce a geometrically valid connectivity, hence the orientation of the new elements needs to be verified. Experimental evaluation of the proposed method indicates that a sequential optimization with local reconnection accounts for at least 70% of the mesh generation time (without post-improvement).

For those reasons, CDT3D employs a parallel speculative local reconnection approach. This approach is characterized by intense communication and resolution of dependences at runtime. The low cost of communication allows the speculative approach to take advantage from dynamic load balancing schemes like the one presented in [8] that is used in this work.

II. Previous Work

Two approaches are typically used to generate a tetrahedral mesh from an input surface mesh: Delaunay or Advancing Front [9]. Delaunay methods can handle constrained polyhedral domains of arbitrary complexity with the use of heuristic mesh modification techniques [10, 11]. Other Delaunay approaches provide mathematically guarantees on the mesh quality but do not preserve the boundary triangulation [12, 13].

The first efforts on the parallelization of existing sequential Delaunay mesh generation algorithms are reported in [14, 15]. These methods are based on the parallelization of a Bowyer-Watson kernel [16, 17]. A tightly-coupled distributed parallel Delaunay refinement algorithm for simple polyhedral domains whose constituent bounding edges and surfaces are separated by angles between 90° to 270° is presented [18]. This algorithm, can create and place large meshes up to 6 times faster than the traditional approach (i.e., sequentially generate a sufficiently dense mesh, partition the mesh into submeshes, distribute the submeshes to the processors, and sequentially refine the mesh) to generating and refining a distributed mesh.

A Delaunay method that allows for safe insertion of points independently without synchronization is presented [19]. This method, based on a carefully constructed octree, splits the work-list of the candidate points up into smaller lists such that the available sequential codes can be used without modifications to process the sublists. The drawback is that the amount of work assigned to different processors can vary significantly. However, by using over-decomposition in a combination with runtime software system for dynamic load balancing [20], the work among the nodes can be redistributed.

In some cases (e.g., high-aspect ratio viscous meshes or three-dimensional meshes with sliver elements formed from four nearly coplanar points) the Delaunay criterion may not be the most suitable. For this reason, other approaches to improve the quality over that of a Delaunay mesh by using topological transformations have been introduced [6, 21–23].

Advancing Front methods offer the advantages of a high quality mesh due to optimal point placement and boundary integrity [24, 25]. Its main drawbacks are complexity and lower element quality when fronts collide in 3-dimensions. An Advancing Front mesh generator for shared memory architectures is presented in [26]. The domain to be meshed is first subdivided spatially using a coarse octree, and then the boxes are meshed in parallel. In [27], a coarse tetrahedral mesh is generated first to provide the basis of block interfaces and then partitioned into a number of subdomains using METIS [28] partitioning algorithms. A volume mesh is generated on each subdomain in parallel using an Advancing Front method, and all subdomains are later combined to create a single mesh. To remove the artifacts in the interfaces between subdomains, an angle-based node smoothing was used, but no topological changes (i.e., local reconnections) are introduced.

Many parallel Delaunay or Advancing Front algorithms have been proposed [8, 18, 19, 26, 29–32]. However, few parallel local reconnection algorithms have been presented in the literature. In [33] a parallel distributed Advancing Front mesh generation method with quality improvement is presented. Quality improvement includes a combination of several algorithms, i.e., diagonal swapping, removal of bad elements, node smoothing, and selective mesh movement. In the first pass of quality improvement, each submesh is reconnected by swapping edges in its interior, so the inter-processor boundary remains unchanged. In the second pass, a new partition is created by adding 1-2 extra layers of elements to each subdomain from the neighboring domains, the submeshes are redistributed among processors, and mesh improvement operations are performed again. However, it is unclear how the work load is balanced after re-partitioning.

A multi-threaded local reconnection and smoothing algorithm for mesh improvement is presented [34]. The parallelization of smoothing operations is based on an existing data-decomposition technique [35], which colors the dual graph of the mesh to subdivide the points into a few independent sets. The parallelization of local reconnection operation is based on a new data-decomposition technique, which defines a feature point in the interior of each local

reconnection operation and sorts the feature points along a Hilbert curve [36, 37]. The decomposition of this Hilbert curve results in an initial load-balanced distribution of local operations, i.e., edge removals from those low quality tetrahedra.

A two-step thread-parallel edge and face swapping algorithm is presented [38]. In the first step, a vertex locking strategy is introduced to select a maximal conflict-free set of edges and faces for swapping. In the second step, edge and face migration between threads is performed to balance the work-load in each thread, and then parallel edge and face swaps are applied without any interference. However, the evaluation of the 3-dimensional algorithm is limited.

Other approaches suggest a combination of smoothing and untangling operations [39, 40]. A log-barrier interior point method is developed to solve a smooth constrained optimization problem and untangle a mesh with inverted elements, improving its quality [41]. This method is parallelized for distributed memory machines using an edge-based coloring communication synchronization technique in which edges corresponding to a graph of communicating processes are colored to synchronize the communication [42].

A parallel optimization technique that smooths independent sets of vertices simultaneously is developed in [43]. This technique performs local vertex movement using a vertex-coloring scheme to avoid conflicting updates to vertex positions. The method is implemented for a parallel random access machine (PRAM) model and distributed memory architectures.

A parallel advancing front algorithm for distributed memory machines based on the advancing partition algorithm is presented in [44]. This method utilizes the advancing front method to generate separators that decouple the domain. The separators are build by generating and inserting points along imaginary partition planes. The generated subdomains are refined then in parallel with no synchronization.

An optimization-based smoothing algorithm for anisotropic mesh adaptivity is presented in [45]. The smoothing kernel solves a non-linear optimization problem by differentiating the local mesh quality with respect to mesh vertex position and employing hill-climbing to maximize the quality of the worst local element. The method is parallelized for hybrid OpenMP/MPI using standard coloring techniques.

Previous work on 3-dimensional Delaunay refinement [8] indicates that a speculative approach performs well on hardware-shared memory. Similarly to a Delaunay reconnection, roll-backs are possible during speculative reconnection with topological transformations, due to intersections of polyhedra that are attempted to transform concurrently. The proposed scheme employs atomic operations to avoid such intersections, thus maintaining a valid connectivity throughout the procedure. To exploit additional parallelism, the proposed method implements a load-balancer to migrate work-units (buckets) from busy threads to threads without work. The granularity of the decomposition (i.e., size of work-units) and the amounts of data for migration can be adjusted to achieve an optimum performance.

Details of the proposed algorithm are presented in the following sections along with an extensive evaluation on quality and scalability aspects of the code for 3-dimensional aerospace configurations.

III. Refinement

The proposed refinement algorithm is essentially a combination of Advancing Front type point placement, direct element subdivision, and parallel multi-threaded connectivity optimization schemes.

A. Procedure

During the refinement, new points are created and inserted into the mesh until the mesh satisfies a desired point distribution function. Points are generated using an Advancing Front type point placement and inserted by direct subdivision of the contained elements. Topological transformations are applied in parallel to reconnect the mesh.
An overview of the procedure is the following (see Figure 2)

1) Compute distribution function for each point on the boundary
2) Mark all tetrahedra as active (i.e., eligible for reconnection)
3) While new field points are accepted:
    a) Deactivate tetrahedra that satisfy the point distribution function
    b) Create new field points
    c) Insert field points
    d) Optimize connectivity

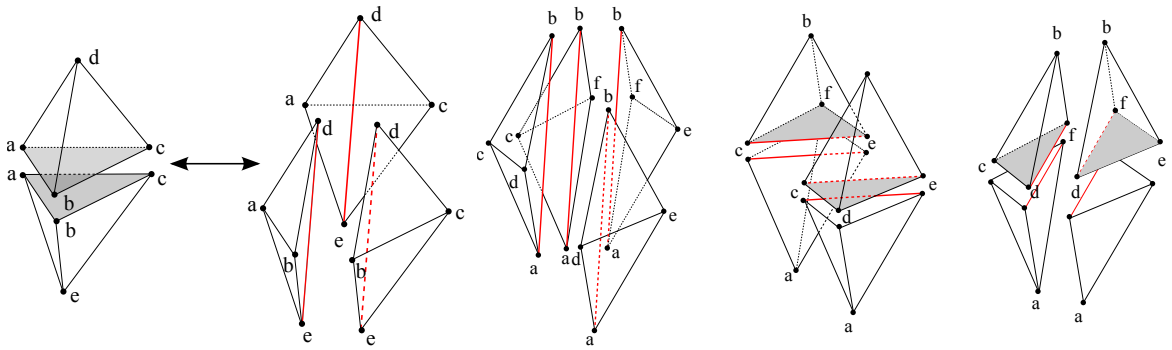
In this paper, the Parallel Connectivity Optimization is described in detail. Main operation of the connectivity optimization is the topological transformations.

