A Unified Framework for Parallel Anisotropic Mesh Adaptation

Christos Tsolakis
Old Dominion University, ctsolakic@gmail.com

A UNIFIED FRAMEWORK FOR PARALLEL ANISOTROPIC MESH ADAPTATION

by

Christos Tsolakis
B.S. Mathematics, July 2014, Aristotle University of Thessaloniki, Greece

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2021

ABSTRACT

A UNIFIED FRAMEWORK FOR PARALLEL ANISOTROPIC MESH ADAPTATION

Christos Tsolakis
Old Dominion University, 2021
Director: Dr. Nikos Chrisochoides

Finite-element methods are a critical component of the design and analysis procedures of many (bio-)engineering applications. Mesh adaptation is one of the most crucial components since it discretizes the physics of the application at a relatively low cost to the solver. Highly scalable parallel mesh adaptation methods for High-Performance Computing (HPC) are essential to meet the ever-growing demand for higher fidelity simulations. Moreover, the continuous growth of the complexity of the HPC systems requires a systematic approach to exploit their full potential. Anisotropic mesh adaptation captures features of the solution at multiple scales while, minimizing the required number of elements. However, it also introduces new challenges on top of mesh generation. Also, the increased complexity of the targeted cases requires departing from traditional surface-constrained approaches to utilizing CAD (Computer-Aided Design) kernels. Alongside the functionality requirements, is the need of taking advantage of the ubiquitous multi-core machines. More importantly, the parallel implementation needs to handle the ever-increasing complexity of the mesh adaptation code.

In this work, we develop a parallel mesh adaptation method that utilizes a metric-based approach for generating anisotropic meshes. Moreover, we enhance our method by interfacing with a CAD kernel, thus enabling its use on complex geometries. We evaluate our method both with fixed-resolution benchmarks and within a simulation pipeline, where the resolution of the discretization increases incrementally. With the *Telescopic Approach* for scalable mesh generation as a guide, we propose a parallel method at the node (multi-core) for mesh adaptation that is expected to scale up efficiently to the upcoming exascale machines. To facilitate an effective implementation, we introduce an abstract layer between the application and the runtime system that enables the use of task-based parallelism for concurrent mesh operations. Our evaluation indicates results comparable to state-of-the-art methods for fixed-resolution meshes both in terms of performance and quality. The integration with an adaptive pipeline offers promising results for the capability of the proposed

method to function as part of an adaptive simulation. Moreover, our abstract tasking layer allows the separation of different aspects of the implementation without any impact on the functionality of the method.

"It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment."
— Carl Friedrich Gauss (1777 - 1855)

ACKNOWLEDGMENTS

They say “it takes a village to raise a child” and I believe that something similar is true for a Ph.D. First and foremost, I would like to thank my advisor Nikos Chrisochoides for the countless hours of discussions and for giving me the chance to be part of a wide spectrum of projects that allowed to build a more general perspective about parallel mesh generation. Also, I would like to thank my committee members: Mike Park, Andrey Chernikov, Desh Ranjan, and Charles E. Hyde for their time reviewing my thesis and their advice and help throughout my Ph.D. studies.

Generating and collecting the experimental data was accomplished thanks to the advice and lessons I took from multiple sources. In particular, I would like to thank Hiroaki Nishikawa for the CFD II class that taught me enough about CFD solvers to be able to use SU2 and produce the results of the evaluation section. Moreover, his unique style of presentations and the enthusiasm he brings into them, will always be a template for my presentations. I would like also to thank the UGAWG (Unstructured Grid Adaptation Working Group) and especially Mike Park and Todd Michal for inviting me to their monthly meetings, which gave me a unique opportunity to have a peek into their mesh adaptation projects and how they apply to both research and industrial level applications. I also thank both of them along with Adrien Loseille for their patience and help towards writing our joined papers. The UGAWG's GitHub repository was a valuable resource for this thesis. Almost all evaluation cases have been created based on material publicly available on the repository or described in detail in their papers. I would like to thank Beckett Zhou for spending many hours explaining to me CFD simulations at a high level and giving me insights about isotropic mesh adaptation in the context of our joined paper.

Special thanks to Gagik Gavalian, Christian Weiss and Charles E. Hyde for introducing us to some of the notions of Nuclear Femtography and showing us alternative uses of meshes in the context of our joined projects.

In 2017, I had the luck to be selected for the Argonne Training Program on Extreme-Scale Computing. Marta García Martínez and the rest of the team at ANL offered us a unique experience that fueled me with ideas for the rest of my Ph.D. journey and beyond.

Also, I would like to thank a few of my professors that I had classes with, and in particular, Michele Weigle for the Information Visualization class that improved significantly the graphs presented in this thesis. Andrey Chernikov for all his classes that provided a solid theoretical foundation for my training as a graduate student and Desh Ranjan for both of the Algorithms classes that had a great balance of challenge and enjoyment.

In addition, I would like to thank Fotis Drakopoulos for helping me adjust when I first arrived in the United States but also for the insights he gave me when it comes to mesh generation in general. CDT3D, the central library used in this thesis, was initially created as part of his thesis. I am really happy and honored to be able to contribute and extend this project. Also, I thank my colleagues and friends Juliette Pardue, Daming Feng, Jing Xu, Eleni Adam, and Kevin Garner for making our lab an enjoyable place to be and the opportunity to work on a number of projects with them. Thomas Kennedy for checking in periodically in our lab and igniting discussions about software engineering and programming languages. Olga Karadimou and Polykarpos Thomadakis, my very good friends and roommates, made life around Hampton Roads more enjoyable. On top of his friendship, I would like to thank Polykarpos for the countless hours we spent discussing about design and abstractions of his runtime project. Although its use in this thesis is limited, it drew, maybe subconsciously, my attention to the idea of separation of concerns and ultimately to the final chapter of this thesis which is derived from a joined paper.

I would like also to extend my thanks to a few commonly unsung heroes such as the people working at the CS main office and especially Ariel Sturtevant, Phyllis Woods, and Christy Chavis for taking care of our paperwork and always helping us with any issues. Special thanks to Danella Zhao the Graduate Program Director at the time of my graduation for making sure everything is submitted on time. Moreover, I would like to thank the Office of Visa & Immigration Service Advising for taking special care of the international students throughout our studies. Special thanks also to the HPC group of ODU and in particular to Terry Stilwell, Min Dong, Wirawan Purwanto, and John Pratt for promptly taking care of all our questions and issues regarding the high performance clusters of ODU.

Last but not least, I would like to thank my parents and brother for their continuous support and love throughout my studies and Maria for her companionship and support. I would like to extend my thanks to all my friends back in Greece who kept the communication between us alive and for coming up with ideas for get-together online events.

This research was sponsored in part by the NASA Transformational Tools and Technologies Project (NNX15AU39A) of the Transformative Aeronautics Concepts Program under the Aeronautics Research Mission Directorate, NSF grant no. CCF-1439079, the Richard T. Cheng Endowment, SURA grant no. CNF19-04 & CNF20-01, the Modeling and Simulation and the Dominion Scholar Fellowships of Old Dominion University. Experiments were supported by the Research Computing clusters at Old Dominion University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

CHAPTER 1
INTRODUCTION

Among the cornerstones of the scientific method, is the use of experiments to validate or contradict a hypothesis. Due to the intricacies and the cost of physical experiments, numerical simulations are often utilized as an alternative. One of the most popular numerical methods for simulations is the Finite-element Method (FEM) [268] which is used for obtaining numerical solutions to differential equations arising from several fields such as fluid flows, structural analysis, and registration of medical data.

Mesh Generation, i.e., the process of discretizing the problem space into primitive/simpler geometric shapes such as triangles and tetrahedra, plays a central role in FEM. A mesh allows decomposing a complex domain into elementary elements where approximation is possible. Moreover, it enables capturing local features of the solution by altering the fidelity of the approximation locally. The fidelity at which one captures the features of the numerical solution during a simulation directly relates to the local size of the mesh, thus making the need for mesh size variation a necessity. However, the locations of these features are not always known *a-priori*. This vicious cycle can be broken by the use of error-based mesh adaptation [16]. Error-based mesh adaptation modifies a mesh based on some error-estimator so that it can accurately capture features of the underlying solution automatically.

A subfield of physics that FEM and mesh adaptivity are used extensively is the Computational Fluid Dynamics (CFD) [10], which is a critical component of the design and analysis of aerospace vehicles. CFD is concerned with simulations involving the flow of a fluid (liquid or gas), and its interaction with contact surfaces. A primal example is the interaction of aircraft components with the atmosphere at different flight conditions. In 2014, NASA published the CFD Vision 2030 Study [237], a technical report that includes findings and recommendations for advancing the capabilities of the CFD simulations in the future. Among the findings of the study is that: *"Mesh generation and adaptivity continue to be significant bottlenecks in the CFD workflow, [...] Additionally, adaptive mesh techniques offer great potential, but have not seen widespread use due to issues related to software complexity, inadequate error estimation capabilities, and complex geometries."* In [203] the authors document the current status of mesh adaptation and provide recommendations in order to achieve the goals set by the CFD Vision study. Among them is to incorporate current parallel mesh adaptation methods with load-balancing techniques in order to be able to achieve good performance on the available and upcoming architectures. In-line with this recommendation is the *Telescopic Approach* for CFD simulations that was proposed in [61]. The *Telescopic Approach* lays down a design that allows to exploit the concurrency that exists at multiple levels in parallel and adaptive simulations. The design spans across the multiple memory hierarchies of a High Performance Computing (HPC) machine (shared, distributed-shared (DSM), distributed with or without out-of-core capabilities) and maps different algorithmic layers to the appropriate level of memory based on the intensity of communication between the meshing kernels.

1.1 AIM OF THIS WORK

Our goal is to create a building block that will serve as the core mesh adaptation module of the *Telescopic Approach* in the context of FEM simulations. Building upon previous work (*CDT3D*) [82], we design and implement a method for parallel mesh adaptation that offers new features in terms of functionality, performance and portability. In particular, we deliver a parallel mesh adaptation kernel that it is designed to be a building block of the *Telescopic Approach* and will run within the limits of a multi-core node. Moreover, to aid towards meeting the requirements set in [237] and the recommendations of [203], we interfaced *CDT3D* with a Computer-Aided-Design (CAD) kernel in order to enable handling curved geometries.

Our process is guided by the following parallel mesh generation attributes [62, 250]:

1. **Stability** is the requirement that the quality of the mesh generated in parallel must be comparable to that of a mesh generated sequentially [59]. The quality is defined in terms of the density and shape of the elements evaluated by some quality measure, and the number of the elements (fewer is better).

2. **Reproducibility** which we introduced in [62]. It is separated into two forms. *Strong Reproducibility* requires that the mesh generation code, when executed with the same input, produces identical results under the following modes of execution: (i) continuous without restarts, and (ii) with restarts and reconstructions of the internal data structures. *Weak Reproducibility* requires that the mesh generation code, when executed with the same input, produces results of the same quality under the following modes of execution: (i) continuous without restarts, and (ii) with restarts and reconstructions of the internal data structures.

3. **Robustness** is the ability of the software to correctly and efficiently process any input

data. Automation is critical for massively parallel computations, because operator intervention is impractical.

4. **Scalability** is the ability of a method to obtain speedup proportional to the number of processors [115]. Amdahl's law [8] suggests that the speedup is always limited by the inverse of the sequential fraction of the software. Therefore, all nontrivial stages of the computation must be parallelized to leverage the current and emerging architectures designed to deliver a million- to billion-way concurrency.

5. **Code Reuse** is a result of a modular design of the parallel software that builds upon previously designed sequential or parallel meshing code, such that it can be replaced and/or updated with minimal effort. Code Reuse is feasible only if the code satisfies the Reproducibility criterion.

1.2 OUTLINE

The rest of this document is organized as follows. Chapter 2 presents the two major pillars of our work. Specifically, Section 2.1 provides an extensive review of past and current parallel mesh generation methods. Moreover, it suggests a taxonomy that classifies the methods based on attributes pertinent to parallel mesh generation and in the context of the *Telescopic Approach*. Section 2.2 provides a concise description of notions related to metric spaces, a core component of our approach to parallel mesh adaptation.

Chapter 3 describes the implementation of our method in detail, highlighting the improvements in terms of both functionality and performance. In Chapter 4, we focus on the first four of the above criteria by evaluating our method on a series of different cases. In particular, Section 4.1 presents data on benchmarks that target a fixed mesh resolution and compares our results with state-of-the-art parallel mesh adaptation methods. Also, in Section 4.2, we evaluate our method within an end-to-end simulation pipeline, where the resolution of the discretization increases iteratively and compare our results with similar analyses present in the literature.

Having presented data on both our functionality and performance improvements we turn our attention into the *Code-reuse* aspect and in particular to the question of how one can future-proof [99] such a mesh generation application. Since hardware and software evolves rapidly, traditional approaches to thread and load balancing often suffer a cost when ported to a new environment. In Chapter 5, we present our attempt to solve this issue based on the notion of *separation of concerns* [77]. The presented approach decouples *functionality* aspects of mesh generation codes from *performance*. In particular, it illustrates a method and presents performance data regarding the use of tasking in place of manually managing threading and load balancing.

In summary, the goal of this dissertation is to create a new parallel anisotropic mesh adaptation method that can serve as building block for scalable parallel mesh generation. Based on our previous experience, we will use the *Telescopic Approach* (described in detail in the next chapter) as a case study. The contributions of this thesis are:

(C_1) A parallel mesh adaptation method with high parallel efficiency on a single multi-core node.

(C_2) The method exhibits comparable quality and performance against state-of-the-art methods.

(C_3) The parallel mesh adaptation method can interface with a CAD kernel allowing to accept a wider variety of inputs.

(C_4) Validation of the method within an adaptive pipeline.

(C_5) A General Tasking Framework that aids towards separating the concerns of functionality from performance for speculative parallel mesh generation methods.

CHAPTER 2

BACKGROUND

This thesis builds on top of two pillars. First is *Parallel Mesh Generation*, the process of discretizing the computation space using geometrical objects such as triangles and tetrahedra while utilizing single or multiple machines in parallel. Section 2.1 presents a thorough review of several parallel mesh generation methods and suggests a taxonomy to classify these methods.

The second is that of *Metric Spaces* and in particular, their use in the context of mesh adaptation. A metric space is a set of objects, three-dimensional points for example, along with a function that defines how to measure distances between its objects. At first regard, it may seem a very simplistic construct, however, one can use it to guide the process of mesh adaptation. Section 2.2 provides all the definitions and properties of metric spaces that are pertinent to this thesis.

2.1 PARALLEL MESH GENERATION PREVIOUS WORK REVIEW

There is a plethora of parallel mesh generation methods that have been implemented over the past decades. Still, there is no up-to-date comprehensive taxonomy and classification of them introduced over the last 15 years. In this section, we build upon previous work in this subject [57] and extend it by augmenting more classification attributes and including new methods that appeared in the literature since then.

There are several parallel mesh generation methods that utilize structured [45], hexahedral [195], and octree [39] meshes but these are not covered in this review neither are mesh multiplication [136] methods [126, 193, 194, 255] that split each element in self-similar elements. Three-dimensional parallel triangulation methods [23, 27, 93, 141, 155] that are designed to update the convex hull of a fixed point set instead of producing a high quality mesh suitable for simulations are also excluded. Parallel methods that rely solely on mesh smoothing [24, 96, 112, 222] are not covered as well. Instead, we focus on three-dimensional tetrahedral mesh generation and mesh adaptation methods and classify them according to a number of attributes described below.

2.1.1 CLASSIFICATION ATTRIBUTES

Our classification builds on top of the observation that parallel mesh generation is a memory intensive operation [12] and therefore communication is a crucial factor when it comes to characterizing a parallel mesh generation method. This classification is in-line with the *Telescopic Approach* suggested in [61] which applies a combination of decomposition techniques for current and emerging architectures with multiple memory/network hierarchies as shown in Figure 1. Parallel mesh generation methods often (over-)decompose the original mesh generation problem into n smaller sub-problems, which are solved (i.e., meshed) concurrently using $n \gg p$ cores [57]. The amount of communication required when solving the generated sub-problems defines the *coupling* between them. In particular, methods that require high amount of communication between the different meshing tasks are categorized as *Tightly-Coupled* (lower left section of Figure 1). Methods that reduce the coupling of the meshing tasks by grouping them into (partially) independent "super tasks" are characterized as *Partially-Coupled* (middle-left section of Figure 1). Methods that decompose the problem into (almost) independent tasks that require only minimal (or no) interaction between them are classified as *Loosely Coupled* (middle-right section of Figure 1). Finally, methods that generate independent meshing tasks where no communication is required are categorized as *Decoupled* (upper right section of Figure 1).

The design of the *Telescopic Approach* is part of a long term goal for parallel mesh generation and adaptation at the Center for Real-Time Computing (CRTC)[1] to achieve and sustain concurrency on current and emerging systems. To accomplish this goal, concurrency is exploited at different scales (levels) corresponding to the latency and the bandwidth of different network/memory hierarchies in order to orchestrate communication and synchronization.

The implementation of the Telescopic Approach relies on multiple abstractions used in the parallel mesh generation community [57]: element, cavity, data-region, and subdomain. These abstract data types vary in granularity and complexity (i.e., type and size of the data structures) and type/intensity of communication/synchronization required to implement their basic operations. The type/intensity of communication/synchronization determines their mapping to different layers of memory/network hierarchy. For example, concurrency at the element or cavity level is permitted only in the shared memory of the cores within a single-chip, bulk and locally synchronous exchange of data among data-regions is permitted

[1]`https://crtc.cs.odu.edu` (Accessed 2021-06-01).

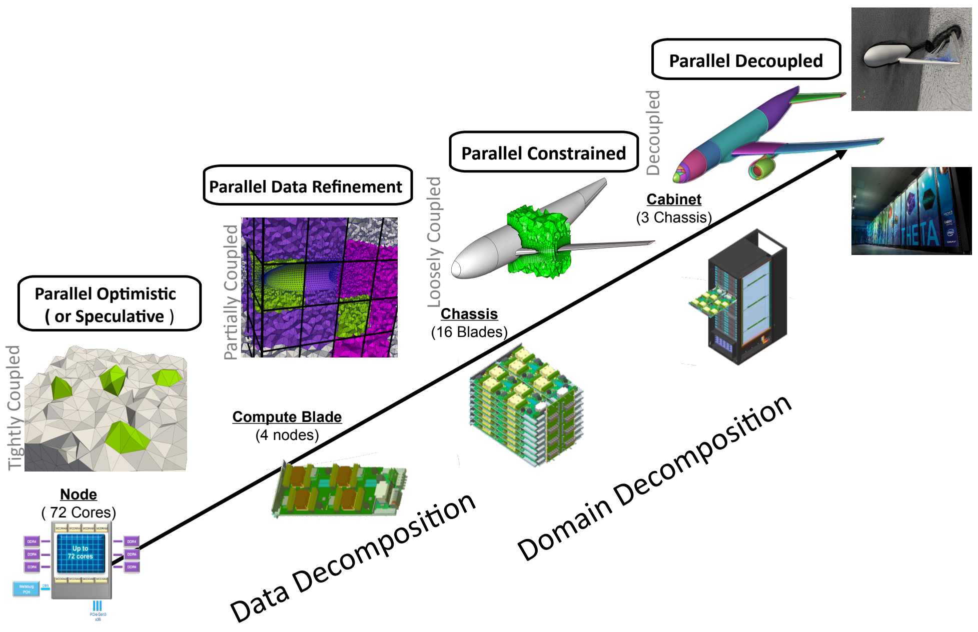


Fig. 1: The *Telescopic Approach*.

only within the distributed shared memory of a few nodes and asynchronous communication of data-buffers is permitted over distributed memory of several hundreds of nodes and/or tens of racks. Given these constraints, from the chip (lower left in Figure 1) to the distributed-memory node levels (upper right in Figure 1), the *Telescopic Approach* deploys:

1. Parallel Optimistic (PO) layer [95, 187], which is a *Tightly-Coupled* method, and is designed to explore concurrency at the CPU level (Node in Figure 1) within the limits of shared memory, using speculative/optimistic execution (explained below). The shared memory domain allows sustaining high volume of communication at low cost.

2. Parallel Data Refinement (PDR) layer [48, 62], which is a *Partially-Coupled* method, and targets multiple CPUs across different nodes (Compute Blades in Figure 1). It can exploit both shared and distributed memory hardware and relies on decomposing the data (versus the mesh) in such a way that limits the communication to bulk-synchronous steps.

3. Parallel Constrained (PC) layer [52], which is a *Loosely-Coupled* method, and targets nodes across different Compute Blades (Chassis in Figure 1) and requires only small

amounts of asynchronous communication. This method relies on domain decomposition methods for constructing its subdomains.

4. Parallel Decoupled (PD) layer [152], which is a *Decoupled* method, and targets nodes across different Chassis (Cabinets in Figure 1) and requires no communication.

In summary, *Coupling* can be defined as:

Coupling: Measures how much the parallel tasks depend on each other. The coupling can be *Tight*, *Partial* or *Loose*. If no dependency between the parallel tasks exists, a method is called *Decoupled* [57].

This is the main focus of this work since it the most general and can be abstracted. The rest of the classification attributes are defined as follows:

Synchronization: Refers to the level the execution units (threads or processes) are synchronized. It can be *Local*, including only a few of the execution units or, it can be *Global*. Global synchronization involves barriers or some other form of collective blocking operation. It should be noted that initialization and finalization of the parallel procedure are excluded since they are unavoidably a blocking operation. Methods that can reach a common state without blocking for each other while still exchanging information are called *Asynchronous* [52].

Granularity: Refers to the size of the *atomic* unit of work. In the literature different granularity levels have been used for the same atomic units of work based on the context. For example, the parallel Delaunay method of [95] is considered by the authors a fine-grained approach since the atomic unit is a cavity. However, later in [94] a finer level of granularity is exploited by introducing lower-level atomic units within a single cavity. For this study, we will consider a single point, element or cavity to be the *Fine* level while, a group of cavities or mesh subdomains will be considered a *Coarse* level of granularity.

Method: Refers to the mesh operation that is parallelized. Historically, mesh multiplication was used [11], Delaunay and Advancing Front methods have also been used extensively [247]. Some approaches (e.g., [114, 128, 181, 197]) which favor modularity have separated the mesh generation procedure into multiple independent modules such as point insertion, local reconnection, edge contraction, etc., while others [159] use a single operator for all of them.

Programming Model: Describes the main toolkit for the implementation of the parallel code: POSIX threads (Pthreads), Open Multi-Processing (OpenMP)[2], Message Passing Interface (MPI)[3], etc.

Decomposition: An essential component of every parallel algorithm. For mesh generation *Data* and *Domain* decomposition are the most common [57]. The distinction is sometimes difficult since a group of vertices and tetrahedra can act both as data and a subdomain. The difference is in the way the decomposed data are handled. Domain decomposition methods treat subdomains as meshes that can be processed independently. Mesh operations are constrained within the subdomains and, depending on the coupling, some care may need to be taken for shared elements between subdomains. Data decomposition methods do not take into account the geometry of the input. Instead, they treat elements as shared data and use different methods to resolve read-write dependencies that can arise during different mesh operations. For domain decomposition, we have two subcategories [57].

Discrete: Refers to the decomposition of an existing volume mesh. Graph-based methods applied on the dual mesh such as METIS [138], Zoltan [74], and Scotch [54] belong to these. Another approach is to use general sorting-based methods such as Cuthill-McKee [68] or more specialized ones such as Hilbert curves [206], PQR [58], etc.

Continuous: Refers to decomposition of the initial surface mesh by creating continuous separators. It can be thought as a geometrical construction problem. If done correctly, it can guarantee very good quality of separators (in terms of dihedral angles for example). For two dimensions a method based on Medial Axis [31] has been already presented in [151]. For three dimensions, to the best of our knowledge, generating a Medial Axis object for general inputs is still an open problem, although commercial tools for generating it exist [37, 103].

Progressive: Relates to how the parallel process is bootstrapped and how load balancing is handled. To the best of our knowledge, it appears in its first form as "variable granularity" in [51] and was later described in detail in [62]. A method that starts from very few units of work and generates more upon runtime which are then shared dynamically among execution units is considered to be *progressive*. On the other hand,

[2]`https://www.openmp.org` (Accessed 2021-05-31).
[3]`https://www.mpi-forum.org` (Accessed 2021-05-31).

if the starting point is a mesh decomposition with a predetermined number of work units, or if load balancing is done by globally redistributing the mesh, then the method is called *non-progressive*.

Figure 2 summarizes the attributes used in this section. For the rest of the section, several parallel mesh generation methods will be presented grouped with respect to their coupling. Each method will be described briefly and then categorized based on the above criteria. To ease cross-referencing we also introduce a naming scheme: `Research Group`$_{\texttt{name}}$ where `name` is the name of the software (if it is defined in the cited literature) or one of its distinctive characteristics based on the above criteria.

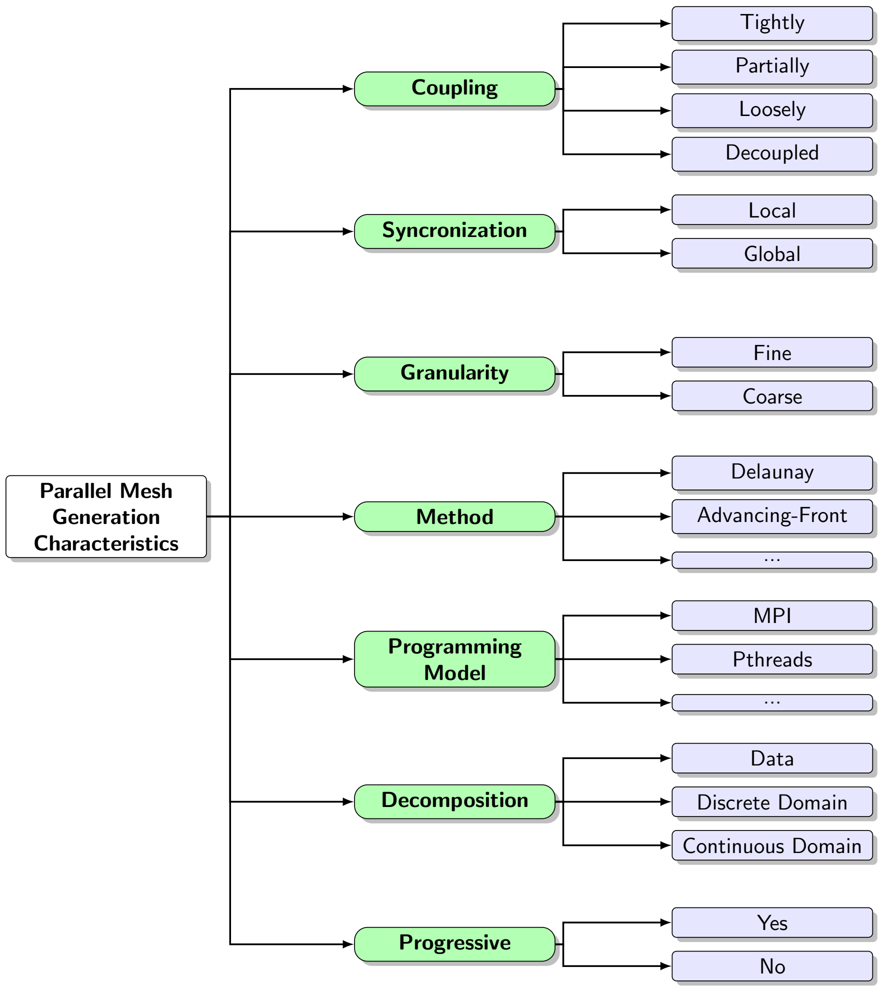


Fig. 2: Attributes of the taxonomy presented in this section.

2.1.2 TIGHTLY-COUPLED METHODS

Tightly-Coupled methods are characterized by intense communication. For this class of methods, the communication can be direct through messages or indirect through accessing regions of shared memory, which due to false sharing and cache invalidation creates overheads. Moreover, synchronization primitives and constructs such as locks and barriers add to the overall overhead. Many of the approaches in this section employ *Speculative* execution (also known as *optimistic*). *Speculative* execution is a technique used in several applications ranging from processors [249] to databases [143]. It allows for the exploitation of more concurrency out of a problem by executing steps of a procedure ahead of time prior to resolving data dependencies between the steps themselves. In the case where steps are not needed, pre-computed results may either be disregarded or additional steps may be required to *rollback* to the previous state of the process. The correctness of this scheme, in the context of parallel algorithms, has been proven in [135] with the introduction of the notion of *Virtual Time* and validated in the context of Parallel Discrete Event Simulations within the *Time Warp* system.

$\texttt{CRTC}_{\texttt{PODM 1.0}}$: One of the first tightly coupled methods for mesh generation is presented in [59, 188]. This method is based on the Delaunay kernel for introducing new points into the mesh. This approach exploits parallelism using MPI. The first step of this approach is a decomposition of the initial mesh using METIS [138]. Once the mesh is distributed, each meshing process executes the Delaunay kernel on a sub-mesh of the decomposition. This method treats each subdomain as a Delaunay sub-mesh in contrast to other methods (see for example [52]) that treat the subdomain as a Constrained Delaunay sub-mesh, so the Delaunay cavity can expand between two subdomains. In this case, a distributed breadth-first search is used to collect all the elements of the cavity. The meshing process will then attempt to lock all the elements in the cavity in order to perform the point insertion. It is of importance that the dependencies are evaluated upon runtime and not as a pre-processing step. This characteristic gives this method the name speculative or optimistic. It is also worth mentioning that the method uses locks on vertices instead of the tetrahedra of a cavity since (a) the vertices of a cavity stay the same when a Delaunay point insertion is performed and (b) pathological cases of neighboring cavities are avoided. Although the number of rollbacks was low, the high number of messages resulted in sub-optimal performance. Still, this approach acts directly upon touched data which improves cache utilization and allows tolerating more than 80% of system latencies. The classification of this method appears in Table 1.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{PODM 1.0}}$	Tightly	Local	Coarse	Delaunay	MPI	Data	No

TABLE 1: Classification of $\texttt{CRTC}_{\texttt{PODM 1.0}}$ based on the criteria of this section.

$\texttt{CRTC}_{\texttt{PODM 2.0}}$: In [93] a tightly-coupled Parallel Delaunay Triangulation algorithm was presented. Cavity and expansion and locking of vertices are performed similarly to the previous approach but in a shared memory setting utilizing atomic operations. Later in [95] this approach was extended to a Parallel Delaunay Refinement method that was able to handle medical image data and optimized for Distributed Shared Memory machines (DSM) using a hierarchical load balancing algorithm between the threads. This algorithm was designed to reduce the remote-memory access and allowed the method to achieve more than 82% weak-scalability speedup on 144 cores. Although the performance improvement is significant, there are signs that the speedup of this method deteriorates at a higher number of cores. The source of inefficiency was not a problem inherent to the approach but rather a limitation of the hardware. In particular, on DSM machines the cost of remote memory access is high because the shared memory address space is emulated using low-level software. Moreover, the physical distance between the nodes incurs unavoidably an overhead for every remote memory access. A locality-aware implementation [87] improved this issue but only by a moderate amount. The classification of $\texttt{CRTC}_{\texttt{PODM 2.0}}$ appears in Table 2. The progressive characteristic comes from the fact that this method uses work stealing to continuously equi-distribute the load among the threads.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{PODM 2.0}}$	Tightly	Local	Fine	Delaunay	Pthreads	Data	Yes

TABLE 2: Classification of $\texttt{CRTC}_{\texttt{PODM 2.0}}$ based on the criteria of this section.

CRTC$_{\texttt{CDT3D}}$: The speculative/optimistic approach has been applied to operations beyond Delaunay point insertion. In [85] the authors use the same underlying idea to parallelize a local reconnection method. As before, the cavity is evaluated and locked upon runtime, and if the locking is successful an appropriate topological flip is performed in order to improve the objective function. In contrast with the previous algorithms, where the Delaunay kernel is the only meshing operation in this work the local reconnection was only a step of a wider set of operations, and since only one operation was parallelized this method performs global synchronization between each mesh operation. This fact as well as the well-known Amdahl's law [8] that is applicable to every parallel code, constrained the efficiency of the code. Still, compared to a state-of-the-art local reconnection approach, the proposed method enhances user productivity by lowering the execution time significantly while delivering comparable quality. The classification of CRTC$_{\texttt{CDT3D}}$ appears in Table 3.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
CRTC$_{\texttt{CDT3D}}$	Tightly	Local	Fine	Flips	Pthreads	Data	Yes

TABLE 3: Classification of CRTC$_{\texttt{CDT3D}}$ based on the criteria of this section.

UBC$_{\texttt{Edge-Face Flip}}$: Another method that utilizes the speculative approach for face and edge flips in a slightly different manner is presented in [264]. In this method, the flip operation is separated in stages which are linked utilizing global synchronization between the threads. First, all candidate flips are checked with respect to whether they optimize the objective function or not, in parallel, with no synchronization. A face and edge migration step follows in an attempt to equi-distribute the load between the threads. The cavities are then locked using an optimistic approach. A second edge and face migration step smooths out any load imbalances and finally, the flips are performed in parallel with no synchronization. The classification of this method appears in Table 4.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`UBC`$_{\texttt{Edge-Face Flip}}$	Tightly	Global	Fine	Flips	OpenMP	Data	Yes

TABLE 4: Classification of `UBC`$_{\texttt{Edge-Face Flip}}$ based on the criteria of this section.

`UChile`$_{\texttt{Tightly}}$: In [217] the authors create a parallel mesh generator utilizing the Longest edge propagation path (Lepp) [214] algorithm speculatively. Lepp recursively splits elements by giving priority to their longest edge. Each application of Lepp on an element evaluates a propagation path that corresponds to a group of tetrahedra that are going to be split. Utilizing this approach in a multi-threaded environment can result in intersecting propagation paths. To remedy this issue, elements contained in a path are locked speculatively. Failure to lock any required element results in releasing all currently locked elements and aborting the refinement of the current element. Table 5 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`UChile`$_{\texttt{Tightly}}$	Tightly	Local	Fine	Lepp	Pthreads	Data	Yes

TABLE 5: Classification of `UChile`$_{\texttt{Tightly}}$ based on the criteria of this section.

`MIT`$_{\texttt{Delaunay}}$: To the best of our knowledge, the first parallel Delaunay method was presented in [192]. The input for this method is a coarse mesh which is decomposed into subdomains using axis-aligned planes. The mesh generation process is based on alternating between a point insertion stage and a load balancing/migration stage. For points whose Delaunay cavity is completely inside a subdomain, the point insertion is performed as usual. If however, the cavity expands between subdomains, the process attempts to globally lock the remote elements. If the locking is successful, a request is sent to the owners of the elements in order to bring them locally. To reduce the idle time while waiting for the requests to be satisfied, each process inserts a number of points up to a pre-specified limit while, keeping a buffer of pending points allowing thus to overlap some communication with computation. The classification of this method is presented in Table 6.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{MIT}_{\texttt{Delaunay}}$	Tightly	Global	Coarse	Delaunay	MPI	Data	No

TABLE 6: Classification of $\texttt{MIT}_{\texttt{Delaunay}}$ based on the criteria of this section.

2.1.3 PARTIALLY-COUPLED METHODS

Partially-Coupled methods cover the majority of the current and past mesh generation methods. They are characterized by a moderate amount of communication, far smaller than the one of the tightly-coupled methods. Due to this reason, the *Telescopic Approach* assigns this method right above the tightly coupled methods. The reduced amount of communication they offer allows them to be implemented efficiently both at the shared and the distributed memory level. This section will be further divided based on the decomposition method.

2.1.3.1 Partially-Coupled Methods based on Data Decomposition

$\texttt{GMU}_{\texttt{Data Decomp.}}$: The first, to the best of our knowledge, multithreaded partially-coupled mesh generation method utilizing data decomposition was presented in [156]. The method is based on the advancing-front point generation method and local reconnection is used to improve the quality of newly-created elements. Data are decomposed by decomposing the active front using an octree. Within each octree leaf, points are generated in parallel by advancing from the active faces. Inter-leaf faces are inactive initially. A shift of the octree along the axes is used to transform the inter-leaf faces to intra-leaf. For edge flipping, the objective function is evaluated in parallel for each candidate flip, which is straightforward since it is only a read operation. The topological transformation is then performed sequentially. According to the author of [156], this decoupling of evaluation and application of the flip was selected because the objective function evaluation is the most resource-heavy part of the local reconnection algorithm. It should be noted that in this work, there is an effort to introduce the *Progressive* characteristic by estimating the work imbalance between the leaves and splitting them in an effort to equi-distribute the workload by over-decomposing the input. The classification of this method appears in Table 7.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{GMU}_{\texttt{Data Decomp.}}$	Partially	Global	Coarse	A. Front	c$doacross	Data	Yes

TABLE 7: Classification of $\texttt{GMU}_{\texttt{Data Decomp.}}$ based on the criteria of this section.

$\texttt{CRTC}_{\texttt{PDR 1.0}}$: A few years later, the Parallel Delaunay Refinement (PDR), (later renamed to Parallel Data Refinement) method emerged [50, 53]. The main focus of this method was scheduling Delaunay refinement in different regions of the mesh in a way that rollbacks were guaranteed to be eliminated. In this way, the correctness was handled by the scheduler of the method, thus allowing the reuse of Commercial-off-the-self (COTS) software to generate the mesh. The three-dimensional implementation utilized TetGen [235] as a Delaunay mesh generator and an octree constructed over an initial coarse mesh was used for data decomposition. Moreover, since the octree was used only as a scheduling mechanism there were no artificial boundaries imposed which could affect the mesh quality. The fact that Delaunay refinement follows strict mathematical rules for the evaluation of the cavity, allowed PDR to express the octree leaf size as a function of the element size of the initial coarse mesh. In this way, a pre-processing stage of the initial coarse mesh involving refinement up to a specific size, guarantees that no cavity will expand beyond the neighboring leaves in the subsequent parallel step, thus making possible to schedule leafs that share no common neighbors concurrently. This method utilizes local synchronization since, only the refinement of neighboring leaves need to be synchronized. Since the work unit in this method is a region enclosed by a leaf, the method is considered coarse-grained. Later version of this algorithm [62] ($\texttt{CRTC}_{\texttt{PDR 2.0}}$) refines the octree as the same time with the mesh producing thus more work units as refinement progresses offering the *Progressive* attribute discussed earlier. The PDR method was revisited as part of the *Telescopic Approach* but this time utilizing $\texttt{CRTC}_{\texttt{PODM 2.0}}$ which is presented in Section 2.1.2. The shared memory implementation was able to achieve about 160 speedup on 256 cores [89] while the hybrid distributed memory implementation reached about 450 speedup on 900 cores [88][4]. It should be noted that this method utilizes parallelism at two levels. The first is on the PDR level with the scheduler assigning for refinement as many leaves as it is possible based on the available work and workers. At the second level, each worker executes an instance of $\texttt{CRTC}_{\texttt{PODM 2.0}}$ which is multithreaded, thus gaining a significant speedup by the combined efficiency of both methods. Table 8 presents the classification for all three variants of PDR.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{PDR 1.0}}$	Partially	Local	Coarse	Delaunay	Pthreads	Data	No
$\texttt{CRTC}_{\texttt{PDR 2.0}}$	Partially	Local	Coarse	Delaunay	Pthreads	Data	Yes
$\texttt{CRTC}_{\texttt{PDR.PODM}}$	Partially	Local	Coarse	Delaunay	MPI+Pthreads	Data	No

TABLE 8: Classification of $\texttt{CRTC}_{\texttt{PDR 1.0}}$, $\texttt{CRTC}_{\texttt{PDR 2.0}}$ & $\texttt{CRTC}_{\texttt{PDR.PODM}}$ based on the criteria of this section.

$\texttt{UBC}_{\texttt{Delaunay}}$: A scheduling approach for refinement similar to PDR but utilizing global synchronization was presented in [264]. Initially, a uniform octree is laid upon the mesh, and elements that fall within a leaf are assigned to the leaf for refinement. In contrast to the original PDR method, the refinement is performed in multiple passes where an independent set of leaves is selected and refined. To exploit more concurrency, a dynamic octree is created within each leaf, with its size initialized to the leaf itself. This "second-level" octree is refined progressively along with the mesh in order to provide more concurrency. It is worth noting that, the second level leaves are not refined to the desired size at once. Instead, several passes are applied, each one targeting elements only above a specific size. The reason is that before each pass the leaves may get split offering thus more concurrency to the system.