IV. Topological Transformations

Topological transformations are necessary in the majority of mesh generation algorithms [3, 7, 22, 23, 35, 46, 47]. Flips or swaps are alternative terms often used in computational geometry literature [48]. A topological transformation modifies the mesh connectivity by replacing a set of elements with a different set of elements that occupy the same space. Topological transformations include a variety of operations such as edge/face flipping but also point insertion or point removal, which makes them a powerful tool for mesh generation since most mesh operations boil down to these basic operations. Topological transformations are typically used in conjunction with an objective function to optimize the mesh. Typical objective functions are the average quality of the elements or the quality of the worst element. The proposed method, implements various objective functions such as: (i) Delaunay empty sphere property [5], (ii) maximization of the minimum Laplacian edge weight [6], (iii) minimization of the maximum dihedral angle, and (iv) maximization of the minimum dihedral angle. The parallel reconnection step uses both the (i) Delaunay empty sphere criterion [5] and (ii) Maximization of the minimum Laplacian edge weight [6] to optimize the mesh. Transformations are also used to enforce a specific connectivity into the mesh without taking into account quality aspects. Such an example is the boundary recovery and the point insertion algorithm employed by CDT3D.

CDT3D implements also recursive $n-m$ flips [37] ($n \geq 3$, $m = 2n-4$). These flips are essentially a combination of $(n-3)$ 2-3 flips followed by a final 3-2 flip. The $n-m$ flip has been proven very effective in terms of robustness for edge removal. However, due to the fact that the number of candidate tetrahedralizations for n tetrahedra surrounding an edge increases exponentially in n, thus the cost to exhibit these tetrahedralizations is a priori non-polynomial [7], recursive $n-m$ flips are employed only during boundary recovery.

Point insertion is implemented using 4-1,6-2,n-2n flips for inserting a point in a tetrahedron, on a shared face and on a shared edge respectively. For performance reasons, the refinement and improvement modules reconnect clusters of up to $n = 4$ tetrahedra. When $n = 4$, the two candidate configurations are computed in advance and the one that maximizes the objective function is selected. During local reconnection three types of flips are used: 2-3, 3-2, 4-4. Flips 2-3 and 3-2 are used to reconnect five non coplanar points. A 2-3 flip removes a face from the mesh, while a 3-2 flip removes an edge (Figure 3(a)). A 4-4 flip interchanges two edges in a set of four tetrahedra surrounding an edge (Figure 3(b)). It is essentially a combination of a 2-3 flip (that inserts a new edge) and a 3-2 flip (that removes an old edge). The first 2-3 flip may temporarily create a flat tetrahedron which will be removed by the followed 3-2 flip. Two candidate configurations are computed in advance for a 4-4 flip. The configuration that optimizes an objective function is then selected.



(a) 2-3/3-2 flip (b) 4-4 flip

Fig. 3 3(a) A 2-3 flip removes a face (abc) from the mesh, creating a new edge (de). Flip 2-3 is geometrically valid if the edge de intersects face abc in the interior. The inverse operation removes an edge (de) from the mesh, creating a new face (abc). 3(b) 4-4 flip. Left: Initial configuration with four tetrahedra ($abcd$, $abde$, $abef$, $abfc$) surrounding an edge (ab). Middle: first alternative configuration with edge ab being replaced by edge ce. The new tetrahedra are: $ceda$, $cfea$, $cdeb$, $cefb$. Right: second alternative configuration with edge ab being replaced by edge df. The new tetrahedra are: $dcfa$, $dfea$, $dfcb$, $defb$.

V. Parallel Connectivity Optimization

Local reconnection with topological transformations has been proven very effective for mesh optimization [3, 7, 22, 23, 35]. Nevertheless, it can be the bottleneck for the performance of the mesh generation. The experimental evaluation of this study indicates that the sequential reconnection scheme accounts for about 80% of the total refinement time. The proposed parallel speculative approach reduces the overheads significantly. To the best of our knowledge, CDT3D is the first software that optimizes the connectivity using parallel tightly-coupled topological transformations throughout the mesh generation (including a post-improvement step).

A. Parallel Procedure

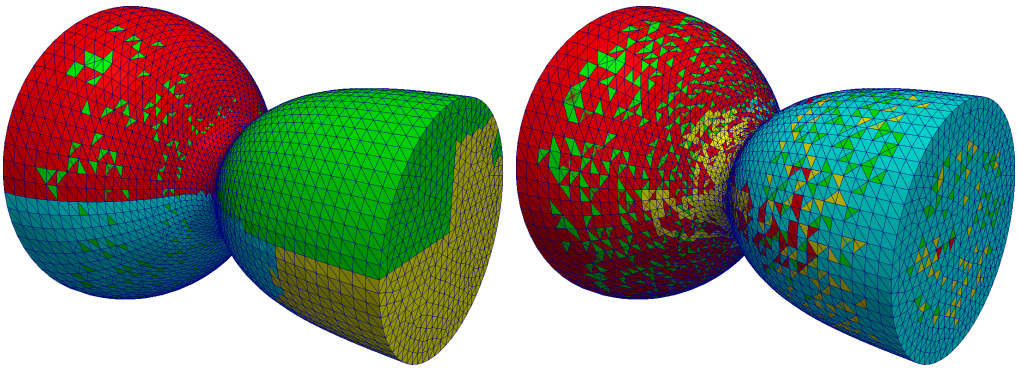
The connectivity is optimized using tightly-coupled topological transformations. The parallelization is based on:

1) Over-decomposition
2) Thread safe operations
3) Load balancing

1. Over-decomposition

The active (i.e. eligible for reconnection) elements of the mesh are grouped into equal-sized *work-units* (*buckets*) which are then distributed to the threads. The granularity (grain size) of the decomposition is adjusted with a parameter $nbuckets \in [nthreads, nactelem]$, where $nthreads$ and $nactelem$ are the number of threads and the number of active elements, respectively. After decomposition, each bucket contains approximately the same number of elements ($nactelem/nbuckets$), and each thread owns approximately the same number of buckets ($nbuckets/nthreads$).

Optionally, the active elements can be pre-sorted in space using a Biased Randomized Insertion Order (BRIO) [49] and then ordered within each group along a Hilbert curve [36], to both improve the geometric locality and reduce the chance of a conflict between concurrent attempts of reconnection for adjacent elements [34]. Figure 4 depicts a decomposition of a tetrahedral mesh of a nozzle with and without element pre-sorting. Each thread maintains a unique list of buckets throughout the reconnection and processes the buckets in a consecutive manner. The elements within each bucket are repetitively reconnected until no new reconnections are possible.



(a) With element pre-sorting (b) Without element pre-sorting

Fig. 4 Decomposition of a tetrahedral mesh of a nozzle into 4 buckets with and without element pre-sorting. The centroids of the elements are sorted using a Biased Randomized Insertion Order (BRIO) [49] and a Hilbert curve [36]. Pre-sorting can potentially speedup the reconnection because it reduces the conflicts between concurrent attempts of reconnections for adjacent set of elements. The surface mesh obtained from CAVS Sims Center at MSU.