[4]In both cases speedup is measured in terms of weak scalability.

Table 9 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{UBC}_{\texttt{Delaunay}}$	Partially	Global	Coarse	Delaunay	OpenMP	Data	Yes

TABLE 9: Classification of $\texttt{UBC}_{\texttt{Point Insertion}}$ based on the criteria of this section.

$\texttt{UCLouvain}_{\texttt{Gmsh}}$: In [176, 211, 212], the authors separated the Delaunay mesh refinement into two steps: (a) point creation (circumcenter evaluation) and (b) point insertion. The separation of parallel refinement in two steps reduces the problem to that of parallel Delaunay triangulation of an *a-priori* known dataset which was first solved for 2D in [30]. In contrast to the previous methods, this approach performs a partitioning step of the candidate points by sorting them along space-filling curves. The mesh is also partitioned using a similar approach. Each thread then attempts to insert points starting from a different part of the curve thus, having a low probability of overlapping with another cavity. If a cavity or a point location walk crosses the boundary of a partition the insertion is aborted. If there are any rejected points, the data are re-partitioned by shifting and rotating the curves. This method was recently extended to flip-based mesh improvement operations [175]. Both the refinement and the mesh improvement operations are present in the open-source package Gmsh [111]. The same approach was applied later to a parallel local reconnection and smoothing method by a different group [229]. The classification of $\texttt{UCLouvain}_{\texttt{Gmsh}}$ appears in Table 10.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{UCLouvain}_{\texttt{Gmsh}}$	Partially	Global	Coarse	Delaunay & Flips	OpenMP	Data	No

TABLE 10: Classification $\texttt{UCLouvain}_{\texttt{Gmsh}}$ based on the criteria of this section.

Up to this point all the partially-coupled methods decompose the input in order to eliminate or reduce the chance of a cavity overlap. A different approach first presented in [137] for bisection methods, in [96] for mesh optimization methods and later analyzed theoretically in [239,254] for Delaunay methods is to directly compute all cavities and choose an independent set of these. Although, this approach incurs a significant overhead upfront, the cost is amortized by the absence of synchronization during the application of the mesh operations.

`Imperial`$_{\texttt{pragmatic}}$[5]: Parallel execution in the *Pragmatic* library utilizes a partially-coupled approach for distributed memory machines. The domain is decomposed prior to re-meshing, mesh vertices are distributed to the different subdomains. The re-meshing algorithm is then applied to all the subdomains independently. Subdomain interfaces are fixed during refinement to prevent inconsistencies while avoiding frequent communications. Once the refinement phase is completed on all subdomains, a re-partitioning phase is used to reduce intersections between the old and the new interfaces. The re-partitioning is carried out using a custom diffusion-based algorithm, where the interfaces are slightly shifted into one of its neighboring domains. Synchronization across partitions is achieved using message passing and is implemented by MPI. This process is repeated until good element quality is obtained everywhere in the domain. *Pragmatic* modifies the input mesh through a series of local edge-based mesh manipulations. First, iterative applications of coarsening (edge collapse), edge/face swapping, and refinement (edge splitting) is used to optimize the resolution and the quality of the mesh. Second, an element-shape-constrained Laplacian smoothing step fine-tunes the mesh element shape measure. The element internal shape function that is optimized is the functional defined in [153]. New surface points are optionally projected onto a CAD model through the EGADS API [121]. *Pragmatic* aims at generating quality meshes for a wide range of numerical simulations, notably for geophysics applications, and it has been integrated with the PETSc library [18, 21] *Pragmatic* initially implemented a tightly-coupled approach at the element level using a hybrid (MPI+Threads) programming model [114]. However, following the essentially message passing nature of PETSc and other packages from its environment, *Pragmatic* parallel development priorities have been redefined and now is relying on message passing programming paradigm everywhere. More than crafting a perfectly scalable code, *Pragmatic* aims at being good enough in the various situations that arise from real-world both 2D and 3D adaptive simulations. An open-source

[5]Software description reflects original author's perspective. Received as part of personal communication with Nicolas Barral on May 2018.

implementation of this method is available at [113]. Table 11 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`Imperial`$_{\texttt{pragmatic}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Data	No

TABLE 11: Classification of `Imperial`$_{\texttt{pragmatic}}$ based on the criteria of this section.

`RPI`$_{\texttt{Omega\_h}}$[6]: *Omega_h* is an open-source mesh adaptation library [128, 130], developed by Rensselaer Polytechnic Institute and subsequently by Sandia National Laboratories. The core algorithm in *Omega_h* consists of two loops: one loop of alternating between edge splitting and edge collapsing to satisfy length, followed by another loop that uses edge swapping and edge collapsing to improve element shape. For parallel execution, each pass of cavity modifications is executed in parallel on a distributed mesh. First, a maximal independent set of cavities to modify is selected, using a modified version of Luby's algorithm [168]. A single layer of element ghosting (i.e., using data replication) along subdomain boundaries is needed so that for every cavity there exists a subdomain with all the elements required for the cavity computation and triangulation. After the completion of a triangulation phase and before the creation of the next independent set, *Omega_h* deploys an element trimming phase where duplicate elements are eliminated. This guarantees the partial components (i.e., subsets of independent cavities) of the next independent set are entirely within single MPI rank. Third, cavities are modified locally within each MPI rank using a multithreaded approach, which creates a new mesh structure to represent the modified mesh.

[6]Software description reflects original author's perspective. Received as part of personal communication with Daniel Ibanez on June 2018.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`RPI`$_{\texttt{Omega\_h}}$	Partially	Global	Coarse	Edge op. & Flips	MPI+Kokkos	Data	No

TABLE 12: Classification of `RPI`$_{\texttt{Omega\_h}}$ based on the criteria of this section.

2.1.3.2 Partially-Coupled Methods based on Domain Decomposition

For the next two methods we use descriptions directly taken from [251] that reflect the understanding of the original authors.

`LaRC`$_{\texttt{refine}}$: *refine* relies on the implementation of a partially-coupled approach that exploits coarse-grain parallelism at the subdomain level using domain decomposition and a homogeneous programming model in a distributed memory environment. The parallel execution strategy is described in [197]. The interior portion of each subdomain is modified in parallel while the border regions between subdomains are fixed. Elements that span boundaries and need to be modified to improve metric conformity are marked for future refinement. A combined load-balancing and migration is performed to equalize the number of nodes on each partition while penalizing elements marked for modification that span subdomains after migration. The re-partitioning step provides edge weights to either ParMETIS [223] or Zoltan [75] graph-based partitioning libraries. The current load-balancing and migration approach has improved parallel scaling properties over the transcript approach described in [197]. *refine* uses a combination of edge split and collapse operations proposed in [181] to modify long and short edges toward unity length in the metric. Node relocation is performed to improve adjacent element shape. A new ideal node location of the node is created for each adjacent element. A convex combination of these ideal node locations is chosen to yield a new node location update that improves the element shape measure in the anisotropic metric [3]. Moreover, *refine* utilizes pliant smoothing [134] improving significantly over the results presented in Ref. [250]. Geometry is accessed through the EGADS [119] and EGADSlite [120] application program interface. The source code *refine* is available at [198]. Table 13 presents the classification of `LaRC`$_{\texttt{refine}}$.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`LaRC`$_{\texttt{refine}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Discrete D.	No

TABLE 13: Classification of `LaRC`$_{\texttt{refine}}$ based on the criteria of this section.

`Boeing`$_{\texttt{EPIC}}$: *EPIC* uses a partially-coupled approach that exploits coarse-grain parallelism at the subdomain level in a distributed memory environment. Given the initial mesh, *EPIC* partitions the mesh into subdomains and performs a complete mesh operator pass consisting of refinement, coarsening, element reconnection, and smoothing operations on the interior of each subdomain while temporarily freezing the mesh at partition boundaries. After each mesh operator pass, *EPIC* updates the decomposition by shifting elements between subdomains. Subdomain re-balancing uses an optimization technique that attempts to maintain an equal work-load balance between subdomains while ensuring that frozen mesh edges near partition boundaries are moved to the interior of a subdomain with each re-balancing step. Multithreading can be used to parallelize the mesh operators at the subdomain level, but has only been implemented for a subset of mesh operations. This incomplete multithreading implementation has seen limited use to date. The *EPIC* anisotropic mesh adaptation process [181] provides a modular framework for anisotropic unstructured ,esh adaptation that can be linked with external flow solvers. *EPIC* relies on repeated use of mesh operator passes to modify a mesh such that element edge lengths match a given anisotropic metric tensor field. The metric field on the adapted mesh is continuously interpolated from the initial metric field. Several methods are available to pre-process the metric to limit minimum and maximum local metric sizes, control metric stretching rates and/or anisotropy, and ensure smoothness of the resulting distribution. In addition, the metric distribution can be limited relative to the initial mesh and/or to the local geometry surface curvature. The surface mesh is maintained on an IGES geometry definition [9] with geometric projections and a local regridding. *EPIC* is routinely used on production applications at the Boeing Company and has been applied on several workshop cases where the parallel implementation makes it practical for large scale problems [180, 182]. Table 14 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{Boeing}_{\texttt{EPIC}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Discrete D.	No

TABLE 14: Classification of $\texttt{Boeing}_{\texttt{EPIC}}$ based on the criteria of this section.

$\texttt{CEMEF}_{\texttt{Edge op.}}$: In [67] and in its anisotropic evolution [76] the authors employ an iterative method alternating between mesh adaptation and mesh re-partitioning. Each iteration performs mesh adaptation while keeping the subdomain boundaries fixed. The re-partition step redistributes the mesh elements by shifting the subdomain boundaries. Instead of the traditional approach that uses the dual graph for mesh partitioning, this method uses a diffusive algorithm that acts directly on the mesh. Weights based on the local mesh quality are used in order to migrate groups of low-quality elements in the same subdomain. Table 15 presents the classification of this method. This method has been applied [63,64] to the open-source software MMG [183] which resulted into ParMMG [184]. A similar approach by a different group is presented in [43].

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CEMEF}_{\texttt{Edge op.}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Discrete D.	No

TABLE 15: Classification of $\texttt{CEMEF}_{\texttt{Edge op.}}$ based on the criteria of this section.

$\texttt{GMU}_{\texttt{Discrete DD}}$: A two level parallel mesh generation algorithm combining MPI and OpenMP with a Parallel Advancing Front mesh generation algorithm is presented in [157]. This method decomposes an initial coarse mesh by applying recursive bisection and space-filling curves. The domains are refined using the parallel advancing front scheme of $\texttt{GMU}_{\texttt{Data Decomp.}}$ presented in the previous section. During subdomain migration, few layers of elements along the subdomain interfaces, which function as “buffer zones”, are sent as well. The extra layers aid towards eliminating low-quality elements between the boundaries of the subdomains. Finally, mesh improvement is applied in a decoupled fashion by fixing the elements between the subdomains and applying the operation only to the internal elements. Table 16 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{GMU}_{\texttt{Discrete DD}}$	Partially	Global	Coarse	A. Front & Flips	MPI+OpenMP	Discrete D.	No

TABLE 16: Classification of $\texttt{GMU}_{\texttt{Discrete DD}}$ based on the criteria of this section.

$\texttt{RPI}_{\texttt{Discrete DD}}$: In contrast to the methods presented so far, the method described in [4,72] deals with parallelism by creating mesh-specific data structures [228] designed to handle concurrent read and write access in a distributed memory environment. This method utilizes template-based refinement and edge-face swaps for quality improvement. Moreover, it has been extended for metric-based adaptation [219]. Each mesh operator pass is coupled with a global synchronization step where all the processes commit and receive modifications of the boundary elements. Table 17 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{RPI}_{\texttt{Discrete DD}}$	Partially	Global	Coarse	Templates & Flips	MPI	Discrete D.	No

TABLE 17: Classification of $\texttt{RPI}_{\texttt{Discrete DD}}$ based on the criteria of this section.

For the next method we use a description directly taken from [251] that reflects the understanding of the original author.

`INRIA`$_{\texttt{Feflo.a}}$: *Feflo.a* employs a partially coupled, coarse-grained approach that exploits parallelism at the subdomain level in a shared memory environment. The initial mesh is decomposed in multiple levels (i.e., domain decomposition). The initial volume is split and adapted in parallel while treating the interface between subdomains as a constrained surface. Once the initial subdomains are complete, a new set of subdomains is constructed entirely of the constrained interface elements of the previous subdomains. This process recurses until all the constrained elements are adapted [159]. *Feflo.a* is an adaptation code developed at INRIA[7] that can process manifold or nonmanifold surface and/or volume meshes composed of simplicial elements. It creates a unit mesh [158, 162] in two steps. The first step improves the edge length distribution with respect to the input metric field. Only classical edge-based operators (insertion and collapse) are used during this step. The second step is the optimization of mesh element shape measures with node smoothing and tetrahedra edge and face swaps. For the surface mesh adaptation, a dedicated surface metric is used to control the deviation of the metric and surface curvature. This surface metric is then combined with the input metric. New points created on the surface are evaluated on a (fine) background surface mesh and optionally on a geometry model via the EGADS application program interface (API) [119]. The classical edge-based operators are implemented by a unique cavity-based operator [164, 165]. This cavity-based operator simplifies code maintenance, increases the success rate of mesh modifications, has a constant execution time for many different local operations, and robustly inserts boundary layer mesh [163]. When the cavity operator is combined with advancing-point techniques, it outputs metric-aligned and metric-orthogonal meshes [160]. Table 18 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
`INRIA`$_{\texttt{Feflo.a}}$	Partially	Global	Coarse	Uniq. Cavity	Unix-sockets	Discrete D.	No

TABLE 18: Classification of `INRIA`$_{\texttt{Feflo.a}}$ based on the criteria of this section.

[7]`https://www.inria.fr/en` (Accessed 2021-06-01)

$\texttt{RPI}_{\texttt{Continuous DD}}$: Among the partially coupled methods, is also one that uses a continuous domain decomposition scheme and was presented in [73]. In particular, a continuous octree-based domain decomposition method is used away from the boundary of the mesh while close to the boundary the method utilized a discrete domain decomposition method. This method generates first a distributed octree over the initial surface mesh with the size of the terminal nodes governed by the spacing requirements of the problem. The internal octree leaves are then refined using templates that guarantee conformity with no communication. For the octree leaves that intersect the input surface, the method utilizes an advancing front technique to fill the space while keeping shared interfaces between subdomains fixed. The octree leaves that intersect the input surface are re-partitioned to make the fixed interfaces internal but also for load balancing purposes. Table 19 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{RPI}_{\texttt{Continuous DD}}$	Partially	Global	Coarse	A. Front & Templates	MPI	Continuous D.	No

TABLE 19: Classification of $\texttt{RPI}_{\texttt{Continuous DD}}$ based on the criteria of this section.

$\texttt{Brown}_{\texttt{Lepp}}$: The Lepp method is revisited in [42] where the authors use an asynchronous method for refining the mesh in parallel. The input for this method is a coarse mesh which is decomposed using the Chaco graph partitioning library [123]. Each process will then alternate between two states. First, refining the tetrahedra using the Lepp method. For propagation paths that span among different subdomains asynchronous messages will be send to neighboring subdomains that contain the point to be inserted along the shared boundary. During the second state, the method processes the incoming messages by inserting the suggested points achieving thus a conforming interface. The process continues until no further points are inserted. Table 20 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{Brown}_{\texttt{Lepp}}$	Loosely	Asynchronous	Coarse	Lepp	MPI	Discrete D.	No

TABLE 20: Classification of $\texttt{Brown}_{\texttt{Lepp}}$ based on the criteria of this section.

2.1.4 DECOUPLED METHODS

On the top of the *Telescopic Approach* are the decoupled methods which pre-process the initial input in a way that the generated subdomains can be refined with no need for further communication. Pre-processing often involves refining the interfaces between the subdomains so that they meet the target spacing. Moreover, in some cases extra care is taken to improve the quality of the separators and/or ensure they conform to the Delaunay property. An efficient method to produce separators in two dimensions involves the use of the Medial Axis object [31]. In three dimensions however, to the best of our knowledge, there is no parallel tetrahedral mesh generation method that takes advantage of it, although commercial tools for generating the 3D Medial Axis exist [37, 103].

$\texttt{INRIA}_{\texttt{Decoupled}}$: One of the first parallel mesh generation methods appears in [101]. Starting from a surface mesh this method computes a separator surface S using an Inertia Axis Decomposition method. A subset of the surface points V is then projected on S and series of operations involving a restricted Voronoi diagram, Dual Delaunay triangulation and convex hull evaluations are used to extract the edges of the original surface that cross S. These edges are refined to their final spacing and become part of the separator between the subdomains. Finally, the subdomains are refined using GHS3D [108] in each subdomain. The authors of this method note that in some cases the partitioning procedure may fail and it has to be restarted from scratch with a new surface separator. Table 21 presents the classification of the method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{INRIA}_{\texttt{Decoupled}}$	Decoupled	None	Coarse	Delaunay	-	Continuous D.	No

TABLE 21: Classification of $\texttt{INRIA}_{\texttt{Decoupled}}$ based on the criteria of this section.

$\texttt{Swansea}_{\texttt{Discrete DD}}$: In [220] the authors present a decoupled Delaunay-based parallel mesh generation method. First, a coarse volume mesh is partitioned sequentially based on a greedy method while making sure that no topological issues are created. While still in the master process, the subdomain interfaces are refined and their quality is improved by applying flips and vertex smoothing. Once the pre-processing is complete, the subdomains are sent to the available processes and their volume is refined in parallel introducing points using a Delaunay-based method. The final mesh satisfies the Delaunay property but only within each subdomain. Table 22 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{Swansea}_{\texttt{Discrete DD}}$	Decoupled	None	Coarse	Delaunay	MPI	Discrete D.	No

TABLE 22: Classification of $\texttt{Swansea}_{\texttt{Discrete DD}}$ based on the criteria of this section.

$\texttt{CRTC}_{\texttt{A. Front}}$: Pre-refinement of subdomain interfaces was also utilized by the authors of [132]. For this method, an initial coarse mesh is decomposed by METIS [138] and then the boundaries of each subdomain are extracted. The triangles of the shared boundary are refined to the target using a 2D Delaunay method. Once the pre-processing is complete the subdomains are packed and sent to different processes that utilize an Advancing Front technique to refine the volume of the mesh. To remedy the low quality of elements near the artificial boundaries, the internal vertices of the tetrahedra attached to the boundary are smoothed. Table 23 presents the classification of this method. Similar approaches appear in [238] and [47]. In the former, the authors utilize recursive bisection of the subdomain interface edges that guarantees conforming subdomain interfaces by construction and utilize NetGen [226] as volume mesh generator. The second approach uses a surface simplification technique on the subdomain interfaces to improve the quality of mesh around the separators and a fine-grained advancing front method to refine the subdomain interfaces to their target spacing. A Delaunay-based method is then used in each subdomain for refining the internal of the subdomains.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{A. Front}}$	Decoupled	None	Coarse	A. Front	MPI	Discrete D.	No

TABLE 23: Classification of $\texttt{CRTC}_{\texttt{A. Front}}$ based on the criteria of this section.

$\texttt{CRTC}_{\texttt{VGRID}}$: Another parallel mesh generation method that utilizes an advancing front technique and in particular VGRID [207, 208] is presented in [263]. Starting from a surface mesh refined at the target spacing, this method recursively bisects the subdomains by creating an interface mesh that decouples the two subdomains. The interface is created by instructing VGRID to generate elements along an imaginary plane that splits a subdomain in two. The sizing of the elements along the imaginary plane matches the final spacing and thus the generated elements can remain fixed during the rest of the procedure. The decoupled subdomains are then packed and sent to workers that refine them independently and with no communication. Table 24 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{VGRID}}$	Decoupled	None	Coarse	A. Front	MPI	Discrete D.	No

TABLE 24: Classification of $\texttt{CRTC}_{\texttt{VGRID}}$ based on the criteria of this section.

$\texttt{UChile}_{\texttt{Decoupled}}$: Reference [215] presents an earlier version of $\texttt{UChile}_{\texttt{Tightly}}$. This parallel method works by (over-)decomposing the initial mesh and applying Lepp at each subdomain. Since the shared edges are refined at all neighboring subdomains to the same level governed by the global sizing criteria and not other topological optimization is performed, the interfaces of the subdomains are guaranteed to be conforming. Table 25 presents the classification of this method.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{UChile}_{\texttt{Decoupled}}$	Decoupled	None	Coarse	Lepp	MPI	Discrete D.	No

TABLE 25: Classification of $\texttt{UChile}_{\texttt{Decoupled}}$ based on the criteria of this section.