2. Thread safe operations

The presence of multiple threads operating on a shared mesh during local reconnection introduces the possibility of data races. First, topological transformations that have intersecting set of elements need to be synchronized. Another issue is that creation and deletion of elements needs to be synchronized in order to retain the data structure's integrity. For the former, atomic operations are employed. In particular, the vertices of the elements that take part in a topological

transformation remain locked throughout the transformation. If a thread encounters a locked vertex, it advances to the next transformation or the next element in the bucket(see line 9 in algorithm 1). Moreover, in case reconnection of a set of elements is not possible (i.e., the reconnection produces invalid elements or the objective function is not optimized), then the element is marked to avoid re-checking. For these operations atomic functions are utilized for synchronization since they perform faster than the conventional pthread try_locks [8].

To avoid communication when creating or deleting elements. New elements are inserted into the bucket of the thread that performs the reconnection. Deleted elements are removed from a bucket only if this bucket belongs to the thread that performs the reconnection; otherwise the element is marked as invalid, and the thread that owns the bucket removes it in a later step.

3. Load balancing

The static decomposition described in subsection1 is not always sufficient for balancing the work load throughout the parallel procedure because: (i) the number of geometrically valid transformations (i.e., those producing non-intersecting elements), and (ii) the number of optimum transformations (i.e. those optimizing the objective function), are not known a priori and may vary significantly among the different work units. To compensate these load differences a dynamic load balancing algorithm based on work-stealing [8, 50] is employed.

The load balancing algorithm migrates work units (*buckets*) between threads, when they run out of work units. For example, when thread T_i has finished processing its own list of buckets, it pushes its id, i, into a global list (*Waiting List*) that tracks down threads without work(see line 41 of algorithm 1). Then T_i yields until an other thread T_j transfers some work units to T_i's work pool. Every time a running thread T_j completes the processing of one bucket, it checks if the *Waiting List* contains any threads. If *Waiting List* is empty, then T_j continues with the next bucket. If *Waiting List* contains threads, then T_j transfers a fraction of its unprocessed buckets to those threads. The user controlled parameter $frbtransf \in (0,1]$ adjusts the fraction of work to be transferred (see lines 29 - 38 of algorithm 1). The default value is 30% but different values can be used for more or less aggressive behavior. Subsection C reports results on the performance of the parallel reconnection for varied fractions. Figure 5 illustrates an example with load balancing.

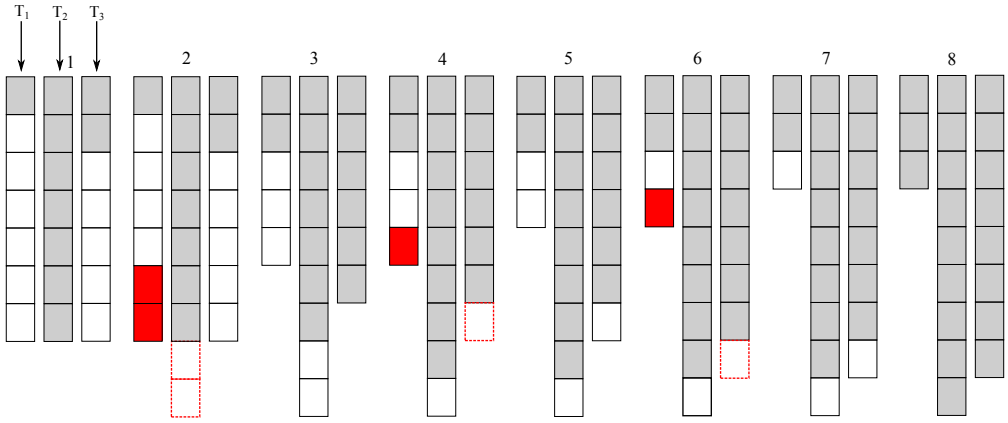


Fig. 5 Load balancing with three running threads in eight time steps. Gray indicates the work-units (buckets) currently processed in each time step. White indicates the work-units that have not been processed yet, in each time step. Solid red indicates the work-units to be transferred from a busy thread. Stroke red indicates the work-units to be received by a waiting thread. At step 1, each thread owns seven work-units and T_2 is on a waiting state. At step 2, T_2 is awakened by T_1 after T_1 transfers two work-units to T_2. At step 3, T_3 is on a waiting state. At step 4, one work-unit is transferred from T_1 to T_3. At step 5, all threads are busy. At step 6, one work-unit is transferred again from T_1 to T_3. At step 7, all threads are busy. At step 8, all threads have completed the processing of all work-units.

```
1  Function ParallelLocalReconnection(tid):
2      Start:
3      while BucketList[tid] ≠ ∅ do                                   /* iterate buckets of tid */
4          bucket = BucketList[tid] →get_next()
           // Bucket Refinement
5          while Flips are perfomed and iter_limit has not been exceeded do   /* termination condition for a
           bucket */
6
7              for tet ∈ bucket do
8
9                  success = tet→lock_vertices()
10                 if not sucess then continue
11                 else if tet is not active then tet→unlock_vertices() continue
12
13                 for neigh : tet→neighbors do                        /* Loop the four neighbors */
14                     if neigh == NULL then continue                          /* a boundary face */
15                     v = neigh→get_opposite_vertex(tet)
16                     success = v→lock() /* the other three vertices are already locked at line 9
                           */
17                     if not sucess then continue
18                     else
19                         if FindCandidate32Flip (tet,neigh,cavity) then Flip32 (cavity)
20                         else if FindCandidate23Flip (tet,neigh,cavity) then Flip23 (cavity)
21                         else if FindCandidate44Flip (tet,neigh,cavity) then Flip44 (cavity)
22                         v→unlock()
23                     end
24                 endfor
25                 tet→unlock_vertices()
26
27             endfor
28         endwhile
           // Load Balancing
29         if WaitingList ≠ ∅ and frbtransf > 0 then       /* there are threads waiting for work and load
           balancing is enabled */
30             w = ceil(unprocessed buckets · frbtransf)
31             while w > 0 do
32                 other_tid = WaitingList →pop()
33                 w_give = max(1,w/WaitingList→size)
34                 Push w_give of unprocessed buckets into BucketList[other_tid]
35                 Notify other_id
36                 w -= w_give
37             endwhile
38         end
39     endwhile
40     if WaitingList →size != #threads -1 then // If I am NOT the last thread to ask for work
41         WaitingList →push_back(tid)
42         Wait Until notified
43         if BucketList[tid] ≠ ∅ then goto Start // Some other thread gave work to tid
44         else return // Proceed to next Refinement Iteration
45     else
46         return // Proceed to next Refinement Iteration
47     end
48 End Function
```

Algorithm 1: Pseudocode of the fine-grained parallel reconnection scheme for topological transformations. `FindCandidate<n><m>Flip` finds an <n>-<m> flip between `tet` and `neigh` and the appropriate number of common neighbors that improves the objective function. `Flip23,Flip32,Flip44` implement the transformations of Figure 3.

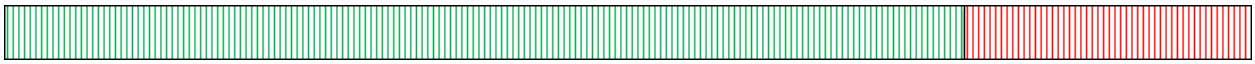
4. Method Overview

A high level description of the algorithm is given in algorithm 1. This function is executed by each thread. Entering this function the mesh tetrahedra have been divided into buckets (see subsection 1) and the buckets have been divided among the threads. Each thread iterates the tetrahedra in each bucket of its own bucketList (lines 3 -7). Prior to any operation the tetrahedron is locked by locking the vertices utilizing atomic operations (line 9). In case, another thread has locked any of the vertices the thread skips this tetrahedron and proceeds to the next one. After acquiring the lock of the `tet`'s vertices the algorithm will attempt to lock one of the 4 neighboring tetrahedra. Since any tetrahedron shares 3 vertices with a neighbor, a lock of the opposite vertex is enough (line 16). Having the two tetrahedra locked allows now to check for candidate flips. `FindCandidate32Flip` will try to find a configuration similar to figure 3. That is, a third tetrahedron that shares and edge with `tet` and `neigh`, note that the edge should be shared exactly by three tetrahedra. Upon return, `cavity` contains the three tetrahedra that will take part in the flip and `Flip32` implements the topological transformation. `FindCandidate23Flip` and `FindCandidate44Flip` operate in a similar fashion.

After refining the tetrahedra in a bucket. The thread will check the $WaitingList$ for idle threads and it will give to the idle threads at most $frbtrasnf\%$ of its buckets (lines 29 - 38). In case the thread runs out of work it will push its id in the $WaitingList$ and wait until either work has been given or the refinement iteration has been terminated (lines 40 - 44).

B. Improving Data Locality by Grouping Element Link-list

CDT3D stores the elements in a double link-list data structure. The link-list contains both active and inactive elements. Some components (i.e., point creation, element deactivation, over-decomposition, and parallel reconnection) do not require a traversal of the inactive elements, therefore a separation between active and inactive elements can potentially improve the performance (Figure 6).



(a) Element link-list with grouped elements.

(b) Element link-list with shuffled elements.

Fig. 6 The two types of link-lists in CDT3D. Green represents the active elements. Red represents the inactive elements.

The grouped link-list reduces threefold the refinement time compared to a shuffled link-list. Table 1 illustrates the improvements in more detail. Figure 7 depicts the quality histogram between the two types of link-list. The two histograms are very close. There is however a significant difference in number of elements (88.88M in Shuffled case, 78.97M in Grouped). This could be attributed to the different order of refinement. It should be noted that, grouping the link-list during the post-processing quality improvement step didn't offer substantial speedup because the time required for smoothing dominates the quality improvement step.

Table 1 Performance Improvement due to grouping active elements for mesh refienement operations (times in min).

Mesh Refinement Operations	Shuffled	Grouped	Improvement
Point Creation	11.10	5.17	2.15x
Element Deactivation	14.66	3.90	3.76x
Mesh Decomposition	10.93	1.60	6.82x
Parallel Reconnection	25.11	9.21	2.73x
Mesh Refinement (total)	62.14	20.19	3.07x

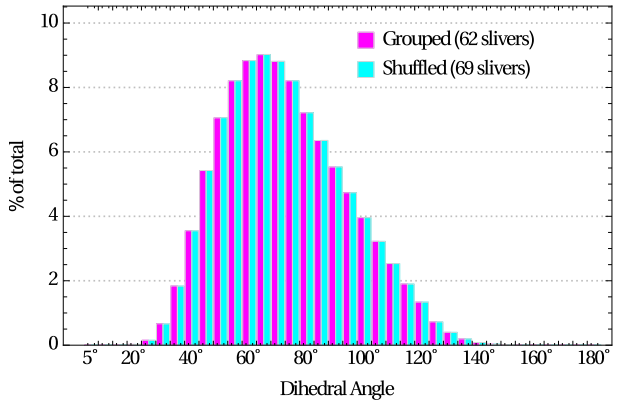


Fig. 7 Element angle distribution (in 5-deg increments) of Lv2b grids generated using a grouped and a shuffled element link-list.

VI. Evaluation Results

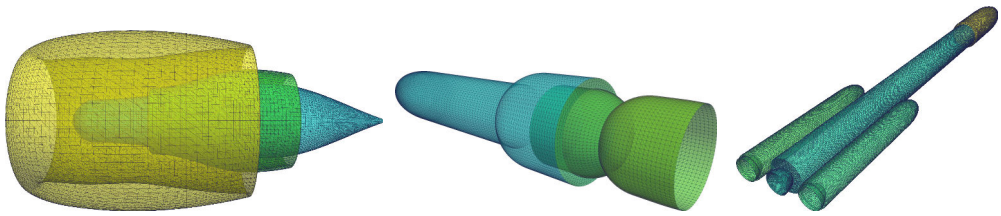
The evaluation on the quality and scalability was performed on geometries pertinent to aerospace applications. In particular, three realistic aerospace configurations from the CAVS Sims center at MSU and a surface triangulation of a DLR-F6 Airbus type aircraft were used.

A. Comparison with AFLR

CDT3D is compared with state-of-the-art unstructured grid technology AFLR v16.9 [3, 4]. AFLR is directly incorporated in several systems. CDT3D and AFLR have a handful of options for quality mesh generation and optimization. Only a few basic options of these codes are tested, hence these comparisons are far from comprehensive. The comparison is performed on three realistic aerospace configurations:

1) An aircraft nacelle with engine inside a section of wind tunnel (Figure 8(a))
2) A rocket with engine, nozzle, and transparent internal data surfaces inside a flow field (Figure 8(b))
3) A launch vehicle with solid boosters inside a flow field (Lv2b) (Figure 8(c))

The surface meshes of the geometries obtained from CAVS Sims Center at MSU in .surf format. The experiments performed on a DELL workstation with Linux Ubuntu 12.10, 12 cores Intel®Xeon®CPU X5690@3.47 GHz, and 96 GB RAM. Table 2 presents the results. This study uses the dihedral angle as a metric to evaluate the element quality.



(a) nacelle with engine (b) rocket with engine (c) launch vehicle with solid boosters

Fig. 8 Surface Meshes used in this evaluation.

Both methods generate meshes of good quality elements. CDT3D exhibits a comparable quality compared to AFLR in both the sequential and the parallel runs. CDT3D eliminates all elements whose dihedral angles are smaller than 6.60° or larger than 159.68°. The corresponding values for AFLR are 5.58° and 164.86°, respectively (Table 2).

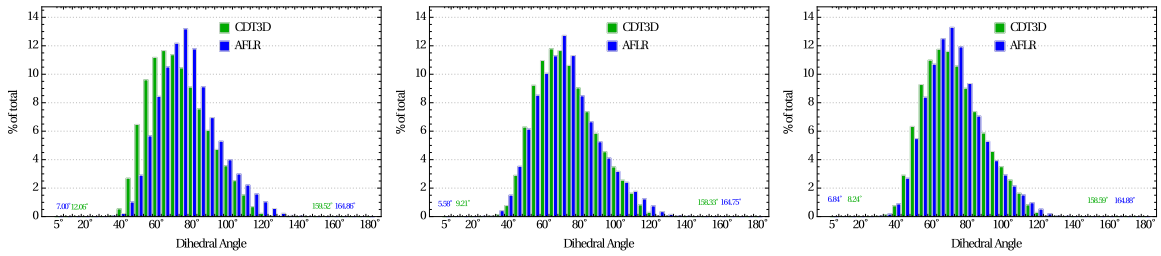
Figure 9 depict the element angle distribution. At the completion of the refinement a small percentage of sliver elements may survive ($< 0.003\%$). CDT3D incorporates effective quality improvement techniques to eliminate the sliver elements.

CDT3D completes the end-to-end mesh generation (including initial mesh construction, refinement, and improvement) up to 1.90 times faster than AFLR, when 12 hardware cores are utilized (Table 2). Most notably, CDT3D refines

Table 2 Evaluation results on unstructured mesh generation. CDT3D is compared with state-of-the-art technology AFLR v16.9.19 [3]. CDT3D's runs are performed with 1 and 12 hardware cores. The sliver elements have a dihedral angle smaller than 2° or larger than 178°. Initial mesh includes Delaunay tetrahedralization and Boundary Recovery. The I/O time is not included.