$\texttt{Swansea}_{\texttt{Continuous DD}}$: In [145] the authors utilize a recursive algorithm that decomposes the input surface triangulation using axis-aligned planes. The intersection of the surface triangulation with each plane creates a piece-wise linear curve which after some pre-processing becomes a simple closed loop. The loop is refined to its terminal spacing with a 2D Delaunay method and is "glued" back to the original surface. Having refined to its terminal size the subdomain interfaces allows to refine the rest of the subdomain with no communication. The final mesh satisfies the Delaunay property but only within each subdomain. Table 26 presents the classification of this method. Same method was presented in [133] by a different group utilizing Triangle [231] and TetGen [235] for the 2D and 3D Delaunay parts of the algorithm respectively.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{Swansea}_{\texttt{Continuous DD}}$	Decoupled	None	Coarse	Delaunay	MPI	Continuous D.	No

TABLE 26: Classification of $\texttt{Swansea}_{\texttt{Continuous DD}}$: based on the criteria of this section.

2.1.5 SUMMARY

The different mesh generation approaches and parallel methods utilized over the last 15 years led to a diverse landscape of parallel mesh generation methods. However, the lack of classification criteria impedes researchers from obtaining a comprehensive view of the field. In this section, we provided an initial attempt defining a set of criteria for classifying parallel mesh generation methods and applied them to methods that appear in [57] and new approaches that have emerged since then. Table 27 summarizes our taxonomy.

Name	Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
$\texttt{CRTC}_{\texttt{PODM 1.0}}$	Tightly	Local	Coarse	Delaunay	MPI	Data	No
$\texttt{CRTC}_{\texttt{PODM 2.0}}$	Tightly	Local	Fine	Delaunay	Pthreads	Data	Yes
$\texttt{CRTC}_{\texttt{CDT3D}}$	Tightly	Local	Fine	Flips	Pthreads	Data	Yes
$\texttt{UBC}_{\texttt{Edge-Face Flip}}$	Tightly	Global	Fine	Flips	OpenMP	Data	Yes
$\texttt{UChile}_{\texttt{Tightly}}$	Tightly	Local	Fine	Lepp	Pthreads	Data	Yes
$\texttt{MIT}_{\texttt{Delaunay}}$	Tightly	Global	Coarse	Delaunay	MPI	Data	No
$\texttt{GMU}_{\texttt{Data Decomp.}}$	Partially	Global	Coarse	A. Front	c$doacross	Data	Yes
$\texttt{CRTC}_{\texttt{PDR 1.0}}$	Partially	Local	Coarse	Delaunay	Pthreads	Data	No
$\texttt{CRTC}_{\texttt{PDR 2.0}}$	Partially	Local	Coarse	Delaunay	Pthreads	Data	Yes
$\texttt{CRTC}_{\texttt{PDR.PODM}}$	Partially	Local	Coarse	Delaunay	MPI+Pthreads	Data	No
$\texttt{UBC}_{\texttt{Delaunay}}$	Partially	Global	Coarse	Delaunay	OpenMP	Data	Yes
$\texttt{UCLouvain}_{\texttt{Gmsh}}$	Partially	Global	Coarse	Delaunay &Flips	OpenMP	Data	No
$\texttt{Imperial}_{\texttt{pragmatic}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Data	No
$\texttt{RPI}_{\texttt{Omega\_h}}$	Partially	Global	Coarse	Edge op. & Flips	MPI+Kokkos	Data	No
$\texttt{LaRC}_{\texttt{refine}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Dis. D.	No
$\texttt{Boeing}_{\texttt{EPIC}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Dis. D.	No
$\texttt{CEMEF}_{\texttt{Edge op.}}$	Partially	Global	Coarse	Edge op. & Flips	MPI	Dis. D.	No
$\texttt{GMU}_{\texttt{Discrete DD}}$	Partially	Global	Coarse	A. Front & Flips	MPI+OpenMP	Dis. D.	No
$\texttt{RPI}_{\texttt{Discrete DD}}$	Partially	Global	Coarse	Templates & Flips	MPI	Dis. D.	No
$\texttt{INRIA}_{\texttt{Feflo.a}}$	Partially	Global	Coarse	Uniq. Cavity	Unix-sockets	Dis. D.	No
$\texttt{RPI}_{\texttt{Continuous DD}}$	Partially	Global	Coarse	A. Front & Templates	MPI	Cont. D.	No
$\texttt{Brown}_{\texttt{Lepp}}$	Loosely	Asynch.	Coarse	Lepp	MPI	Dis. D.	No
$\texttt{INRIA}_{\texttt{Decoupled}}$	Decoupled	None	Coarse	Delaunay	-	Cont. D.	No
$\texttt{Swansea}_{\texttt{Discrete DD}}$	Decoupled	None	Coarse	Delaunay	MPI	Dis. D.	No
$\texttt{CRTC}_{\texttt{A. Front}}$	Decoupled	None	Coarse	A. Front	MPI	Dis. D.	No
$\texttt{CRTC}_{\texttt{VGRID}}$	Decoupled	None	Coarse	A. Front	MPI	Dis. D.	No
$\texttt{UChile}_{\texttt{Decoupled}}$	Decoupled	None	Coarse	Lepp	MPI	Dis. D.	No
$\texttt{Swansea}_{\texttt{Continuous DD}}$	Decoupled	None	Coarse	Delaunay	MPI	Cont. D.	No

TABLE 27: Summary table, Dis. D. : Discrete Domain Decomposition, Cont. D. : Continuous Domain Decomposition.

Using as a guide the *Telescopic Approach* for parallel mesh generation, we built our taxonomy based on its defining characteristic; the coupling between the generated sub-problems. The coupling governs the amount of communication between the subproblems which is one of the main overheads when it comes to parallel applications in general. Synchronization is often another big source of overhead in parallel application and as such is included in our classification. Incorporating granularity in our criteria allows to distinguish methods that exploit parallelism at the element versus the subdomain level. Also, it allows to describe hybrid methods that operate at multiple granularities at the same time. The meshing method and Programming model are self-explanatory and depend on the implementation of each method. Finally, we used the decomposition method and the characterization of whether a method is progressive or not. The former is an essential part of every parallel algorithm that defines how the work units are created while, the latter indicates whether the method redistributes work implicitly by continuously moving work units from one process to the other or not.

As noted in the introduction, the goal of this dissertation is to create the meshing building block for the *Telescopic Approach*. This means that by design we are focusing on a Tightly-coupled method designed to explore concurrency at the CPU level within the limits of shared memory. Moreover, we will be utilizing the speculative approach which has been found in the past [95, 187] to perform very well at this layer of the memory hierarchy. The speculative approach will be applied at the element and vertex level, thus making our method fine-grained. Our programming model includes both manual methods for managing resources such as Pthreads, but also supports tasking environments. Details about the use of tasking environments appear in Section 5. Finally, we will adopt the Data Decomposition scheme of the previous isotropic implementation [82, 85] that enables the method to be progressive and extend it based on the needs of new parallel operations as described in Section 3. Table 28 summarizes the characteristics of our approach based on the criteria of this section. To the best of our knowledge the method presented in this work is the first tightly-coupled speculative fine-grained method for anisotropic 3D mesh adaptation.

Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
Tightly	Local	Fine	Edge op. & Flips	Pthreads/task-based	Data	Yes

TABLE 28: Classification of our method.

2.2 METRIC SPACES IN THE CONTEXT OF MESH ADAPTATION

One of the important characteristics when generating a mesh is to keep its size low since it relates directly to the computational cost of the solver. Orthogonal to the low computational cost is the requirement of capturing both large and small features of the solution. Historically, the mesh would be further refined on regions of interest based on *a-priori* knowledge of the problem [189]. However, as the demand for more complex simulations arises, identifying regions of interest is not only difficult due to the complex configurations and flows but in some cases infeasible due to the high cost of human intervention.

Early attempts to control the mesh density at regions of interest relied on re-distributing the points of a structured grid [246]. Mesh multiplication techniques approach the problem by splitting elements into self-similar elements [11]. In [25] the authors create overset meshes of different resolutions driven by estimates of the local truncation error. The need for capturing *anisotropic* features, i.e., features with different variations in each direction, pushed the research towards unstructured meshes. In [204] the authors drive their advancing front point creation method by producing a point spacing governed by error-estimators. The first use of a two-dimensional Delaunay method to create anisotropic elements appears in [177]. The author captures the anisotropy of a two-dimensional mesh using a three-dimensional control surface that encodes local size and orientation via its curvature. The Delaunay cavity evaluation is then modified to correspond to isotropic evaluations on the high-order surface. Creating anisotropic meshes with the Delaunay method was revisited in [109], but this time the error estimator was provided in the form of a symmetric positive definite matrix. The use of a symmetric positive definite matrix allows inducing a metric that can be used to evaluate geometrical quantities such as the length while taking into consideration the local orientation and size requirements. They also introduced the idea of the *unit mesh*. The crux of this approach is to transform the problem of creating the *best mesh* by creating one mesh with all its edge lengths equal to 1 when measured by the new metric. The idea of metric-based mesh generation through unit meshes has been extended and improved over the years. Its effectiveness has been demonstrated in a wide variety of mesh adaptation problems [5, 161, 167]. Moreover, theoretical work [166] showed that the notion of the unit mesh, when coupled with the appropriate metric, can control the interpolation error obtained on a metric-adapted mesh.

Apart from the unit mesh approach, there are several other strategies for generating anisotropic meshes based on frame fields [196], convex functions [98], anisotropic Voronoi methods based on geodesic distances [144, 147], Delaunay methods [32], and particle-based approaches [265]. These approaches are out of the scope of this work since many of them focus on two-dimensional or surface meshing only, and they target mainly graphics applications.

Another approach for generating anisotropic meshes is through the use of higher dimensional spaces [41, 71, 149, 266]. These approaches are reminisced of the Delaunay-based approach discussed above [177]. The basis of these methods is a theorem [186] that proves the existence of a map between Riemannian spaces and higher dimensional Euclidean spaces. In practice, this approach transforms the problem of generating an anisotropic mesh in $\mathbb{R}^3$ to that of generating an isotropic mesh in $\mathbb{R}^n$ where $n > 3$. Some approaches [41, 149, 266] generate the mesh directly in $\mathbb{R}^n$ through the use of n-dimensional Voronoi diagrams. Others [71] perform only predicate and length computations in $\mathbb{R}^n$ while generating a mesh in $\mathbb{R}^3$. The latter allows to re-use all the widely-used meshing operations such as edge-face flips and vertex smoothing in higher dimensional Riemannian spaces.

In this work, we adopt the unit-mesh approach for creating metric-adapted meshes. The choice is based on the fact that it is more established and well-tested on a wide variety of CFD applications [6]. In contrast, the higher-dimensional embedding method is a relatively new approach used mainly in graphics applications. Moreover, a general approach for creating the required mapping from an arbitrary Riemannian metric to a self-intersection-free high dimensional Euclidean space only recently appeared in the literature [266].

The rest of the chapter presents the most important definitions and properties of the Euclidean and Riemannian metric spaces, which provide a theoretical framework for using metric tensors and metric-based geometrical evaluations in the context of mesh adaptation. The material is drawn in part from [2, 20, 80, 161, 166, 171, 200] and summarized here for completeness. Section 2.2.1 introduces the notion of the metric tensor and describes spaces equipped with a constant metric tensor. Also, it addresses aspects related to how geometrical evaluation such as lengths and angles can be adapted to take into account the metric tensor information. In Section 2.2.2, we introduce spaces where the metric tensor varies from point to point. This construction is closer to the metric-based adaptation problem where the sizing and orientation requirements vary from point to point. Finally, in Section 2.2.3 we present a few metric operations that are used throughout this work.

2.2.1 EUCLIDEAN METRIC SPACES

Informally, the Euclidean metric space can be thought as a transformed space near the vicinity of a point. In the context of mesh adaptation, the mesh in the vicinity of the point is transformed through affine transformations[8] in order to create an isotropic mesh in the transformed space. Choosing the transformations appropriately, results in a mesh that exhibits the desired sizing and orientation in the physical space. One of the characteristics of Euclidean spaces is that it enables to alter the way we measure geometrical quantities by the introduction of a new metric which is built on top of positive definite matrices.

Definition 1 (Positive Definite Matrix)**.** *A real symmetric* 3×3 *matrix* M *is called positive-definite iff* $\forall u \in \mathbb{R}^3$, $u \neq 0 \implies u^T M u > 0$.

Remark 1. For a positive-definite matrix $\mathcal{M}$, the bi-linear function $\langle u, v \rangle_{\mathcal{M}} := u^T \mathcal{M} v$ defines an inner product in $\mathbb{R}^3$.

Remark 2. It can be shown that every inner product on a real vector space can induce a vector space norm by the map $\|u\|_{\mathcal{M}} = \sqrt{\langle u, u \rangle_{\mathcal{M}}}$ and subsequently a metric via the formula $d_{\mathcal{M}}(x, y) = \|x - y\|_{\mathcal{M}}$. Due to this property, the terms *metric tensor* and *metric* are used interchangeably in the metric-based adaptation literature and throughout this thesis.

The pair $(\mathbb{R}^3, \mathcal{M}) := (\mathbb{R}^3, d_{\mathcal{M}})$ constitutes therefore a metric space. For the rest of the chapter, it will be called a **Euclidean Metric Space** or simply a **Euclidean Space**. In a metric space, notions such as length, angle, area, and volume, which are of particular interest in mesh generation, can be defined by substituting the identity metric with the one induced by the positive-definite matrix $\mathcal{M}$:

$$\text{length of a segment} \qquad \ell_{\mathcal{M}}(x, y) = d_{\mathcal{M}}(x, y) \tag{1}$$

$$\text{angle between non-zero vectors} \qquad \cos(\theta_{\mathcal{M}}) = \frac{\langle u, v \rangle_{\mathcal{M}}}{\|u\|_{\mathcal{M}} \|v\|_{\mathcal{M}}}, \quad \theta_{\mathcal{M}} \in [0, \pi] \tag{2}$$

Notice that by using $\mathcal{M} = \mathcal{I}$, where $\mathcal{I}$ is the identity matrix, we recover the familiar formulas. Next, we introduce a general transformation map from the physical space $(\mathbb{R}^3, \mathcal{I})$ to the Euclidean space $(\mathbb{R}^3, \mathcal{M})$. The first step to derive this map is given by the spectral theorem of linear algebra [241]:

[8]An affine transformation between two vector spaces can be defined as a linear transformation plus a shift [241]: $T(\overrightarrow{v}) = A\overrightarrow{v} + \overrightarrow{v_0}$. A linear transformation is a transformation that satisfies the linear property: $T(c\overrightarrow{v} + d\overrightarrow{w}) = cT(\overrightarrow{v}) + dT(\overrightarrow{w})$. Examples of affine transformations are rotation, translation, scaling, reflection, etc.

Theorem 1 (Spectral Decomposition). *If $M \in \mathbb{R}^{n\times n}$ is a symmetric matrix then it can be factorized as: $M = PDP^T$ where P is an orthogonal matrix whose columns are eigenvectors of M and $D := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ the diagonal matrix of the eigenvalues of M.*

This theorem enables also the representation of the metric tensor as an ellipsoid centered at the point where the metric tensor is defined. This representation stems from the fact that a unit ball centered at a point c can be defined as $B(c) = \{x \in \mathbb{R}^3 : (x-c)^T(x-c) \leq 1\} = \{x \in \mathbb{R}^3 : \langle x-c, x-c\rangle \leq 1\}$. Substituting the natural inner product $\langle\cdot,\cdot\rangle$ with $\langle\cdot,\cdot\rangle_{\mathcal{M}}$ one gets a unit ball in $(\mathbb{R}^3, \mathcal{M})$ and it can be shown that it corresponds to an ellipsoid back in the physical space:

$$\begin{aligned}
B_{\mathcal{M}}(c) &= \{x \in \mathbb{R}^3 : \langle x-c, x-c\rangle_{\mathcal{M}} \leq 1\} = \{x \in \mathbb{R}^3 : (x-c)^T\mathcal{M}(x-c) \leq 1\} \\
&= \{x' \in \mathbb{R}^3 : (Px' - Pc')^T PDP^T(Px' - Pc') \leq 1\} \quad \text{where} \quad x' = P^T x \qquad (3)\\
&= \{x' \in \mathbb{R}^3 : (x'-c')^T D(x'-c') \leq 1\} \quad \text{since} \quad PP^T = \mathcal{I}_3 \\
&= \left\{x' \in \mathbb{R}^3 : \left(\frac{x'_1 - c'_1}{1/\lambda_1}\right)^2 + \left(\frac{x'_2 - c'_2}{1/\lambda_2}\right)^2 + \left(\frac{x'_3 - c'_3}{1/\lambda_3}\right)^2 \leq 1\right\} \qquad (4)
\end{aligned}$$

In (3) we transform the space by applying a rotation based on matrix P. The rotation results in aligning the coordinate axis with the eigenvectors of $\mathcal{M}$. Thus, equation (4) defines an ellipsoid centered at c with semi-axes aligned to the eigenvectors of $\mathcal{M}$ and their lengths given by $\frac{1}{\sqrt{\lambda_i}}$, see also Figure 3.

As noted in [166] the spectral decomposition enables the definition of the map

$$\begin{aligned}
D^{\frac{1}{2}}P^T : \quad & (\mathbb{R}^3, \mathcal{I}) \to (\mathbb{R}^3, \mathcal{M}) \\
& x \mapsto D^{\frac{1}{2}}P^T x, \qquad \text{where } D^{\frac{1}{2}} := \operatorname{diag}\left(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3}\right)
\end{aligned}$$

This function maps a vector from the physical space to the Euclidean space induced by $\mathcal{M}$. Using the above map in conjunction with common geometric formulas, one can transform results from the physical space to the Euclidean space. For example, notice that we can recover the definition of the inner product in $\mathcal{M}$ since,

$$\langle u, v\rangle_{\mathcal{M}} = \langle D^{\frac{1}{2}}P^T u, D^{\frac{1}{2}}P^T v\rangle = \left(D^{\frac{1}{2}}P^T u\right)^T \cdot D^{\frac{1}{2}}P^T v = u^T\mathcal{M}v$$

and thus recover equations (1) and (2). Also, using the algebraic properties of the mixed

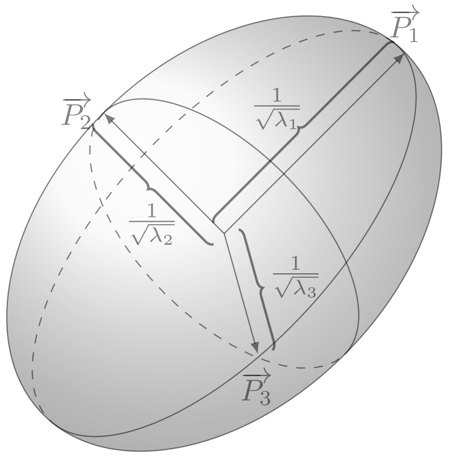

$\overrightarrow{P_1}$
$\overrightarrow{P_2}$
$\overrightarrow{P_3}$
$\frac{1}{\sqrt{\lambda_1}}$
$\frac{1}{\sqrt{\lambda_2}}$
$\frac{1}{\sqrt{\lambda_3}}$

Fig. 3: Visualization of a metric tensor $\mathcal{M} = P\,\mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)P^T$ as an ellipsoid. The orthonormal eigenvectors $\vec{P}_1, \vec{P}_2, \vec{P}_3$ define the principal directions of the ellipsoid while the quantities $\frac{1}{\sqrt{\lambda_i}}$ the length of the semi-axes.

product one can evaluate the volume of a parallelepiped K in the Euclidean space ($\mathbb{R}^3$, $\mathcal{M}$) as a scalar multiple of its volume in ($\mathbb{R}^3$, $\mathcal{I}$):

$$\begin{aligned}
|K|_{\mathcal{M}} &= \langle x, (y \times_{\mathcal{M}} z)\rangle_{\mathcal{M}} = \left\langle \left(D^{\frac{1}{2}}P^T\right)x, \left(\left(D^{\frac{1}{2}}P^T\right)y \times \left(D^{\frac{1}{2}}P^T\right)z\right)\right\rangle \\
&= \det\left(\left(D^{\frac{1}{2}}P^T\right)x, \left(D^{\frac{1}{2}}P^T\right)y, \left(D^{\frac{1}{2}}P^T\right)z\right) \\
&= \det\left(\left(D^{\frac{1}{2}}P^T\right)(x, y, z)\right) = \det\left(D^{\frac{1}{2}}\right)\cdot\det\left(P^T\right)\cdot\det(x, y, z) \\
&= \sqrt{\det(\mathcal{M})}\cdot 1\cdot\langle x, (y \times z)\rangle = \sqrt{\det(\mathcal{M})}|K|_{\mathcal{I}} \qquad (5)
\end{aligned}$$

where $|K|_{\mathcal{I}}$ is the volume of the parallelepiped in the physical space. This formula can then be used to measure the volume of a tetrahedron in the Euclidean space since its volume is given by $\frac{1}{6}\langle x, (y\times z)\rangle$ where x, y, z are the vectors emanating from a vertex of the tetrahedron.