Case	Software	#Cores	%Slivers (w/o improv.) ($\times 10^{-3}$)	#Tets (w/ improv.) (M)	Min/Max Angle (w/ improv.) (deg)	Time			
						Initial Mesh (sec)	Refinement (min)	Quality Improvement (min)	Total (min)
Nacelle	CDT3D	1	3.74	43.65	13.57°/153.44°	1.36	20.01	14.30	34.33
		12	3.70	42.85	12.06°/159.52°	1.36	5.02	18.59	23.64
	AFLR	1	2.97	43.16	7.00°/164.86°	5.63	22.59	6.40	29.09
Rocket	CDT3D	1	2.96	118.41	9.39°/159.30°	1.58	52.85	64.56	117.44
		12	2.95	119.06	9.21°/158.33°	1.58	14.51	68.23	82.76
	AFLR	1	3.05	123.13	5.58°/164.75°	6.76	131.89	25.41	157.42
Lv2b	CDT3D	1	5.09	98.21	6.60°/159.68°	5.45	41.57	94.63	136.29
		12	4.69	113.99	8.24°/158.59°	5.45	12.92	62.36	75.37
	AFLR	1	3.49	104.10	6.84°/164.88°	16.97	98.24	18.51	117.03

the mesh up to 2.5 and 10 times faster compared to AFLR, when 1 and 12 hardware cores are utilized, respectively. On the other hand, AFLR exhibits very good performance at the quality improvement step. Both methods complete the construction of the initial mesh (i.e., Delaunay tetrahedralization and boundary recovery) at a negligible cost (less than 1% of the total generation time).



(a) Nacelle (b) Rocket (c) Lv2b

Fig. 9 Element angle distribution (in 5-deg increments) after improvement of Nacelle, Rocket and Lv2b meshes. The dihedral angle extrema are reported for each method.

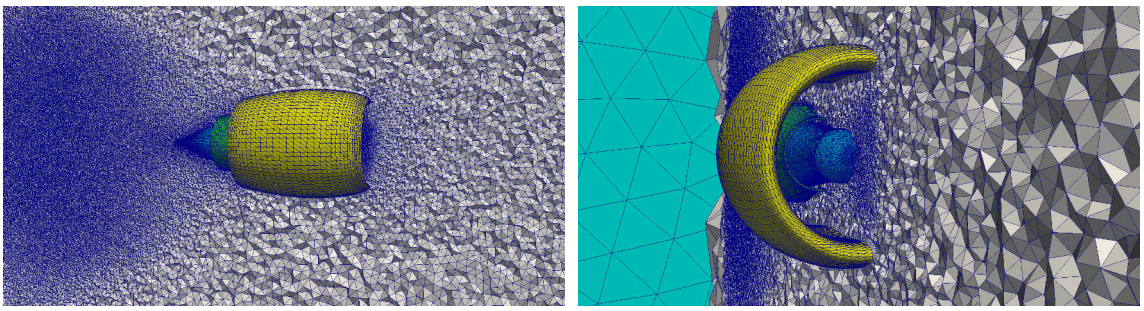


Fig. 10 Detail views of tetrahedral field cuts of aircraft nacelle generated with CDT3D.

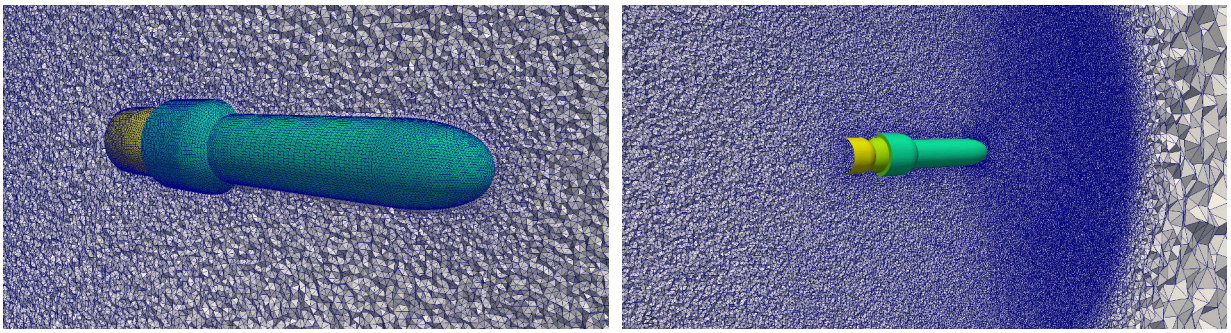


Fig. 11 Tetrahedral field cuts of the rocket mesh generated with CDT3D.

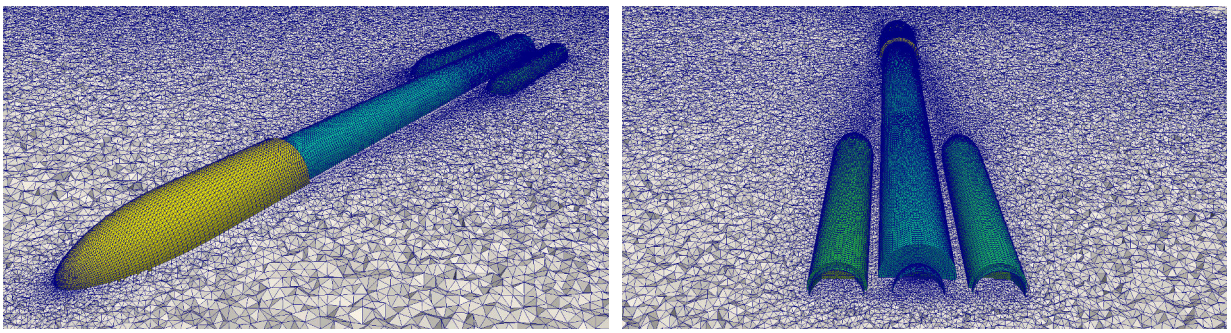
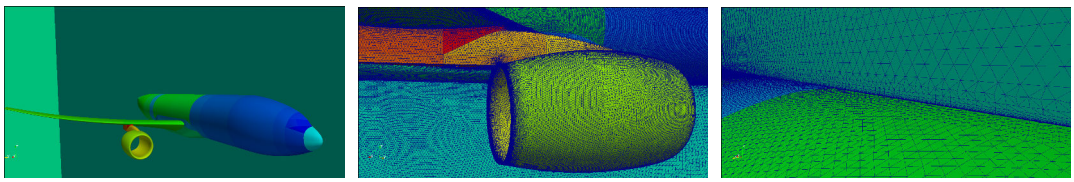


Fig. 12 Tetrahedral field cuts of the Lv2b mesh generated with CDT3D.

B. Results using NASA's Common Research Model

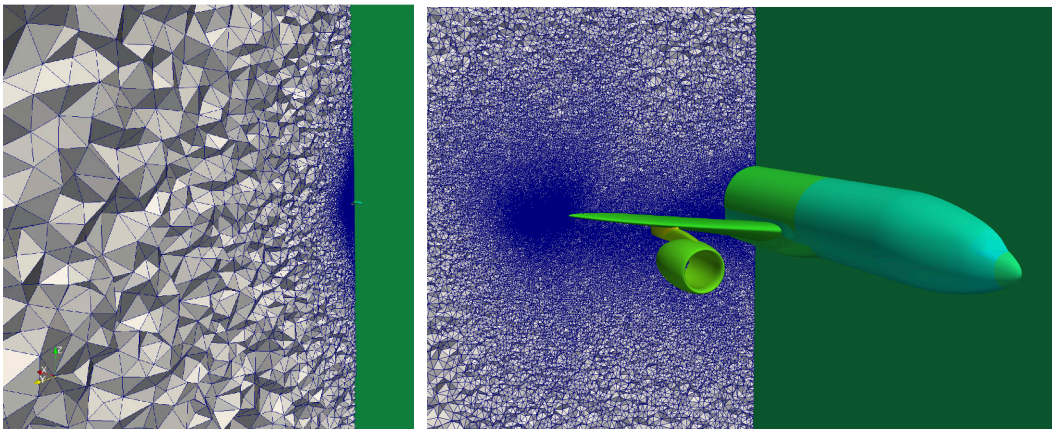
This study evaluates the scalability of CDT3D using NASA's Common Research Model[*]. The model is a DLR-F6 Airbus type aircraft. A tetrahedral mesh of this model is generated with VGRID[†] [51, 52] for the purposes of the 6th AIAA CFD Drag Prediction Workshop[‡]. The surface mesh is then extracted using UGC[§] and it is passed to CDT3D for volume mesh generation. Figure 13 depicts the input surface mesh. In this case, boundary recovery is challenging, because the surface mesh contains high aspect-ratio anisotropic triangles at the junction of the symmetry wall and the aircraft's body (Figure 13(c)). In CFD, the boundary is usually recovered from an isotropic triangulation rather than an anisotropic triangulation, because boundary layers are first generated. Therefore the isotropic generator starts with a new isotropic boundary surface. Nevertheless it is good to demonstrate robustness. A tetrahedral mesh is attempted to generate with AFLR, but the execution failed due to a topological error in boundary recovery.