2.2.2 RIEMANNIAN METRIC SPACES

In a Euclidean space, the metric tensor is a constant matrix; if one allows the metric tensor to vary smoothly over the computational domain, then the **Riemannian Metric space** or simply **Riemannian space** is defined. Even though, in a Riemannian space there is no global definition of the inner product, it is still a useful construct for mesh adaptation

purposes, since in most applications, the metric varies within the domain. In practice, the metric tensor is defined only at the vertices of a mesh so, to be able to calculate the above geometric quantities at any point of the domain we need to approximate the value of the metric tensor field throughout the computational domain. Following the widely used approach, we utilize the Log-Euclidean interpolation framework introduced in [14]. One of the main advantages of this scheme over the ones used previously in the literature is that the interpolation operator is commutative, thus allowing the combination of metric tensors in any order. Moreover, in [181] the Log-Euclidean framework was found to be superior in comparison to the power average and the simultaneous matrix reduction methods [33] in various numerical evaluations. In the Log-Euclidean framework the value of the metric at an arbitrary point of the domain is interpolated as follows:
Given a set of vertices x_i , $i = 1, \dots, k$ and $\mathcal{M}_i = \mathcal{M}(x_i)$ their corresponding metrics, then for a point x of the domain with barycentric coordinates a_i

$$x = \sum_{i=1}^{k} a_i \cdot x_i, \quad \text{with} \quad \sum_{i=1}^{k} a_i = 1,$$

the interpolated metric is defined by

$$\mathcal{M}(x) = \mathcal{M}_{mean}(x) = \exp\left(\sum_{i=1}^{k} a_i \ln \mathcal{M}_i\right) \tag{6}$$

Notice that since $\mathcal{M}_i$ are positive definite, they have positive eigenvalues and therefore the exponential and logarithm of the metrics are well defined and given by

$$\ln(\mathcal{M}) := P \ln(D) P^T \quad \text{and} \quad \exp(\mathcal{M}) := P \exp(D) P^T,$$

where $\ln(D) := \operatorname{diag}(\ln(d_1), \ln(d_2), \ln(d_3))$ and $\exp(D) := \operatorname{diag}(\exp(d_1), \exp(d_2), \exp(d_3))$.

The length of a segment in an adapted mesh plays critical role in the quality of the mesh and it is one of the metrics used in this work to quantify the quality of the generated mesh. In differential geometry the length of a vector $\mathbf{u} = xy$ is computed using the integral [205]:

$$\ell_{\mathcal{M}}(xy) = \int_0^1 \|\gamma'(t)\|_{\mathcal{M}} \, \mathrm{dt} \quad \text{where } \gamma(t) = x + t \cdot \mathbf{u} \tag{7}$$

However, this evaluation is expensive since $\mathcal{M}$ varies along the edge. In practice, we use the variational law presented in [2]:

Definition 2. *Let $e = p_1p_2$ be an edge of the mesh and $\mathcal{M}_1 := \mathcal{M}(p_1)$, $\mathcal{M}_2 := \mathcal{M}(p_2)$ the corresponding metric tensor on the endpoints of the edge. Without loss of generality, assume that $\ell_{\mathcal{M}_1}(e) > \ell_{\mathcal{M}_2}(e)$, the length of e can then be approximated by:*

$$\ell_{\mathcal{M}}(e) = \ell_{\mathcal{M}_1}(e)\frac{a-1}{a\ln(a)}, \qquad \textit{where } a = \frac{\ell_{\mathcal{M}_1}(e)}{\ell_{\mathcal{M}_2}(e)}. \tag{8}$$

Formally, in a Riemannian space the volume is evaluated using a continuous version of formula (5). However, in order to avoid the expensive calculations, we interpolate the metric on the centroid of the element and then use the Euclidean volume formula (5).

Finally, the measures used to report mesh quality also need to be adapted. In this work, we adopted the mesh quality measures used by the Unstructured Grid Adaptation Working Group[9]. The edge lengths are evaluated using:

$$\begin{aligned} L_e &= \begin{cases} \frac{L_a - L_b}{\log(L_a/L_b)} & |L_a - L_b| > 0.001 \\ \frac{L_a + L_b}{2} & \textit{otherwise} \end{cases}, \\ L_a &= (v_e^T \mathcal{M}(a) v_e)^{\frac{1}{2}},\ L_b = (v_e^T \mathcal{M}(b) v_e)^{\frac{1}{2}} \end{aligned} \tag{9}$$

where v_e is the vector along edge ab. Since the goal is to create a unit mesh, edges of length below or above 1 are considered sub-optimal. The Mean Ratio [154] shape measure is also approximated in the discrete metric,

$$Q_k = \frac{36}{3^{1/3}} \frac{\left(|k|\sqrt{\det(\mathcal{M}_{\text{mean}})}\right)^{\frac{2}{3}}}{\sum_{e \in L} v_e^T \mathcal{M}_{\text{mean}} v_e}, \tag{10}$$

where v_e is the vector along the edge e of element k, $|k|$ the isotropic volume of the element, $\mathcal{M}_{\text{mean}}$ is the interpolated metric tensor evaluated at the centroid of element k, and L the set of all edges of k. The measure is normalized by the volume of an equilateral element and as such its range is $[0, 1]$ with 1 being the optimal quality for an element.

In this section we presented the Riemannian space merely as a way to compute lengths within the metric-based adaptation context. However, its usefulness is far beyond this usage. It has been shown that using the notion of Riemannian spaces in metric-based adaptation, one can move from the ill-posed problem of generating the "best" mesh that minimizes the linear interpolation error of a function evaluated on a mesh, to the well-posed problem of evaluating a continuous metric field that minimizes the interpolation error [166].

[9]`https://ugawg.github.io/` (Accessed 2021-05-31).

2.2.3 METRIC OPERATIONS

In the last section of this chapter, we describe a few important operations on metric tensors. In particular, we present a method for intersecting different metric tensors at a given point. This operation will become especially useful in later chapters where we combine metrics from different sources. We also present the notion of *metric complexity* which provides a way to control the level of refinement of a metric-adapted mesh globally. Controlling the metric complexity enables the creation of successively adapted meshes that will enable us in Section 4.2 to create an error-based adaptive pipeline. Finally, we describe briefly the construction of the *multiscale metric*, the method adopted in this work for constructing metric fields.

2.2.3.1 Metric Intersection

During adaptation, we may have more than one metric tensor defined at a vertex. An example is a metric tensor based on the error coming from the CFD solver and the second being constructed based on the curvature of the geometry. Moreover, the user may need to impose additional constraints such as a global maximum edge size or, in general, any other kind of metric limiting [181]. One approach to combine multiple metrics is metric intersection.

If we visualize the metric tensors as ellipsoids (see Figure 3) then the metric intersection procedure can be seen as evaluating the largest ellipsoid included in the intersection of the other two (see Figure 4). The metric intersection can be evaluated through simultaneous diagonalization of the two metrics [33]. Formally, the simultaneous diagonalization of two positive definite matrices A and B produces a basis P that can be used to diagonalize both matrices simultaneously, i.e., $A = PD_AP^T$ and $B = PD_BP^T$ [125]. Using these decompositions one can define a new positive definite matrix $C = P\,\mathrm{diag}(\max(e_i, d_i))P^T$, where e_i, d_i the diagonal elements of D_A and D_B respectively. C corresponds to the largest ellipsoid included in the intersection of the ellipsoids of A and B [1]. In practice, we adopt the approach presented in [20] which we describe here for completeness.

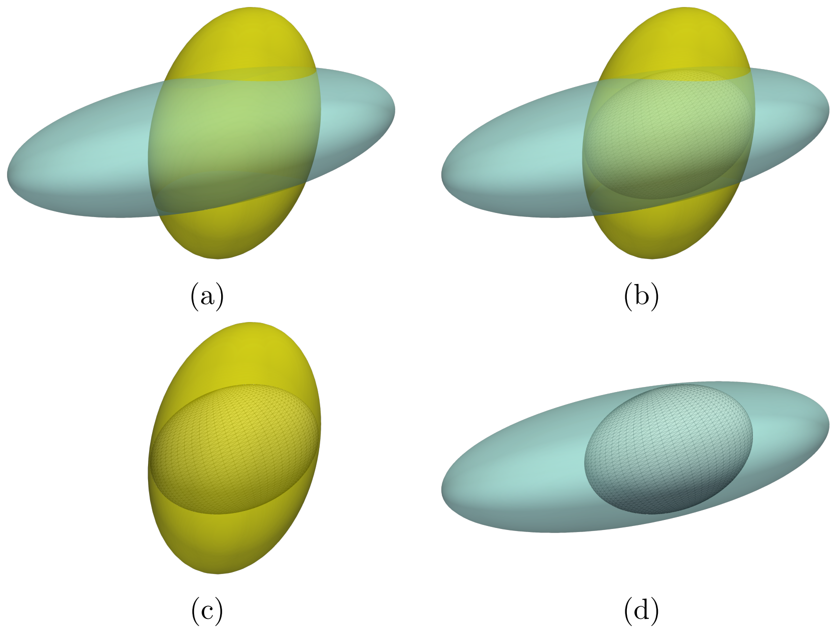


Fig. 4: Metric Intersection. (a) Two metric tensors to be intersected. (b) The metric corresponding to the intersection result. (c) The intersection result and the first metric. (d) The intersection result and the second metric.

Given two metric tensors $\mathcal{M}_1, \mathcal{M}_2$, we first transform both matrices by multiplying them by $(\mathcal{M}_1^{-1/2})$[10] :

$$\overline{\mathcal{M}_1} = \mathcal{M}_1^{-1/2}\mathcal{M}_1\mathcal{M}_1^{-1/2} = \mathcal{I}$$
$$\overline{\mathcal{M}_2} = \mathcal{M}_1^{-1/2}\mathcal{M}_2\mathcal{M}_1^{-1/2}$$

Then, we diagonalize $\overline{\mathcal{M}_2}$, $\overline{\mathcal{M}_2} = PDP^T$. P can be used to diagonalize $\overline{\mathcal{M}_1}$ trivially: $\overline{\mathcal{M}_1} = P\mathcal{I}P^T$. The new metric can then be calculated as:

$$\overline{\mathcal{M}_{1\cap 2}} = P\,\mathrm{diag}(\max(d_i, 1))P^T$$

Finally, we apply the inverse transformation to return to physical space:

$$\mathcal{M}_{1\cap 2} = \mathcal{M}_1^{1/2}\overline{\mathcal{M}_{1\cap 2}}\mathcal{M}_1^{1/2}$$

[10]Since $\mathcal{M}$ is positive definite, $\mathcal{M}_1^{-1/2}$ can be defined by means of spectral decomposition: $\mathcal{M}_1^{-1/2} := P\,\mathrm{diag}(1/\sqrt{\lambda_1}, 1/\sqrt{\lambda_2}, 1/\sqrt{\lambda_3})P^T$.

Geometrically, this process corresponds to transforming both metric tensors using $\mathcal{M}_1^{-1/2}$ which reduces $\mathcal{M}_1$ to a unit circle thus making the rest of the process trivial. For more algorithms around intersections of ellipses and metric tensors one can refer to [80].

2.2.3.2 Metric Complexity

Traditionally, mesh convergence studies use a parameter h representing the edge length of a uniform mesh in order to create a series of successively refined meshes that reduce the approximation error [216]. In the context of metric-based adaptation, a single parameter based on edge length is not sufficient to capture the sizing requirements since, it depends on the direction of the element. To resolve this issue, the notion of *metric complexity* of a continuous metric is introduced [166]. To demonstrate its practicality, we first rearrange the spectral decomposition of the metric tensor at a point x, $\mathcal{M}(x) = PDP^T$ as follows:

$$\mathcal{M}(x) = d^{\frac{2}{3}}(x)P(x)\begin{pmatrix} r_1^{-2}(x) & 0 & 0 \\ 0 & r_2^{-2}(x) & 0 \\ 0 & 0 & r_3^{-2}(x) \end{pmatrix} P^T(x)$$

where,

$$r_i = \frac{h_i^3}{h_1h_2h_3} \text{ and } d = \frac{1}{h_1h_2h_3} = \sqrt{\lambda_1\lambda_2\lambda_3}$$

In this representation we introduced d the *density* of the metric field. Increasing or decreasing d does not affect the orientation or the anisotropic stretching ratio[11]. Based on d we define the *complexity* $\mathcal{C}$ of $\mathcal{M}$:

$$C(\mathcal{M}) = \int_\Omega d(x)\,\mathrm{dx} = \int_\Omega \sqrt{\det(\mathcal{M}(x))}\,\mathrm{dx} \tag{11}$$

In practice, the metric field is known only on mesh vertices so, one can evaluate the metric complexity of a discrete metric with:

$$C(\mathcal{M}) = \sum_{i=1}^{N} \sqrt{\det(\mathcal{M}_i)}V_i \tag{12}$$

where V_i is the volume of the Voronoi dual surrounding each node. The complexity allows to quantify the global level of accuracy. In fact, it can be shown that it is equal to approximately

[11]One can show that $d^{\frac{2}{3}}r_i^{-2} = \lambda_i$ [166] so, a scalar multiple of d results into scaling all eigenvalues by the same factor.

half the number of vertices of a unit mesh covering the domain [166]. It also allows to define a series of successively "denser" meshes that can be used in place of the traditional h-refined series of meshes. The complexity of a metric can be scaled to a target value using:

$$\mathcal{M}_{target} = \left(\frac{\mathcal{C}_{target}}{\mathcal{C}(\mathcal{M})}\right)^{2/d} \mathcal{M}, \tag{13}$$

where d is the dimension of the problem which is 3 throughout this work. Since the complexity is defined based on the density, increasing (resp. decreasing) it, will create a uniformly refined (resp. coarsened) mesh with the same orientation and element shape locally and the same distribution of elements across the domain.

2.2.3.3 Metric Construction - Multiscale Metric

Having established a way to control the size and orientation of the mesh locally through metric tensors, it remains to specify how to construct metric tensors at each point of the mesh. There are several different methods to construct a metric tensor, among others, the multiscale metric [161], output-based metrics [90] and optimization schemes [259]. For this work, the multiscale error metric is selected due to its wide use, simplicity and availability in open-source projects such as the `refine` adaptation mechanics [198]. This approach has been confirmed both theoretically [166] and experimentally [167] in a number of applications. Moreover, it can be shown that the multiscale metric controls the L^p-norm interpolation error of the scalar field that is constructed from.

The multiscale metric is evaluated as follows [5, 161, 200]: Given a mesh along with the discrete values of a scalar field evaluated at each mesh vertex, the Hessian is reconstructed at each vertex. The Hessian H is then factorized into eigenvectors and eigenvalues by means of spectral decomposition: $H = PDP^T$. The factors are recombined into a metric tensor by taking the absolute value of the eigenvalues in order to guarantee positive definiteness $\mathcal{M}_1 = P|D|P^T$ [109]. The metric tensor is then scaled locally based on its determinant $\mathcal{M}_2 = \det(\mathcal{M}_1)^{-1/(2p+d)}\mathcal{M}_1$, where p is the power of the norm used to control the interpolation error which is L^2 and d is the dimension of the problem which is 3 throughout this work. Finally, the metric is scaled globally to the target complexity using equation (13):

$$\mathcal{M} = \left(\frac{\mathcal{C}_{target}}{\mathcal{C}(\mathcal{M}_2)}\right)^{2/d} \mathcal{M}_2. \tag{14}$$

The final step in constructing a metric field is the application of gradation. The gradation aims at putting constraints over the largest size deduced from a metric and smoothing the metric transition between edge-connected vertices. In this work we adopt the *mixed-space-gradation* method presented in [2] and utilize its implementation in `refine` [198].

CHAPTER 3

PARALLEL METRIC-BASED ADAPTATION

Our approach to parallel mesh adaptation builds on top of the *Telescopic Approach* (see Figure 1 in page 7). The *Telescopic Approach* lays down a design that allows to exploit the concurrency that exists at multiple levels in parallel and adaptive simulations. The design spans across the multiple memory hierarchies of an exascale machine and maps different algorithmic layers to the appropriate level of memory based on the intensity of communication between the meshing kernels. The lowest, closest to the hardware layer, that can sustain high volume of communication at low cost, is the Parallel Optimistic layer which is designed to explore concurrency at the CPU level within the memory limits of a shared memory node, using speculative/optimistic execution. In [85] we presented a fine-grained speculative scheme for local reconnection for generating isotropic meshes. This chapter describes implementation details in:

- Extending existing mesh operations by (i) making them metric-aware, and (ii) making them applicable for boundary adaptation (Section 3.2).
- Introducing initial support for geometrical models (Section 3.3).
- Applying the speculative fine-grained scheme beyond the local reconnection operation (Section 3.4).

The method presented in this section belongs to the wider class of tightly coupled methods. Tightly Coupled methods are described in detail in Section 2.1.2. To the best of our knowledge the method presented in this work is the first tightly-coupled speculative fine-grain method for parallel 3D anisotropic mesh adaptation. There are however a number of mesh adaptation methods that utilize data or domain decomposition. The most pertinent to this work are *refine*, *EPIC*, *Feflo.a*, *Pragmatic* and *Omega_h*. Their parallel characteristics and mesh adaptation features are already discussed in Section 2.1.

3.1 ISOTROPIC MESH GENERATION

The metric-based approach of this thesis builds on top of *CDT3D* [82]. *CDT3D* is a mesh generation toolkit developed at the CRTC lab[12] of Old Dominion University. Its main characteristics are stability, end-user-productivity and modular design. End-user-productivity for isotropic mesh generation was demonstrated in [85] where *CDT3D* was compared against a state-of-the-art advancing front mesh generator. Its modular design allowed in [267] the addition of refinement zones for the isotropic method than enable its use on Large Eddy Simulations (see Figure 5).

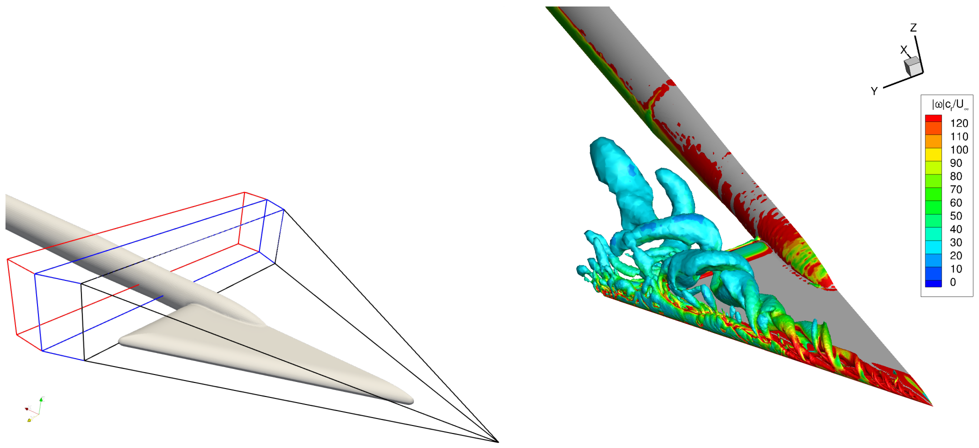


Fig. 5: Refinement Zones for Large Eddy Simulation over Delta wing. Left: Refinement zones used in [267]. Each polyhedron defines different sizing constrains. Right: Simulation results.