(a) Aircraft with symmetry plane (b) Wing-nacelle-pylon system (c) Anisotropic boundary layers

Fig. 13 Surface mesh of a DLR-F6 Airbus type aircraft with anisotropic boundary layers on a symmetry plane; #points: 1006144; #triangles: 2012288.

[*]https://commonresearchmodel.larc.nasa.gov/2012/01/19/hello-world-2
[†]https://geolab.larc.nasa.gov/GridTool/Training/VGRID/
[‡]https://aiaa-dpw.larc.nasa.gov
[§]http://www.simcenter.msstate.edu/



(a) Overall view (b) Detail view

Fig. 14 Cut of the tetrahedral mesh of the flow field of DLR-F6 Airbus aircraft, generated with CDT3D. A smaller mesh (≈200 M tetrahedra) is depicted due to limitations in visualization.

Table 3 Performance results on parallel refinement of mesh of a flow domain around a DLR-F6 Airbus aircraft. The (included) sorting time and efficiency are reported in parenthesis. The sliver elements have a dihedral angle smaller than 2° or larger than 178°. #Iter is the number of mesh generation passes.

	nthreads	%Slivers		#Iter	Time			Speedup *(% Efficiency)*		
		#Tets (Bi)	(w/o improv.) ($\times 10^{-2}$)		Recon./Iter (min)	Reconnection (hours)	Refinement (hours)	Recon./Iter	Reconnection	Refinement
	1	1.414	1.438	61	51.69	52.56	58.98	1	1	1
w/o sorting	12	1.413	1.472	73	4.81	5.85	13.10	10.74 *(89.58)*	8.98 *(74.83)*	4.50
	24	1.455	1.563	83	2.51	3.48	11.81	20.59 *(85.67)*	15.11 *(62.96)*	5.00
	36	1.438	1.487	79	1.98	2.62	10.59	26.10 *(72.17)*	20.06 *(55.72)*	5.57
	48	1.451	1.556	118	1.84	3.62	14.36	28.09 *(58.52)*	14.52 *(30.25)*	4.10
w/ sorting	12	1.414	1.563	89	3.99	5.92	17.38 (3.62)	12.95 *(107.92)*	8.88 *(74.00)*	3.39
	24	1.439	1.499	75	1.98	2.48	12.74 (3.16)	26.10 *(108.63)*	21.21 *(88.38)*	4.62
	36	1.458	1.518	93	1.67	2.60	14.87 (4.00)	30.95 *(85.75)*	20.25 *(56.25)*	3.96
	48	1.448	1.625	122	1.39	2.84	18.77 (5.08)	37.18 *(77.06)*	18.49 *(38.52)*	3.14

This evaluation conducts a total of nine runs; a sequential run and two sets of parallel runs for varied number of threads (12-48). The first set of parallel runs is performed without element sorting (default option in CDT3D). The second set is performed with element sorting. Element sorting has been suggested in the literature [34] because it reduces the overlaps between regions involved in concurrent transformations, thus providing a good initial work load distribution for parallel reconnection. Quality improvement is disabled because the anisotropic surface creates a high number of slivers on the boundary which is time consuming to remove. The elements are sorted based on the coordinates of their centroids, according to a Hilbert curve [36, 37]. CDT3D sorts the elements in each refinement pass before over-decomposition. Figure 14 depicts a cut of the isotropic tetrahedral mesh. Table 3 presents the results, the additional cost for element sorting is in parenthesis. This cost is included in the refinement module. Also refinement includes overheads due to other sequential components. Speedup and efficiency are calculated using as a base case CDT3D with 1 thread. The experiments performed on a DELL workstation with Linux Red Hat Enterprise, 24 hardware cores (2x Intel®Xeon®CPU E5-2697v2@2.70 GHz), Hyper Threading support and 757 GB RAM.

When using hardware threads the efficiency of the average reconnection iteration is close to 90%. Sorting the elements offers more than 10% improvement allowing thus for superlinear speedup. Hyper-Threading performs also well with up to 72% efficiency without sorting and 85% with sorting. These results verify the choice of implementing the topological transformations using the speculative approach. The high density of communication of these operations matches well with the lower part of the telescopic approach. However, due to the non-determinism of the parallel execution the required number of iterations varies significantly. This could be related to the fact that in case of rollbacks (see lines 10 and 17 of algorithm 1) elements are not checked again until the next iteration, requiring thus more iterations to refine all active elements. Future work will investigate efficient methods to revisit these conflicts during the same

iterations in combination with contention managers that proved quite effective in previous work [8].

Increase in number of iterations causes an increase the total reconnection time decreasing thus the efficiency. Yet still the code exhibits up to 88% efficiency on 24 threads when sorting is enabled. The last column includes the overall speedup for reference. Since only the reconnection part is parallel Amdahl's law constrains the efficiency of the overall algorithm. When 1 thread is utilized, the non-parallelized components together account for $(58.98 - 52.56)/58.98 = 10.89\%$ of the total refinement time. When 48 threads are utilized (without sorting), they account for 74.80%. When 48 threads are utilized and sorting is employed, the overheads due to sequential components increase to 84.94% (Table 3). Still, CDT3D enhances user productivity offering a significant improvement over the more than 2 days required for the single threaded execution.

The use of sorting during the mesh decomposition decreases the reconnection time due to the reduced number of rollbacks. However, due to the big size of the mesh its cost in non-negligible. Furthermore, since sorting enforces different order of refinement and thus a different mesh throughout the iterations, it may also affect indirectly the number of iterations. Evaluating these dependencies as well as optimizing and parallelizing the sorting procedure is part of the future work.

C. Results with Speculative Execution

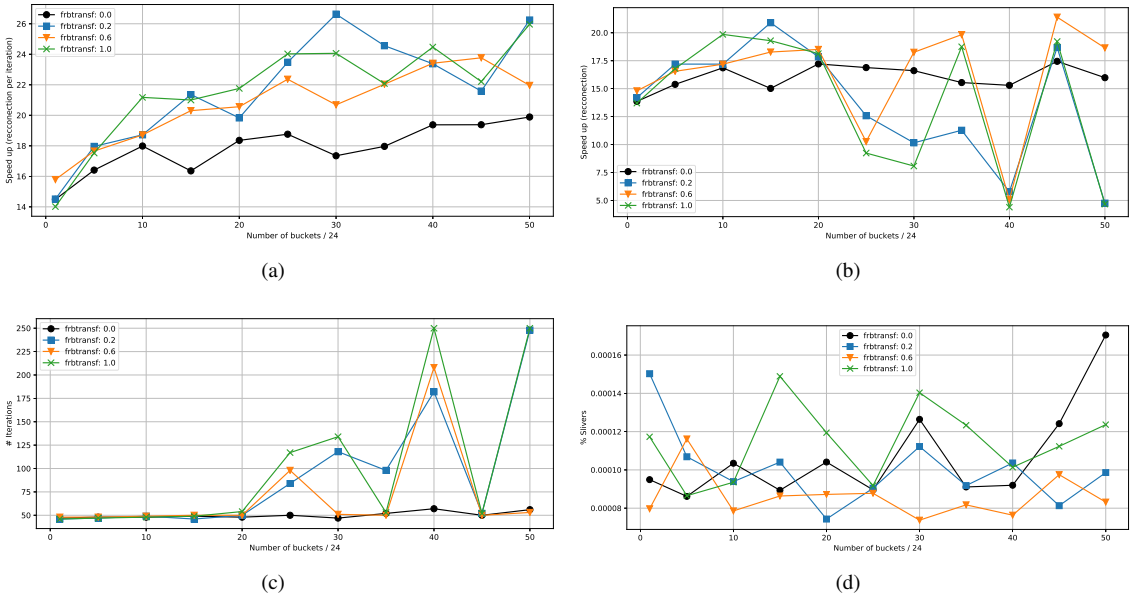


Fig. 15 Performance results on parallel reconnection and refinement of Lv2b grid, for varied granularities for over-decomposition (nbuckets/24), and fractions for bucket-migration (frbtransf) using 24 hardware threads.