Figure 6 offers a high-level description of the isotropic mesh generation pipeline. The isotropic *CDT3D* accepts a triangulated surface mesh as an input which it kept constrained throughout the meshing process. In the first stage, the triangulated surface mesh is recovered using methods based on Delaunay tetrahedralization [236]. In practice, recovering the boundary is accomplished by creating a boundary conforming tetrahedral mesh, i.e., all the

[12]`https://crtc.cs.odu.edu` (Accessed 2021-31-15).

faces of the input surface appear as a face of some tetrahedron. The robustness of boundary recovery implementation has been evaluated extensively, and it was found in-par with state-of-the-art boundary recovery methods [82]. Mesh refinement introduces points iteratively into the mesh using advancing front point creation and direct insertion. The advancing front method offers great control on point density and especially on the growth of the spacing between the generated points. The spacing of the points is initialized by the spacing (i.e., edge lengths) of the surface mesh. The growth of the size of the tetrahedra follows an exponential distribution with parameters controlled by the user. After each point creation iteration, the connectivity of the mesh is optimized in parallel using a fine-grained topological scheme for local reconnection [85], optimizing a combined criterion of the Delaunay in-sphere criterion and the maximization of the minimum edge-weight [22] used. The combined criterion is evaluated for every set of two/three neighboring tetrahedra that a face/edge flip can be applied. The configuration that improves the combined criterion is then used. In the last stage, the mesh quality is improved using a combination of mesh smoothing, parallel local reconnection, and heuristics to target the improvement of low quality elements. Extensive evaluation of the effectiveness of the quality improvement step against state-of-the-art local reconnection methods can be found in [85].

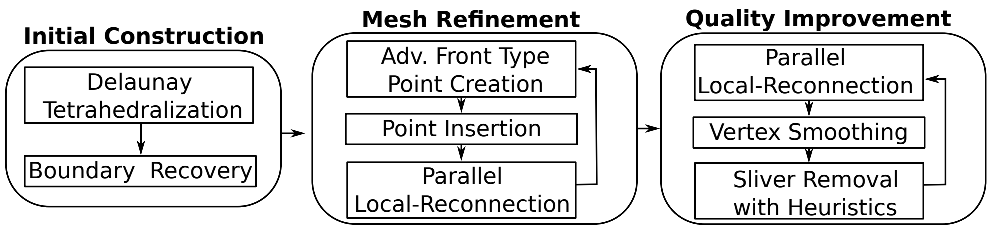

Initial Construction
Delaunay Tetrahedralization
Boundary Recovery
Mesh Refinement
Adv. Front Type Point Creation
Point Insertion
Parallel Local-Reconnection
Quality Improvement
Parallel Local-Reconnection
Vertex Smoothing
Sliver Removal with Heuristics

Fig. 6: High-level pipeline of isotropic *CDT3D* as presented in [85].

3.2 METRIC-BASED ADAPTATION WITHIN THE *CDT3D* LIBRARY

As with any software project, the goal is to maximize code reuse while introducing additional features, which in this case, is the metric-based adaptivity. To achieve this, one can decompose a mesh operation into *topological* and *geometrical* steps. Topological steps access and modify only the connectivity information (a 2-3 flip, for example, see Figure 9a) and as such, there is no need for modifications for metric adaptation. On the other hand, geometrical steps, such as evaluating a predicate that decides whether a flip should be performed, will need to incorporate the metric information. The rest of this section presents the most significant modifications required to enable metric adaptation in *CDT3D*. Moreover, *CDT3D* is extended in order to handle CAD-based information. Figure 7 depicts the metric-adaptive pipeline built and evaluated throughout this work. Its components are described in the following sections.

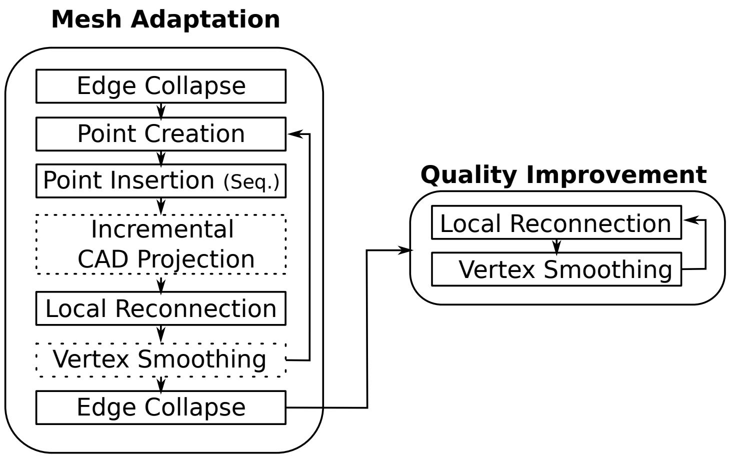


Fig. 7: Pipeline of the presented approach. Dotted modules are utilized only when CAD data are available.

3.2.1 POINT CREATION STRATEGIES FOR ANISOTROPIC MESH ADAPTATION

The isotropic version of *CDT3D* generates points using an advancing front scheme. In particular, a collection of faces between active[13] and inactive tetrahedra called the *front* is first identified and then points are created in a direction perpendicular to each front face [82]. Utilizing metric-based distance evaluations and taking into account the orientation matrix of the metric can be used to create a metric-adaptive advancing front scheme. This scheme brings all the benefits of an advancing front method as well as the ability to create metric-aligned meshes as presented in [160, 170, 172].

In an earlier attempt, we extended *CDT3D* with a metric-based advancing front scheme [252]. In particular, we coupled *CDT3D* with the open-source MMGS software [70, 183] which offers 3D surface metric-based adaptation capabilities. The pipeline consisted of extracting the surface of the input mesh and adapting it using MMGS, then creating a boundary conforming mesh employing boundary recovery, and finally generating a mesh using the input mesh as a background mesh that provides the metric field values. This approach, however, has several issues. First, the sequential nature of the surface adaptation operation quickly becomes a bottleneck. Moreover, this approach requires to recover the boundary of an anisotropic mesh each time, and although *CDT3D*'s surface recovery algorithm has shown to be robust even in low quality triangulations [82], it is expected to have issues in highly stretched anisotropic meshes. Finally, disregarding the previous mesh and regenerating one from scratch at each adaptive iteration requires more time especially, when dealing with larger meshes and targeting a constant metric complexity.

To overcome these issues, we introduced a new centroid-based point creation method. The centroid-based method will check the edge lengths of an active element in the metric space, and if any of them does not satisfy the spacing requirements, it will produce its centroid as a candidate point. If the tetrahedron has a boundary face, or if any of its edges is a ridge (i.e., it is between two different surface markers) encroachment rules similar to those used in Constrained Delaunay refinement [233] are utilized. The candidate point will be checked for encroachment (in the metric space) against the boundary face, and if encroachment occurs, the candidate is rejected and the centroid of the boundary face becomes the new candidate. The same procedure is applied for the new point which is checked for encroachment against any ridge edges, see Figure 8 for more details. Once a

[13]A tetrahedron is considered active, if it does not satisfy the spacing or quality requirements.

```
Function GenerateCandidatePoints(t)
  Input: An active tetrahedron $t$
  Result: Candidate point(s) for element $t$
  $candidatePoints = \{\}$
  $c \leftarrow centroid(t)$
  for boundary face $f$ of $t$ do
      if $c$ encroaches upon $f$ then
          if all edges of $f$ are long then
              $p \leftarrow centroid(f)$
              for ridge edge $e$ of $f$ do
                  $candidatePoints.append(midpoint(e))$
              end for
              if candidatePoints is empty then // no ridge was found
                  $candidatePoints.append(c)$
              end if
          else
              for long edge $e$ of $f$ do
                  $candidatePoints.append(midpoint(e))$
              end for
          end if
      else
          $candidatePoints.append(c)$
      end if
  end for
  if candidatePoints is empty then
      /* empty means that either there are no boundary faces or no
         encroachment occurs.  In either case we keep the original point.
         */
      $candidatePoints.append(c)$
  end if
  return candidatePoints
```

Fig. 8: Encroachment rules of the centroid-based point-creation method.

candidate is created, the metric is interpolated using formula (6) of page 38. Depending on its location we use 2, 3 or 4 points if it lies on an edge, face or inside a tetrahedron. Inspired by [170], we store alongside the metric value at a point $\mathcal{M}(p)$ its logarithm $\log(\mathcal{M}(p))$. Although this requires more space, it reduces significantly the time required for metric interpolation.

Similarly to the isotropic code, the candidate points are compared to each other for proximity (in the metric space) and candidates too close to each other are disregarded. This step minimizes the number of short edges created at each iteration, thus allowing to employ edge collapse only at the end as a post-processing step (see also Figure 7). A similar approach called *Anisotropic Filtering* has been presented in [160].

3.2.2 LOCAL RECONNECTION IN METRIC SPACES

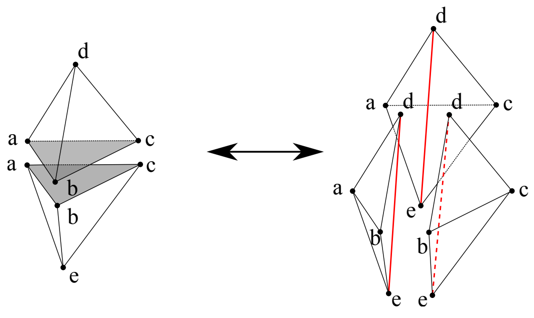
The local reconnection stage of *CDT3D* iterates over active tetrahedra of the mesh and optimizes their connectivity. A local reconnection pass consists of four types of flips [146] depicted in Figure 9. The flip operations are purely topological transformations, however, the criteria they use are based on geometrical quantities, and as such, they need to be modified in order to incorporate the metric-based information. The first criterion used in conjunction with a 2-3/3-2 flip is Delaunay-based: face *abc* will be replaced with edge *ed* if *d* is in the circumsphere of *abce* (see Figure 9a). Extensions of the Delaunay criterion for metric spaces has been suggested in [33] where the authors express the equations that derive the circumcenter of a tetrahedron using metric-based lengths. However, they mention that no general solution to these equations was found. To circumvent this issue, they provide approximations of the criterion based on the *Delaunay measure*. Let $K = (x_1, x_2, x_3, x_4)$ be a tetrahedron, the Delaunay measure of a point p with respect to K is defined as:

$$\alpha_{\mathcal{M}}(p, K) = \frac{d_{\mathcal{M}}(O_K, p)}{d_{\mathcal{M}}(O_K, x_1)} \tag{15}$$

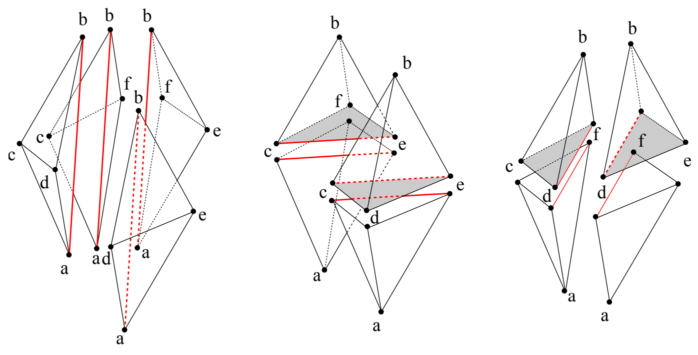
where, O_K is the circumcenter of K evaluated in metric $\mathcal{M}$. If $\alpha_{\mathcal{M}}(p, K) < 1$ then p is in the circumsphere of K. Notice that we did not specify $\mathcal{M}$ explicitly. In fact, by incorporating the metric from 1, 2 or even all 4 points of K one can get better approximations of the Delaunay criterion [33]. In this work, we adopt the criterion presented [79], that uses not only metric information from K put also from the point p itself:

$$\begin{cases} \alpha_{\mathcal{M}_p}(p, K) < 1 \\ \sum_{i=1}^{4} \alpha_{\mathcal{M}_{x_i}}(p, K) + \alpha_{\mathcal{M}_p}(p, K) < 5 \end{cases} \tag{16}$$

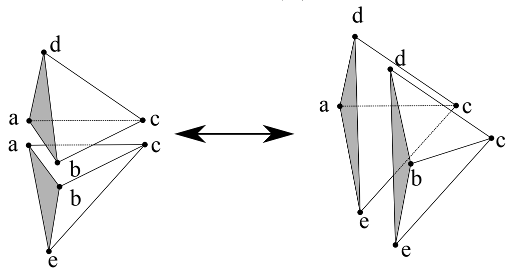
In practice, this criterion consists into evaluating the traditional Delaunay criterion in 5 different euclidean metric spaces and averaging the results. Similarly, in order to optimize the connectivity on the surface of the mesh, a 3D in-circle test (in the metric space) is



(a) 2-3 and 3-2 flips.



(b) The three configurations of a 4-4 flip.



(c) 2-2 Flip, *abd* and *abe* are boundary faces.

Fig. 9: Topological Flips utilized by *CDT3D* for local reconnection.

coupled with a 2-2 Flip. In particular, a surface edge ab is flipped for ed if d is in the circumcircle of abc (see Figure 9c).

The second criterion used is the maximization of the minimum Laplacian edge weight of an element K [22]. This criterion is combined with the 2-3/3-2 and the 4-4 Flips (see Figure 9a and 9b). In isotropic mesh generation it consists in evaluating the following quantity:

$$Q(K) = \max_{i=1\ldots6} \frac{\langle n_{F_{i,1}}, n_{F_{i,2}} \rangle}{6|K|}$$

where $F_{i,1}, F_{i,2}$ are the two faces attached to the i-th edge of a tetrahedron K and $n_{F_{i,1}}, n_{F_{i,2}}$ the respective face normals. The algorithm performs a flip only if the new configuration maximizes this value. For the anisotropic case, the formula for $Q(K)$ is adapted using a metric tensor $\mathcal{M}$ interpolated on the centroid of K. Moreover, in order to avoid the numerically expensive evaluation of the normals, the formula is replaced with one that uses only inner products [172]:

$$Q_{\mathcal{M}}(K) = \max_{i=1\ldots6} \frac{\langle e_i, e_j\rangle_{\mathcal{M}} \cdot \langle e_i, e_k\rangle_{\mathcal{M}} - \langle e_i, e_i\rangle_{\mathcal{M}} \cdot \langle e_j, e_k\rangle_{\mathcal{M}}}{6|K|_{\mathcal{M}}} \tag{17}$$

The new 2-2 Flip we added in *CDT3D* along with the ability to insert points on the boundary introduces essentially the ability to perform boundary refinement at the same time with the volume. The lack of this capability in our previous approach [250] resulted in sub-optimal results. Figure 10 compares the quality of our method prior and after adding boundary refinement capabilities. The introduction of boundary refinement capabilities improved the quality of the mesh by an order of magnitude.

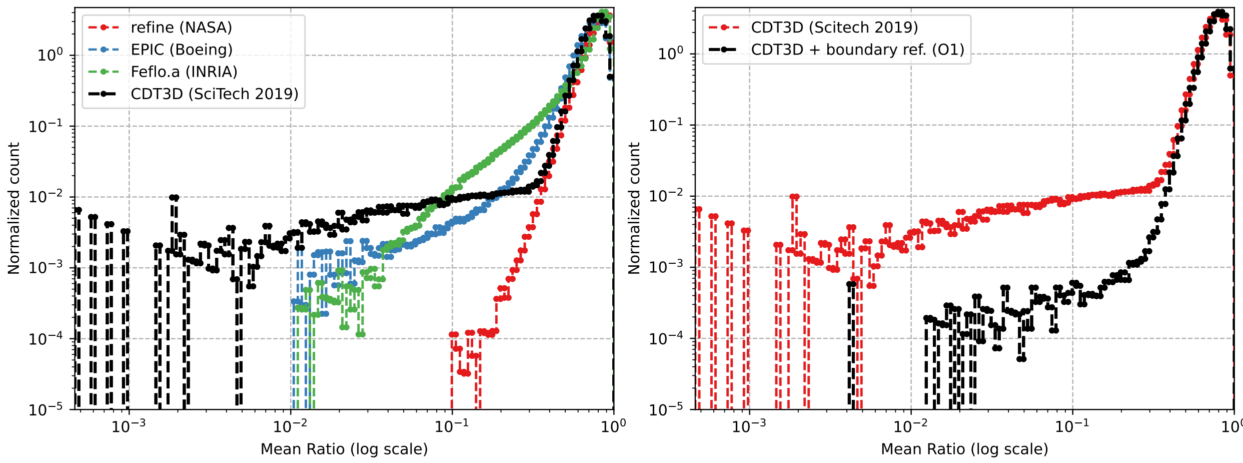


Fig. 10: Effect of boundary refinement to the quality of the final mesh. Left: Mesh quality of the initial implementation compared to other mesh adaptation methods [250]. Right: Mesh quality after the introduction of boundary refinement.

3.2.3 VERTEX SMOOTHING

During the Quality Improvement step of the isotropic mesh generation pipeline of *CDT3D* (see Figure 6) the user can enable a combination of Laplacian [38] and optimal point placement smoothing [82] applied upon all mesh vertices not lying on the boundary in order to further improve the mesh quality. Utilizing these methods for metric-based adaptation did not yield a substantial improvement in mesh quality while, on the other hand, added a significant overhead in the running time. As an alternative, a different vertex relocation strategy was employed. First, only vertices with at least one attached element that has mean ratio shape measure below 0.1 are considered. This value is based upon common practices present in the literature [129, 251]. Moreover, a non-smooth optimization method similar to the one presented in [97] was employed by optimizing the minimum value of the mean ratio measure among all the elements attached to the vertex. For simplicity, the current implementation does not utilize the integer programming-based solution presented in [97] or the computation of the active set of the gradients used in [140] in order to determine the optimal search direction. Instead, it uses a reduced search space comprised by the segments that connect the vertex to be relocated with the centroids of the faces of its cavity (see Figure 11 left). This search space was found to be sufficient for the cases of this study. Once the search space is determined, the vertex will be moved incrementally along all the search directions and the position that optimizes the quality will be selected.

For vertices lying on the surface, the search space is constrained to the segments that connect the moving vertex to the midpoints of the edges of the corresponding surface cavity (see Figure 11 right). If the vertex lies on a ridge the search directions are only two; towards either ends on the ridge. Along with the optimization criterion the method always ensures that no elements are inverted and thus no subsequent untangling step is needed. Figure 12 depicts the effect of the new vertex smoothing algorithm as part of the quality improvement loop of Figure 7 on top of previous optimizations.

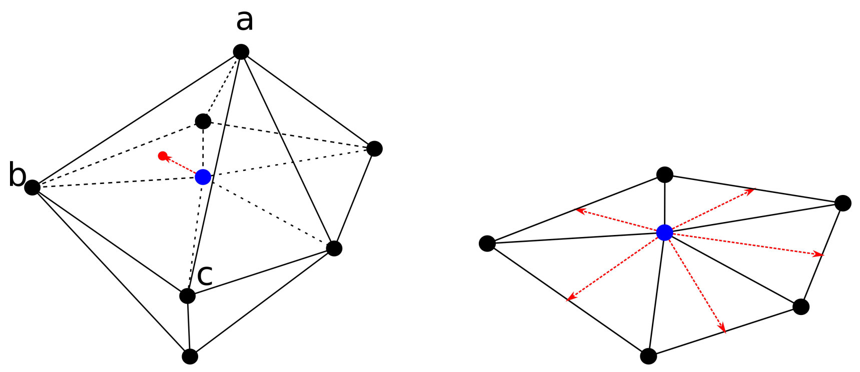


Fig. 11: Search Space for Smoothing Operation. Left: The search space of the blue point includes the 10 segments that connect the point to the centroids of the faces of the cavity. One of these segments is shown here in red. Right: For a point of the surface of the mesh, the search space consists of the edges that connect the moving point to the midpoints of the edges of the surface cavity.

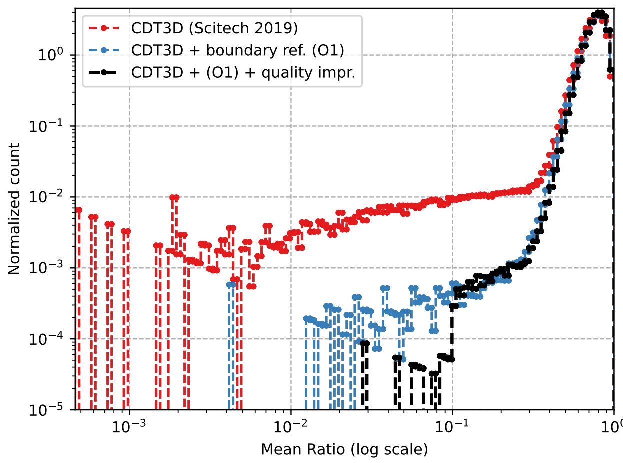


Fig. 12: Effect of the new vertex smoothing method on top of the improvements of the previous subsection.

3.2.4 EDGE COLLAPSE

The goal of an edge collapse operation is to remove edges with length smaller than a target value. It can also be used to make a mesh coarser or even create an empty mesh with almost no volume points which can serve as a starting point for an advancing front method [160]. In the context of the isotropic mesh generation, *CDT3D* utilizes it to improve quality and spacing in regions of the mesh where colliding fronts of the advancing front method create edges smaller that the target spacing. Its value is even greater for anisotropic mesh adaptation especially when the mesher is integrated in an adaptive loop due to the fact that the optimal mesh spacing is not known *a-priori* and thus an adaptation iteration may require to coarsen previously over-refined regions of the mesh.

In this work, the edge collapse operation is utilized as a pre- and post-refinement operation (see Figure 7). The pre-refinement step removes short edges present in the input mesh. By default an edge is considered short if it is smaller than $1/\sqrt{2}$ as measured by the metric-based length (see equation (1) in page 35). Depending on the input mesh, the user may increase this value in order to create a coarser initial mesh which can lead to better quality of the final mesh. This configuration is used later in the blast case presented in section 4.1.3.2, a similar approach appears in [181]. The post-refinement use of the operation allows to remove any short edges created during refinement. Its configuration is similar to the pre-refinement step.

The ability to adapt the boundary of the mesh at the same time with the volume, along with the new smoothing operation and the addition of a metric-based edge collapse enhanced the quality of the generated mesh by 3 orders of magnitude with respect to our initial attempt. Figure 13 presents our earlier data along with our new improved approach.

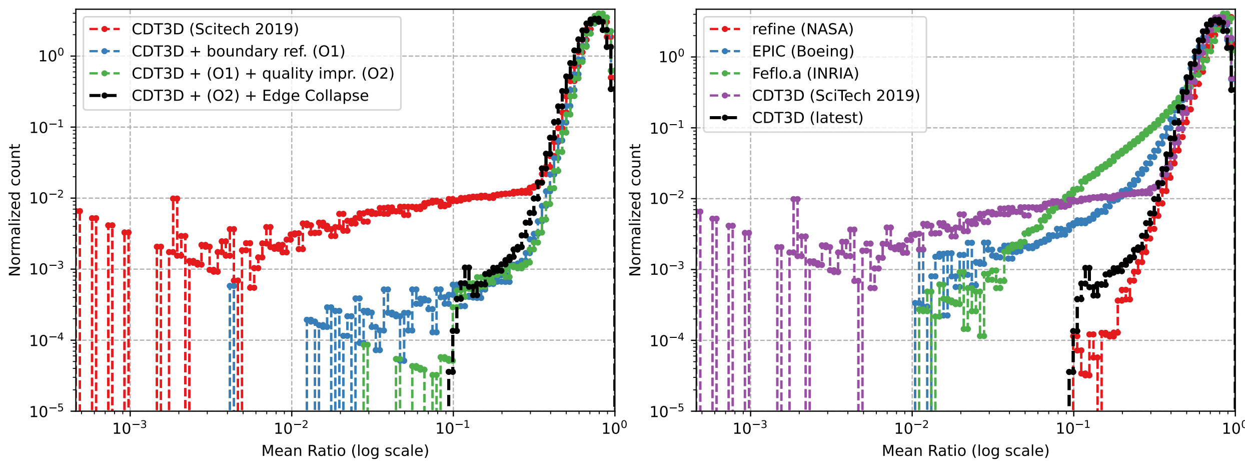


Fig. 13: Quality improvement adding Vertex Smoothing. Left: Effect of the edge collapse new vertex smoothing method on top of the improvements of the previous subsection. Right: Our new improved approach versus our previous results presented in [250].

3.3 HANDLING GEOMETRY THROUGH METRIC SPACES

Up until this point of the chapter, we considered as our input a surface mesh or, in general, a volume mesh conforming to the surface. Another very important representation is the geometrical description of the surface of the input model. Industrial applications as well as several high-quality research-focused workshops such as the High-Lift prediction workshop[14], the International Workshop on High-Order CFD Methods[15], the Sonic Boom Prediction Workshop[16], and the Geometry and Mesh Generation workshop[17] make extensive use of geometrical descriptions since many flow quantities of interest depend on the geometry. Building a mesh based on the geometrical description of the problem is essential for these studies. Moreover, accessing geometrical information while adapting the mesh leads to a better domain discretization and thus more accurate solution.

There are many methods that can be used to build this representation but traditionally CFD simulations, and the engineering community in general, leaned towards the use of the Boundary Representation method (BREP or B-rep) [243]. The B-rep method uses a combination of topological entities (Faces, Edges and Vertices) along with their geometrical

[14]`https://hiliftpw.larc.nasa.gov` (Accessed 2021-05-31).
[15]`https://how5.cenaero.be` (Accessed 2021-05-31).
[16]`https://lbpw.larc.nasa.gov` (Accessed 2021-05-31).
[17]`http://www.gmgworkshop.com` (Accessed 2021-05-31).

description as (analytic) surfaces, curves and points. B-rep data can also hold boolean operations such as intersection and union between entities as well as higher-level operations such as extrusion and sweeping [119]. The topological description holds adjacency information that allows to find all entities connected to a given one. For example, Figure 14 depicts part of the topological decomposition of the B-rep of a simple model. The model is composed out of 7 Faces (green). The right face is further decomposed into its constituent 5 Edges and 5 Vertices. Similar decomposition is stored for the rest of the faces but it is omitted for brevity. The geometrical information includes a two-variable parametric description $f(u, v)$, (called *uv-parametrization*), for each Face, a single variable parametric description $f(t)$, (called *t-parametrization*), for each Edge and the coordinates of each Vertex. For the rest of this chapter we will use the notions Geometrical Face/Edge/Vertex when referring to the B-rep entities to avoid confusion with entities of the mesh itself.

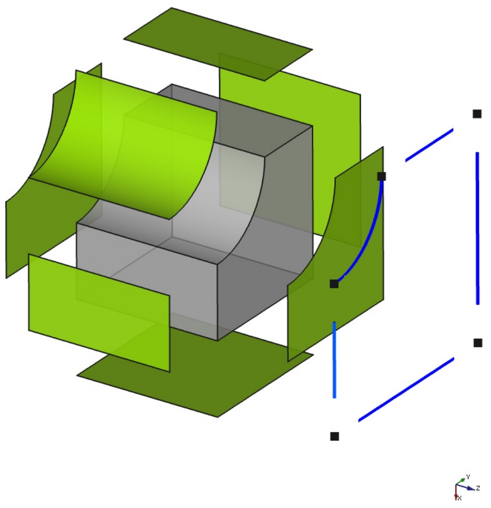


Fig. 14: Decomposition of the B-rep of the cube-cylinder case[18].

B-rep data are usually handled by a Computer-Aided Design (CAD) kernel which is responsible both for generating B-rep data and for handling queries to them. More details

[18]Acquired from `https://github.com/UGAWG/adapt-benchmarks`. (Accessed on 2021-05-31).

about the B-rep method, its advantages and disadvantages as well as how it compares to other methods can be found in [102, 243]. A detailed description of the importance but also the challenges that arise by using a CAD system during mesh adaptation appears in [202]. In this section, based on the material of [202], we provide a high-level summary of two approaches that can be used to incorporate geometrical information to a meshing procedure and describe in detail our implementation within the *CDT3D* library.

The B-rep information can be incorporated into a mesh generator in at least two ways. The first option is to build a surrogate geometry by constructing a discrete, often high order, surface mesh that captures all the features of the input geometry at a desired resolution. This approach has the advantage of controlling the fidelity of the constructed representation. Also, it allows fixing inconsistencies of the continuous representation that can occur due to different tolerances of each continuous patch. One can construct a surrogate geometry even when a geometrical description is not available based on the input surface mesh. This approach is currently utilized by the *Feflo.a* and *EPIC* mesh adaptation software [202]. Although quite powerful, this approach is out of the scope of this work due to its complexity and the fact that our focus is to provide a minimal implementation that can add preliminary Geometry support to *CDT3D*.

The second approach, which is used in this work, is to maintain an association between each boundary vertex of the mesh and its adjacent geometric entities. This approach allows to query the appropriate Geometrical entity through the CAD kernel. It has the disadvantage of inheriting the issues present in the B-rep model but, it provides access to the CAD kernel in a simple manner. Moreover, it aligns better with our goals which is to introduce preliminary support for B-rep data to our mesh adaptation method. Currently, this approach is also favored by the *refine* mesh mechanics suite [198].

In practice, we introduced in each mesh vertex a pointer to the lowest dimension geometric entity that is adjacent to. This information together with topological and geometrical queries to the geometry kernel allows to evaluate uv (or t) parameters for any mesh vertex. The current implementation makes use of the EGADS geometry kernel [121] through a generic API which could be adapted for another CAD kernel in the future.

Geometry information is used throughout the mesh adaptation procedure in several ways. First, newly introduced boundary points are projected to the surface using a dedicated module (see Figure 7) right after vertex insertion. Projection is performed by evaluating the closest point p' of the B-rep to a given mesh vertex p. This capability is provided by the CAD kernel. The mesh vertex p is then relocated to p' only if this operation does not create any inverted elements attached to vertex p. If it does, we try to move p to $(p+p')/2$, i.e., the midpoint between the two points. This procedure is applied recursively until we find a valid position or reaching a recursion limit which we set to 5. Mesh vertices that were placed in an intermediate position are recorded and they are included for projection in the next iteration of the algorithm. The local reconnection that will be applied on the vicinity of point p may enable to move the point closer to its projection in a subsequent iteration. Projecting a mesh vertex to the B-rep involves a Newton-Raphson root finding method and therefore its speed and result depends heavily on the initial guess of the projected point. To speedup the procedure, we approximate the uv (or t) parameters of a newly created point during the point creating module based on the values of the vertices belonging to the triangle or edge being split. This approximate uv (or t) parameters are then cached for this point and they are used as initial guess during the next projection stage.

Information of the analytic expression of the underlying surface is also used to minimize the deviation of the discrete surface mesh form its analytic description. In practice, the deviation is evaluated as the dot product between the normal of a discrete triangle and the normal of the geometry surface evaluated at the centroid of discrete triangle [202], (see also Figure 15). The deviation is minimized as part of the local reconnection pass (see Figure 7) using 2-2 Flips (see Figure 9c). The Edge Collapse operation can also use deviation of the surface cavity as an extra quality criterion when deciding whether a surface edge should be collapsed. Controlling the deviation not only produces a mesh that approximates the surface better, but makes the operations more robust avoiding cases that will lead to tangling the mesh.

Geometry constrains such as curvature and feature size can be expressed also as lengths and directional information and can be therefore encapsulated into yet another metric [110] which is combined with the solution based information via metric intersection (see Section 2.2.3.1). In this work, we follow the approach presented in [202] and its implementation in the *refine* mesh mechanics suite [198] in order to build a feature-based metric derived from the geometry model. When utilizing CAD projection, we also found advantageous to perform vertex smoothing at the end of the adaptive iteration (see Figure 7) allowing to improve the quality on the vicinity of the projected vertices.

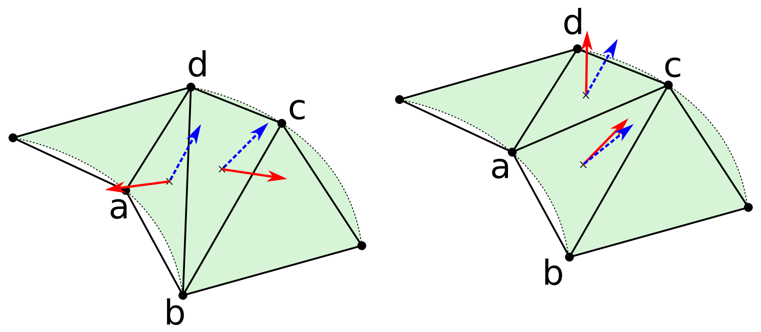


Fig. 15: Deviation Improvement. A flip of the edge *bd* for *ac* reduces the deviation between the discrete normals (red solid vectors) and the analytic normals computed at the centroids of the triangles (blue dashed vectors).

3.4 SPECULATIVE IMPLEMENTATION IN THE CONTEXT OF THE *TELESCOPIC APPROACH*

In this section, the speculative fine-grained scheme presented in [85] is extended to the operations presented in the previous sections. The *speculative* or *optimistic* method lies on the bottom (closer to the CPU) of the *Telescopic Approach* [61] and it is designed to take advantage of the low cost of communication inside the chip by utilizing direct memory access among the threads. In contrast to higher levels of the *Telescopic Approach*, no explicit data decomposition is performed. Instead, each thread will attempt to perform an operation while capturing the necessary dependencies on the fly. Failure to do so results in a rollback and the method will try again later if the operation is still applicable.

3.4.1 SPECULATIVE LOCAL RECONNECTION KERNEL

The fine-grained speculative scheme for local reconnection employed by *CDT3D* has been already presented in detail in [85]. We provide here only a summary. *CDT3D* maintains at any time a list of "active" (i.e., eligible for reconnection) elements. This list is split into "buckets" (i.e., sublists) which are then distributed among the threads. The elements within a bucket have no geometrical relation and the grouping is performed only for enabling an efficient and simple work sharing algorithm [85]. After the buckets are formed, each thread iterates its buckets and attempts to lock the vertices of the cavity of a flip using atomic operations. For the flips presented in this work (see Figure 9), it boils down to the element itself and some of its face neighbors. If the attempt is successful, the objective function is evaluated before the flip and after for each of the applicable flips. If the new flip improves the objective function (which can be either of the two mentioned above), then it is applied otherwise, the element connectivity remains unchanged. The thread will then unlock all the vertices and proceed to the next element of the bucket. Unsuccessful attempt to lock any part of a cavity results into unlocking any acquired vertices and moving to the next element in the bucket. The skipped element will be revisited (if it is not deleted as part of another flip) in a subsequent iteration. The algorithm iterates until no flip can be applied or if a maximum limit of iterations is reached.

3.4.2 PARALLEL POINT CREATION KERNEL[19]

The point creation kernel is structured in a similar fashion to the local reconnection kernel. After the buckets are formed, they are assigned to different threads and each thread iterates the elements of its bucket in order to generate *candidate* points for insertion.

New candidate points are compared against existing mesh vertices and currently candidates for proximity (in the metric space). This check allows to avoid the creation of points too close to each other that will impact the quality and the local density requirements. A similar approach named *Anisotropic Filtering* is presented in [160]. Once a point passes all proximity tests it is stored in the list of the contained element. Storing the candidate points in the contained element gives a significant advantage; both the proximity checks and the subsequent point insertion step (see Figure 7) can be performed in constant time since the point location step of the direct insertion kernel will only require constant time to execute.

In contrast to local reconnection, the point creation step does not perform any topological modification and thus no cavity locks are required. Moreover, vertices are allocated into thread-local memory pools [49] and thus vertex allocation can be performed asynchronously. The only step that requires synchronization is when it comes to adding the candidate point to the internal list of the contained element. Our experiments showed that this lock is short-lived and making use of spinlocks is a sufficiently efficient solution to handle concurrency.

3.4.3 SPECULATIVE EDGE COLLAPSE AND VERTEX SMOOTHING

The data structures of *CDT3D* have been designed for efficiency and low memory consumption and as such only vertices, tetrahedra and surface triangles are stored explicitly in memory [82]. This however, impedes the creation of an efficient edge collapse operation

[19]Parallel implementation was developed in collaboration with Fotis Drakopoulos.

based on the element-lists described above since the same edge will be visited multiple times as it can be adjacent to an arbitrary number of elements. Moreover, the lack of a dedicated edge object does not allow to iterate through the edges and locate those that require collapsing. Although, the edge information could be generated once and maintained throughout the mesh adaptation procedure, it was found to affect significantly the runtime of the method since it increases the amount of book-keeping after each topological modification [82].

For these reasons, the edge collapse operation is structured around iterating vertices instead of tetrahedra. However, *CDT3D* has no global structure which holds all the vertices and the actual vertex-based data are managed by thread-local memory pools [49] that make the operation of iterating the vertices non-trivial. To overcome this issue one should first observe that the memory pools described in [49] and utilized by *CDT3D* are designed as a list of arrays. This implementation detail allowed us to build random-access iterators based on the C++ API[20] by defining the increment `++` operator appropriately so that it can jump between the different arrays of the list. The new iterators not only offer simple access to the underlying memory pools but, they do it in a cache-friendly manner. Moreover, this streamlined design gives the opportunity to experiment with concurrent constructs such as `#pragma omp parallel for` of OpenMP to exploit parallelism.

For edge coarsening the algorithm iterates through the vertices of the mesh and exploits parallelism utilizing a `#pragma omp parallel for schedule(guided)` OpenMP construct. Each thread picks and locks (speculatively) the vertex (a in Figure 16) corresponding to the iterator value. Then it speculatively locks its adjacent vertices (blue in Figure 16). If any of the locks fails, the thread will release any acquired locks and it will pick the next vertex. Notice that locking the vertices implicitly grants exclusive access to all their adjacent tetrahedra (red elements of Figure 16). Once the required locks have been acquired, the edge lengths between the vertex a and the rest of the vertices of its cavity are evaluated. If an edge with length less that a user-defined value is found (default : $1/\sqrt{2}$ in metric space) then the edge will be collapsed. Additional criteria such as skipping edge collapses that will increase the surface deviation (see Section 3.3) are also applied for edges on the surface of the mesh. Finally, the edge is collapsed by moving the second point to the first. The `guided` scheduler was selected because it performs on average better for cases that require different levels of coarsening such as the delta wing case and the blast case described later in Section 4.1.3.

Vertex smoothing follows a similar pattern. The vertices are iterated in parallel and

[20]`https://en.cppreference.com/w/cpp/iterator/random_access_iterator` (Accessed 2021-06-01).

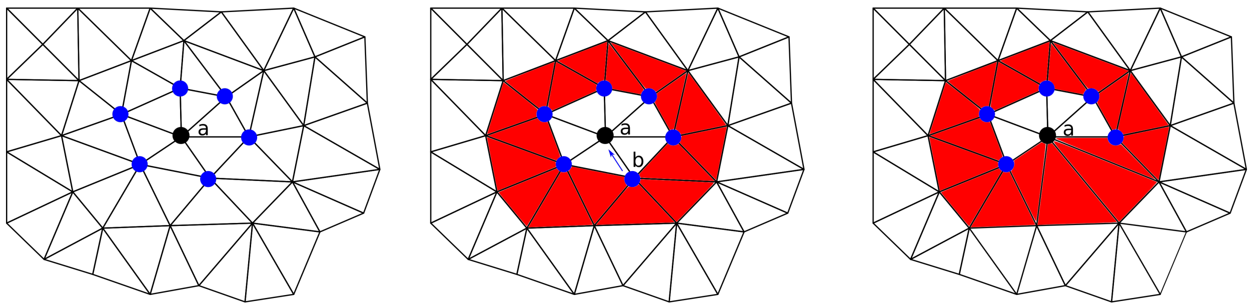


Fig. 16: Steps of speculative edge collapse.

the vertex corresponding to the iterator value along with its adjacent vertices are locked speculatively. Then, if any of the attached tetrahedra has quality below a user-provided limit, the vertex will be relocated using the method described in Section 3.2.3. The procedure is repeated for a fixed number of iterations. The quality metric and the limit are user-configurable, for this study we use the mean ratio (see equation (10) in page 33) and a limit of 0.1.

3.4.4 PARALLEL IMPLEMENTATION

In this section we discuss the efficiency of our parallel implementation for the metric-based operations presented in the previous section as well as for the end-to-end mesh adaptation process.