A set of runs are conducted to assess the performance of the CDT3D for varied granularities for over-decomposition, and fractions for bucket-migration. The experiments performed on a DELL workstation with Linux Red Hat Enterprise, 24 hardware cores (2x Intel®Xeon®CPU E5-2697v2@2.70 GHz) and 757 GB RAM. Each run is performed with 24 hardware cores. The granularity is adjusted with parameter $nbuckets$. The higher the number of buckets, the finer the decomposition. The fraction for bucket-migration is adjusted with parameter $frbtransf$. The higher the fraction, the higher the number of buckets to be transferred between threads. The rest of the parameters are fixed among the runs. The experiments are conducted on the Lv2b geometry. Figure 15 depicts the results.

Enabling the load balancer ($frbtransf > 0$) increases the average speedup per iteration for all bucket counts. The best speedup obtained is 26.63 which corresponds to a superlinear efficiency of 110% (Figure 15(a)). However, the best speedup for the reconnection module is 21.39 ($frbtransf = 0.6$) due to the variation on the number of iterations (Figures 15(b) and 15(c)). The correlation to the number of iterations can be clearly seen by the fact that for ($nbuckets/24 > 20$) Figures 15(b) and 15(c) are almost symmetric to each other. From Figure 15(b) one can see that with lower number of initial buckets < 20, load balancing improves the performance of reconnection, while as

the number of initial buckets grows the results are inconclusive. On the other hand, the number of iterations remains approximately the same when no data-migration is performed, regardless of the level of granularity. Figure 15(d) depicts the number of tetrahedra for each run. The final size of the mesh ranges between 75 to 89 million elements. It should be noted that all the generated meshes are of a high quality. In the worst case scenario, only 0.00017% of the elements have a dihedral angle smaller than 2° or larger than 178°. Overall, the speculative results show that CDT3D achieves a good trade-off between the percentage of slivers, the number of mesh generation iterations and the reconnection time, when $nbuckets/24 : 15 - 20$ and $frbtransf : 0.2 - 0.6$.

The above results suggest that over-decomposition should be used with care. A finer grain size and thus a higher number buckets caused a noticeable performance deterioration. Investigating the cause is part of the future work. As with the results of the previous section, this could be related to the current absence of treatment for rollbacks during each iteration.

VII. Conclusion

A new speculative local reconnection method for unstructured mesh generation of high quality isotropic elements has been presented. To the best of our knowledge, this is the first non-Delaunay fine-grain tightly-coupled parallel method that optimizes the mesh connectivity in parallel throughout the generation procedure, including a post-improvement step. Similar tightly-coupled parallel Delaunay-based methods using speculative execution model appeared in [18, 53]. The connectivity optimization is based on a speculative approach that has been proven to perform well on hardware-shared memory (i.e., single chip). The results are very encouraging and suggest that integration with next layers as in [54, 55] can scale linearly to both Distributed Shared Memory and Distributed Memory platforms. The mesh generator is evaluated on a variety of aerospace configurations. The results indicate that the high quality and performance attributes of this method are comparable to existing state-of-the-art technology. In the future, performance is expected to improve by completing the fine-grained parallelization of other components (i.e., point creation as well as vertex smoothing) which is currently under active development. Moreover, a more throughout study on the dependence of the meshing time and the final mesh quality to the input parameters will be conducted. Given the importance of mesh adaptation in aerospace applications, generating anisotropic meshes adapted to a metric field will be investigated. The proposed framework will be part of an Extreme-Scale Anisotropic Mesh Generation Environment to meet industries expectations and NASA's CFD Vision for 2030 [2, 56].

VIII. Acknowledgments

This work is in part funded by NSF grant no. CCF-1439079, Richard T. Cheng Endowment and the Modeling and Simulation fellowship of Old Dominion University.

References

[1] Chrisochoides, N., Chernikov, A., Fedorov, A., Kot, A., Linardakis, L., and Foteinos, P., "Towards Exascale Parallel Delaunay Mesh Generation," *18th International Meshing Roundtable*, Springer Berlin Heidelberg, 2009, pp. 319–336. doi:10.1007/978-3-642-04319-2_19.

[2] Chrisochoides, N. P., "Telescopic Approach for Extreme-scale Parallel Mesh Generation for CFD Applications," *46th AIAA Fluid Dynamics Conference*, 2016, p. 3181.

[3] Marcum, D. L., and Weatherill, N. P., "Unstructured grid generation using iterative point insertion and local reconnection," *AIAA Journal*, Vol. 33, No. 9, 1995, pp. 1619–1625.

[4] Marcum, D., "Generation of unstructured grids for viscous flow applications," *33rd Aerospace Sciences Meeting and Exhibit*, 1995, p. 212.

[5] Delaunay, B., "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, Vol. 7, No. 793-800, 1934, pp. 1–2.

[6] Barth, T., "Numerical aspects of computing high Reynolds number flows on unstructured meshes," *29th Aerospace Sciences Meeting*, 1991, p. 721.

[7] George, P.-L., and Borouchaki, H., "Back to Edge Flips in 3 Dimensions," *12th International Meshing Roundtable*, 2003, pp. 393–402.

[8] Foteinos, P. A., and Chrisochoides, N. P., "High quality real-time image-to-mesh conversion for finite element simulations," *Journal of Parallel and Distributed Computing*, Vol. 74, No. 2, 2014, pp. 2123–2140.

[9] Thompson, J. F., Soni, B. K., and Weatherill, N. P., *Handbook of grid generation*, CRC press, 1998.

[10] George, P. L., Hecht, F., and Saltel, É., "Automatic mesh generator with specified boundary," *Computer methods in applied mechanics and engineering*, Vol. 92, No. 3, 1991, pp. 269–288.

[11] Baker, T. J., "Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation," *Engineering with Computers*, Vol. 5, No. 3-4, 1989, pp. 161–175.

[12] Shewchuk, J. R., "Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery," *IMR*, Citeseer, 2002, pp. 193–204.

[13] Murphy, M., Mount, D. M., and Gable, C. W., "A point-placement strategy for conforming Delaunay tetrahedralization," *International Journal of Computational Geometry & Applications*, Vol. 11, No. 06, 2001, pp. 669–682.

[14] Chrisochoides, N. P., and Sukup, F., *Task parallel implementation of the Bowyer-Watson algorithm*, Vol. 235, Citeseer, 1996.

[15] Okusanya, T., and Peraire, J., "3-D Parallel unstructured mesh generation," *Proc. Joint ASME/ASCE/SES Summer Meeting*, 1997.

[16] Bowyer, A., "Computing Dirichlet tessellations," *The Computer Journal*, Vol. 24, No. 2, 1981, pp. 162–166. doi:doi:10.1093/comjnl/24.2.162.

[17] Watson, D., "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes," *The Computer Journal*, Vol. 24, No. 2, 1981, pp. 167–172. doi:doi:10.1093/comjnl/24.2.167.

[18] Nave, D., Chrisochoides, N., and Chew, L. P., "Guaranteed: quality parallel delaunay refinement for restricted polyhedral domains," *Proceedings of the eighteenth annual symposium on Computational geometry*, ACM, 2002, pp. 135–144.

[19] Chernikov, A. N., and Chrisochoides, N. P., "Three-dimensional Delaunay refinement for multi-core processors," *Proceedings of the 22nd annual international conference on Supercomputing*, ACM, 2008, pp. 214–224.

[20] Barker, K., Chernikov, A., Chrisochoides, N., and Pingali, K., "A load balancing framework for adaptive and asynchronous applications," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 2, 2004, pp. 183–192.

[21] Lawson, C. L., "Properties of n-dimensional triangulations," *Computer Aided Geometric Design*, Vol. 3, No. 4, 1986, pp. 231–246.

[22] Joe, B., "Construction of three-dimensional improved-quality triangulations using local transformations," *SIAM Journal on Scientific Computing*, Vol. 16, No. 6, 1995, pp. 1292–1307.

[23] Shewchuk, J. R., "Two discrete optimization algorithms for the topological improvement of tetrahedral meshes," *Unpublished manuscript*, Vol. 65, 2002.

[24] Löhner, R., and Parikh, P., "Generation of three-dimensional unstructured grids by the advancing-front method," *International Journal for Numerical Methods in Fluids*, Vol. 8, No. 10, 1988, pp. 1135–1149.

[25] Lo, S. H., "Volume discretization into tetrahedra-II. 3D triangulation by advancing front approach," *Computers & Structures*, Vol. 39, No. 5, 1991, pp. 501–511.

[26] Löhner, R., "A parallel advancing front grid generation scheme," *International Journal for Numerical Methods in Engineering*, Vol. 51, No. 6, 2001, pp. 663–678.

[27] Ito, Y., Shih, A. M., Erukala, A. K., Soni, B. K., Chernikov, A., Chrisochoides, N. P., and Nakahashi, K., "Parallel unstructured mesh generation by an advancing front method," *Mathematics and Computers in Simulation*, Vol. 75, No. 5, 2007, pp. 200–209.

[28] Karypis, G., and Kumar, V., "A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, 1998.

[29] Chrisochoides, N., and Nave, D., "Parallel Delaunay mesh generation kernel," *International Journal for Numerical Methods in Engineering*, Vol. 58, No. 2, 2003, pp. 161–176.

[30] Remacle, J.-F., Bertrand, V., and Geuzaine, C., "A two-level multithreaded Delaunay kernel," *Procedia Engineering*, Vol. 124, 2015, pp. 6–17.

[31] Said, R., Weatherill, N., Morgan, K., and Verhoeven, N., "Distributed parallel Delaunay mesh generation," *Computer methods in applied mechanics and engineering*, Vol. 177, No. 1-2, 1999, pp. 109–125.

[32] Löhner, R., "A 2nd generation parallel advancing front grid generator," *Proceedings of the 21st international meshing roundtable*, Springer, 2013, pp. 457–474.

[33] Löhner, R., "Recent advances in parallel advancing front grid generation," *Archives of Computational Methods in Engineering*, Vol. 21, No. 2, 2014, pp. 127–140.

[34] Shang, M., Zhu, C., Chen, J., Xiao, Z., and Zheng, Y., "A Parallel Local Reconnection Approach for Tetrahedral Mesh Improvement," *Procedia Engineering*, Vol. 163, 2016, pp. 289–301.

[35] Freitag, L. A., and Ollivier-Gooch, C., "Tetrahedral mesh improvement using swapping and smoothing," *International Journal for Numerical Methods in Engineering*, Vol. 40, No. 21, 1997, pp. 3979–4002.

[36] Boissonnat, J.-D., Devillers, O., and Hornus, S., "Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension," *Proceedings of the twenty-fifth annual symposium on Computational geometry*, ACM, 2009, pp. 208–216.

[37] Si, H., "TetGen, a Delaunay-based quality tetrahedral mesh generator," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 41, No. 2, 2015, p. 11.

[38] Zangeneh, R., "Thread-parallel mesh generation and improvement using face-edge swapping and vertex insertion," Master's thesis, University of British Columbia, 2014.

[39] Benítez, D., Rodríguez, E., Escobar, J. M., and Montenegro, R., "Performance evaluation of a parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes," *Proceedings of the 22nd International Meshing Roundtable*, Springer, 2014, pp. 579–598.

[40] Kim, J., Panitanarak, T., and Shontz, S. M., "A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling," *International Journal for Numerical Methods in Engineering*, Vol. 94, No. 1, 2013, pp. 20–42.

[41] Sastry, S. P., Shontz, S. M., and Vavasis, S. A., "A log-barrier method for mesh quality improvement and untangling," *Engineering with Computers*, Vol. 30, No. 3, 2014, pp. 315–329.

[42] Sastry, S. P., and Shontz, S. M., "A parallel log-barrier method for mesh quality improvement and untangling," *Engineering with Computers*, Vol. 30, No. 4, 2014, pp. 503–515.

[43] Freitag, L., Jones, M., and Plassmann, P., "A parallel algorithm for mesh smoothing," *SIAM Journal on Scientific Computing*, Vol. 20, No. 6, 1999, pp. 2023–2040.

[44] Zagaris, G., Pirzadeh, S., and Chrisochoides, N., "A Framework for Parallel Unstructured Grid Generation for Practical Aerodynamic Simulations," *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2009.

[45] Gorman, G. J., Southern, J., Farrell, P. E., Piggott, M., Rokos, G., and Kelly, P. H., "Hybrid OpenMP/MPI anisotropic mesh smoothing," *Procedia Computer Science*, Vol. 9, 2012, pp. 1513–1522.

[46] de L'isle, E. B., and George, P. L., "Optimization of tetrahedral meshes," *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, Springer, 1995, pp. 97–127.

[47] Klingner, B. M., and Shewchuk, J. R., "Aggressive tetrahedral mesh improvement," *Proceedings of the 16th international meshing roundtable*, Springer, 2008, pp. 3–23.

[48] Edelsbrunner, H., and Shah, N. R., "Incremental topological flipping works for regular triangulations," *Algorithmica*, Vol. 15, No. 3, 1996, pp. 223–241.

[49] Amenta, N., Choi, S., and Rote, G., "Incremental constructions con BRIO," *Proceedings of the nineteenth annual symposium on Computational geometry*, ACM, 2003, pp. 211–219.

[50] Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H., and Zhou, Y., *Cilk: An efficient multithreaded runtime system*, Vol. 30, ACM, 1995.

[51] Pirzadeh, S., "Unstructured viscous grid generation by advancing-front method," Tech. rep., NASA, 1993.

[52] Pirzadeh, S., "Unstructured viscous grid generation by the advancing-layers method," *AIAA Journal*, Vol. 32, No. 8, 1994, pp. 1735–1737.

[53] Foteinos, P., and Chrisochoides, N., "High Quality Real-time Image-to-mesh Conversion for Finite Element Simulations," *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ACM, New York, NY, USA, 2013, pp. 233–242. doi:10.1145/2464996.2465439.

[54] Feng, D., Tsolakis, C., Chernikov, A. N., and Chrisochoides, N. P., "Scalable 3D Hybrid Parallel Delaunay Image-to-mesh Conversion Algorithm for Distributed Shared Memory Architectures," *Computer-Aided Design*, Vol. 85, 2017, pp. 10–19. doi:10.1016/j.cad.2016.07.010.

[55] Feng, D., Chernikov, A. N., and Chrisochoides, N. P., "A Hybrid Parallel Delaunay Image-to-mesh Conversion Algorithm Scalable on Distributed-memory Clusters," *Computer-Aided Design*, 2018, p. to appear.

[56] Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD vision 2030 study: a path to revolutionary computational aerosciences," 2014.