As input we use a mesh of a delta wing with planar faces and $800,000$ metric complexity. The input mesh has $1,439,310$ vertices and $8,470,523$ tetrahedra while, the target metric has complexity $1,600,000$. The difference in complexity causes the mesh size to double during adaptation. The experiments were performed on the `wahab` cluster of Old Dominion University using dual socket nodes equipped with two Intel®Xeon®Gold 6148 CPU @ 2.40GHz (20 slots) and 368 GB of memory. The compiler is `gcc 7.5.0` and the compiler flags `-O3 -DNDEBUG -march=native`. Each run was repeated 5 times and the results were averaged using the geometrical mean [91]. For the base case we ran the parallel code using one core.

Figure 17 depicts the total efficiency of the method as well as its breakdown with respect to the two main modules of *CDT3D* (see also Figure 7). The end-to-end efficiency is 92.3% at 40 cores. The efficiency for the Mesh Adaptation and Mesh Quality Improvement modules is

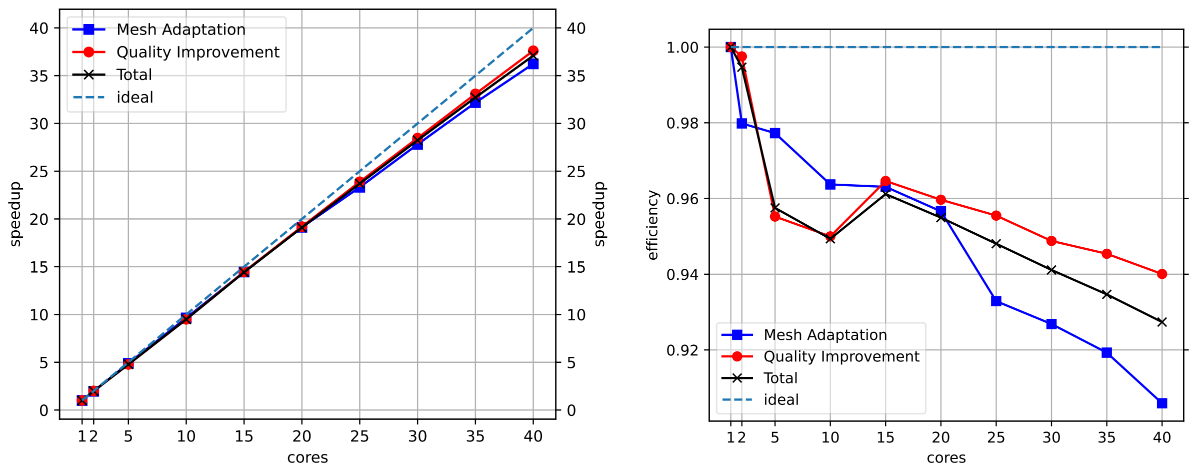


Fig. 17: Speedup and efficiency of the two main modules of *CDT3D* (see also Figure 7).

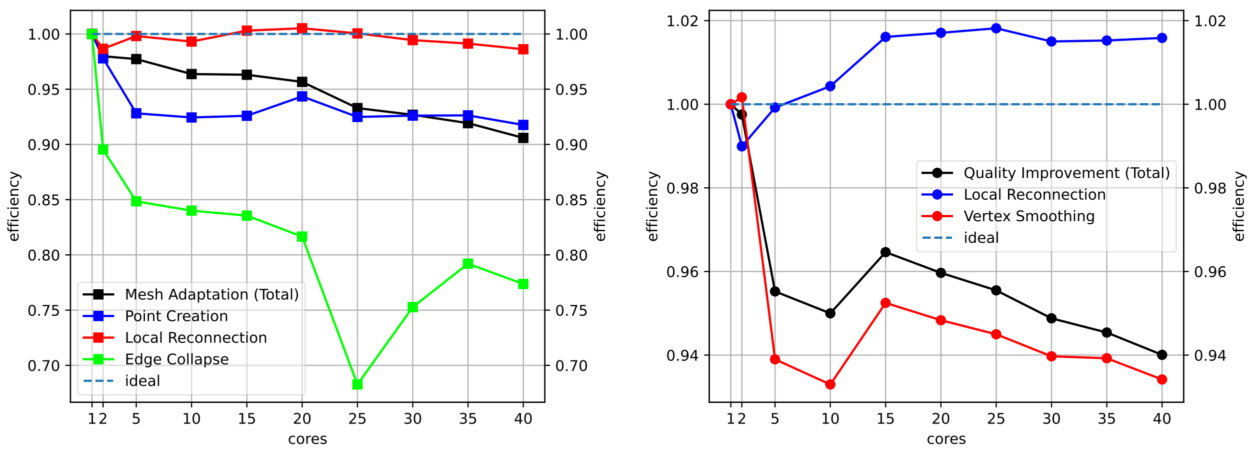


Fig. 18: Efficiency breakdown of the mesh adaptation and quality improvement modules of *CDT3D* (see also Figure 7).

90.5% and 94% respectively. Figure 18a presents a breakdown for the efficiency of the Mesh Adaptation module. The Local Reconnection operation performs the best with more than

98% efficiency. The super-linear speedup is caused by the "buckets" described in Section 3.4. Splitting the list of active elements into buckets and repeatedly performing reconnection over the same bucket improves the cache locality. Point Creation benefits from the same construct but its spin-lock implementation for updating the internal list of the contained element (see Section 3.4.2) results into a lower efficiency. Edge Collapse exhibits a lower speedup in comparison to the other two operations due to the generic OpenMP implementation that was used to exploit parallelism. Still, this implementation of Edge Collapse delivers $80-85\%$ efficiency for up to 20 cores and $75-80\%$ efficiency for more cores. At 25 cores the edge collapse efficiency drops significantly. This is in part attributed to the dual nature of the host machine. At 25 cores the code is using one and a half sockets and the OpenMP back-end of the operation does not have any special treatment for accessing memory from a different socket. For Quality improvement (see Figure 18b) the super-linear performance of Local Reconnection is more prominent due to the (approximately) constant size of the mesh during the Quality improvement phase (no vertices are introduced). The Vertex Smoothing operation exhibits 93.4% efficiency on 40 threads. Notice also that the efficiency curve of the Quality Improvement stage (black) follows the trends of the smoothing operation. This is due to the fact that smoothing is the dominant operation in terms of time and also because the efficiency of local reconnection is approximately constant. Figure 19 presents a breakdown of the mesh adaptation module of *CDT3D*. Local reconnection accounts for more than 75% of the total mesh adaptation time. The other two major parallel mesh operations Point Creation and Edge Collapse, are responsible for less than 10% of the mesh adaptation time. The effect of the sequential point insertion is becoming increasing higher as expected by Amdahl's law [8] but still it is less than 4% of the total time.

Figure 20 depicts the percentage of the total time that corresponds to each operation. The time to smooth the vertices corresponds to about 60% of the total running time while, mesh adaptation takes about 12% of the total time.

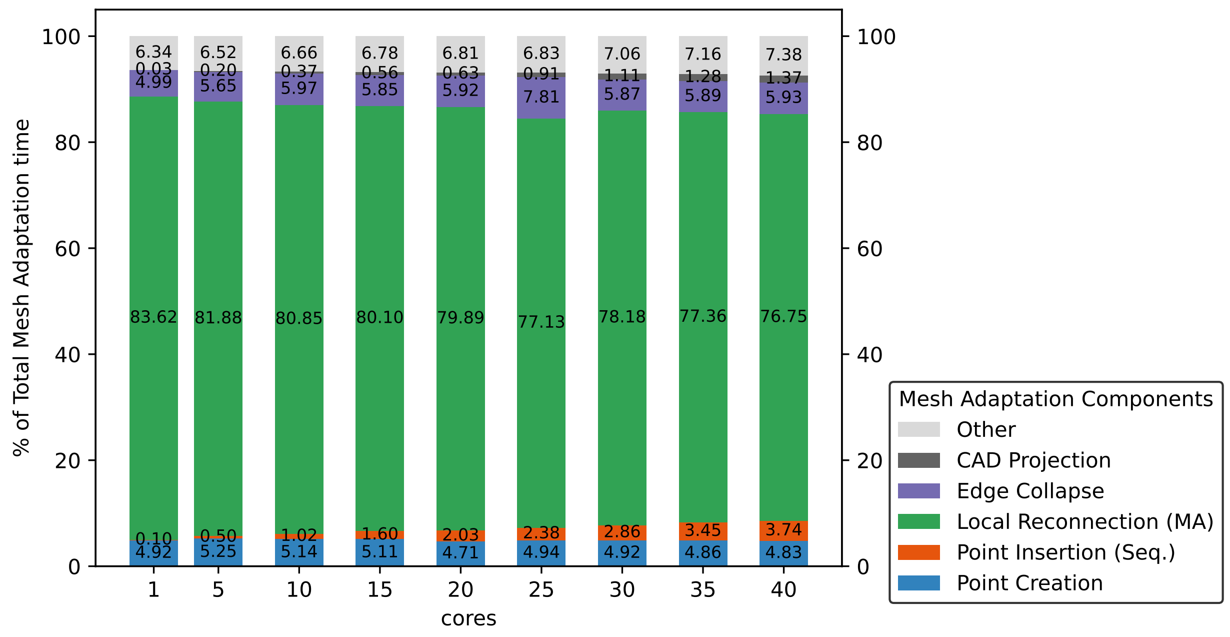


Fig. 19: Breakdown of the mesh adaptation time into the basic operations of *CDT3D* (see also Figure 7).

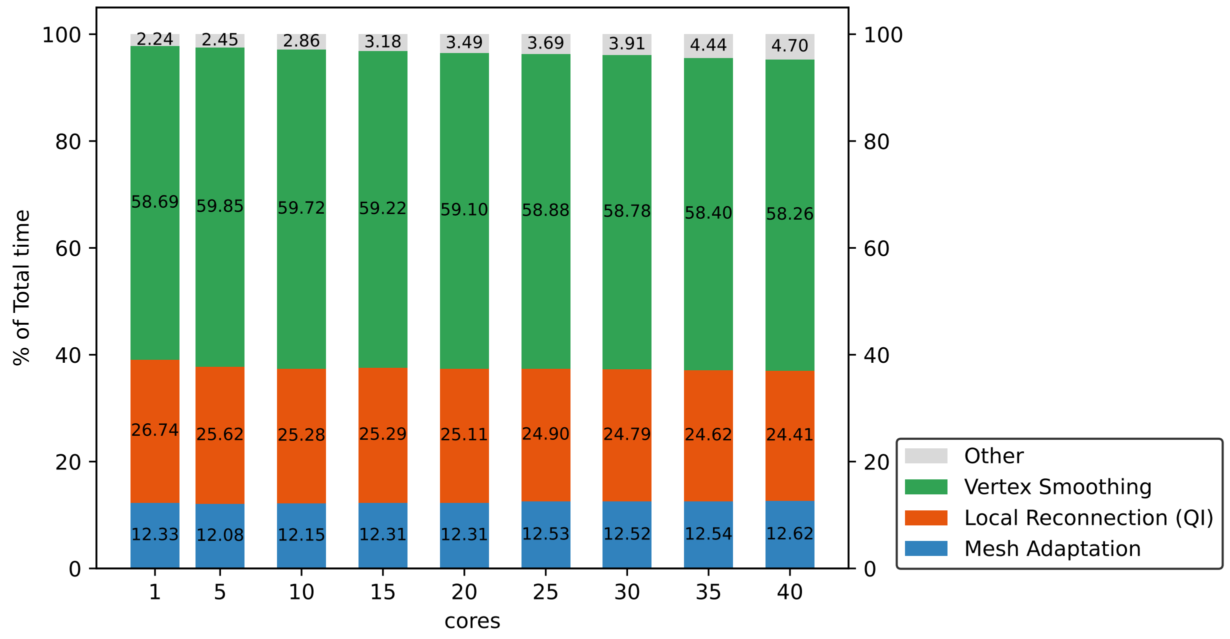


Fig. 20: Breakdown of the total time of *CDT3D* (see also Figure 7).

CHAPTER 4

EVALUATION

The cases we use to evaluate our method are separated in two categories. First, in Section 4.1, we focus on cases that target a fixed complexity and provide (i) quantitative results with respect to parallel performance and (ii) qualitative with respect to metric conformity of the adapted mesh. The goal of metric conformity is given a (mesh, metric) pair to create a unit mesh [166] where the edges are unit-length and the elements are unit-volume with respect to the given metric. In particular, we use equation (9) of page 39 to measure the length of an edge and the mean ratio equation (10) of page 39. The parallel performance is evaluated in terms of traditional metrics such as strong and weak speedup.

For the second group of cases, in Section 4.2, we build an adaptive pipeline with open-source and publicly available tools and utilize our method as the mesh adaptation module. The adaptive pipeline is an iterative process that creates a (mesh, metric) pair at each iteration which we provide to our method. To evaluate the effectiveness of our approach as part of the adaptation pipeline, we compare qualitative data derived from the adaptation pipeline with results drawn from the literature.

4.1 MESH ADAPTATION AT CONSTANT COMPLEXITY

In this section, we evaluate our method using cases where the metric complexity remains constant. We use two ways to specify the metric field. First, directly as an analytic function $F(p) = \mathcal{M}(p)\,, p \in \mathbb{R}^3$. This approach allows to study the metric conformity in isolation to the metric construction method. The second approach uses solution-based metric fields derived from a CFD solution. In particular, we use metric fields constructed from the scalar field defined by the local Mach number of a flow The metric construction scheme is the multiscale metric described in Section 2.2.3.3. The complexity of the geometries we utilize increases incrementally and includes curved geometries with associated CAD data. Specifically the cases of this section include:

- Analytic metric field defined over a planar surface (Section 4.1.1).
- Analytic metric fields defined over a curved surface (Section 4.1.2).
- Solution-based metric fields defined over a planar surface (Section 4.1.3).
- Solution-based metric fields defined over a curved surface (Section 4.1.4).

Before delving into the results, we define the analytic metric fields we use in this study which are drawn from the first benchmark [129] of the Unstructured Grid Adaptation Working Group (UGAWG)[21].

$$\texttt{Linear} := \begin{pmatrix} h_x^{-2} & 0 & 0 \\ 0 & h_y^{-2} & 0 \\ 0 & 0 & h_z^{-2} \end{pmatrix} \tag{18}$$

where: $h_x = h_y = 10^{-1}, h_z = h_0 + 2(0.1 - h_0)|z - 0.5|, h_0 = 0.001$

$$\texttt{Polar-1} := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_r^2 & 0 & 0 \\ 0 & h_\theta^2 & 0 \\ 0 & 0 & h_z^2 \end{pmatrix} \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{19}$$

where: $r = \sqrt{x^2 + y^2}, \theta = \text{atan2}(y, x), h_r = h_\theta = 0.1, h_z = h_0 + 2(0.1 - h_0)|r - 0.5|,$

$h_0 = 0.001$

$\texttt{Polar-2} := \texttt{Polar-1}$

$$\text{where: } d = 10\,(0.6 - r) \quad \text{and} \quad h_\theta = \begin{cases} 0.1 & \text{if } d < 0 \\ d/40 + 0.1(1 - d) & \text{if } d \geq 0 \end{cases} \tag{20}$$

`Linear` represents a shear layer in absence of curvature, while `Polar-1` and `Polar-2` a curved shear layer. `Polar-2` is derived from `Polar-1` using a low gradation so it is possible to satisfy with high-quality elements by resolving curvature in the tangential direction near the layer.

Sections 4.1.1 and 4.1.3 include also quantitative performance data in terms of speedup and total running time. The experimental set up for these data is the following: Both *refine* and *CDT3D* were compiled using the `intel 19.0.4.243` compiler and data were collected on Old Dominion University's `wahab` cluster using dual socket nodes each one featuring two Intel®Xeon®Gold 6148 CPU @ 2.40GHz (20 slots) and 368GB of memory. *Feflo.a* data for Section 4.1.1 and the strong scaling data of 4.1.3.1 were collected on a dual socket machine equipped with two Intel®Xeon®E5-2697 v2 @ 2.70GHz (12 slots) CPUs, while for Section 4.1.3.2 and the weak scaling data of 4.1.3.1 on a dual socket machine with two Intel®Xeon®E5-2680 v2 @ 2.80GHz (20 slots) CPUs. The execution times and hardware specifications are omitted for the *EPIC* results to protect proprietary data.

[21]`https://ugawg.github.io/` (Accessed 2021-05-31).

4.1.1 ANALYTIC METRIC OVER PLANAR SURFACE

The first geometry is a unit cube with `Polar-2` defined over the domain. The initial mesh conforms to the `Polar-2` metric with a complexity of 7,600 (see Figure 21). The values of the metric field at the vertices of the mesh are scaled to 500,000 complexity for this test using formula (13) from page 43. Adapted meshes with approximately 1,000,000 vertices are expected. For this case we compare our method with *EPIC*, *refine* and *Feflo.a*.

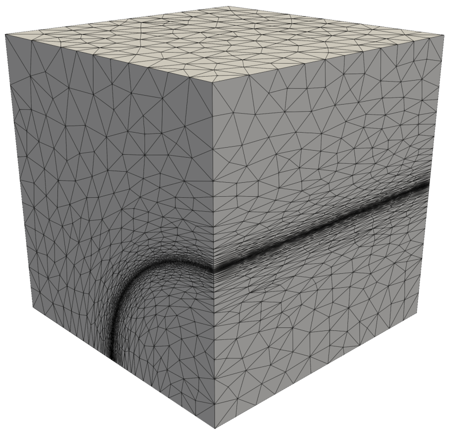


Fig. 21: Cube with polar-2 analytic metric, complexity of 7,600.

Metric conformity results in terms of edge length histograms and mean ratio measure are shown in Figures 22 and 23. The mean ratio is bounded between one and zero, where a mean ratio near one indicates better metric conformity than a mean ratio near zero. In linear scale, all methods appear to exhibit good overall quality with *refine* generating the highest number of elements in the range $[0.8, 1.0]$. The log scale makes the differences more prevalent. *refine* produces elements with the highest minimum mean ratio of 0.4, *CDT3D* has the second best quality result, while the lowest mean ratio is around 0.01 for *EPIC* and *Feflo.a*. The ideal edge length distribution is clustered tightly around unity. Figure 23 (left) reveals that *refine*, *CDT3D* and *EPIC* generated edges with less variance, while *Feflo.a* produced both the shortest edges and the largest edges.

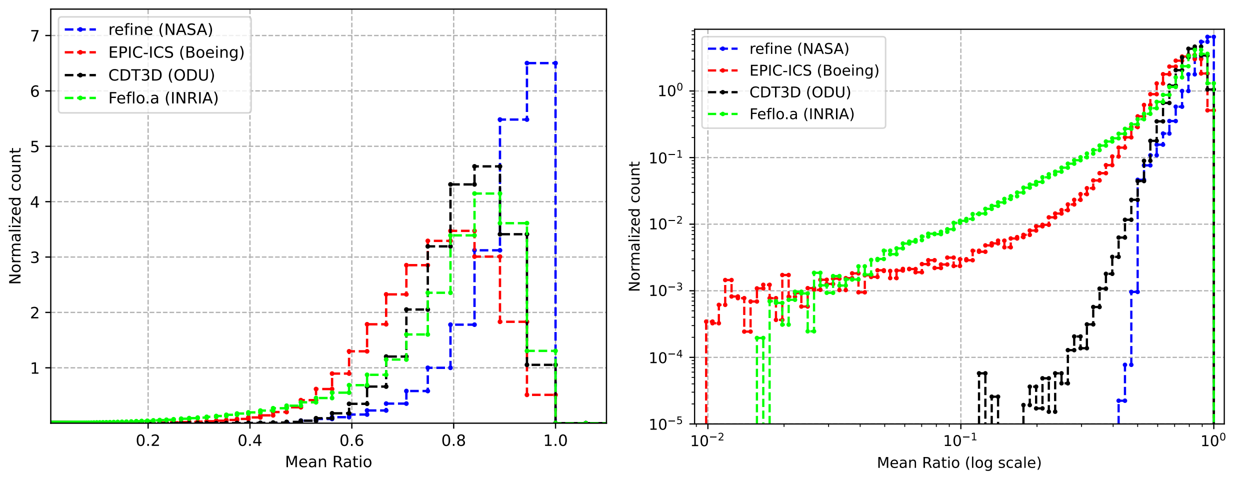


Fig. 22: Comparison of the mean ratio of the generated grids for the Cube case in linear and logarithmic scales.

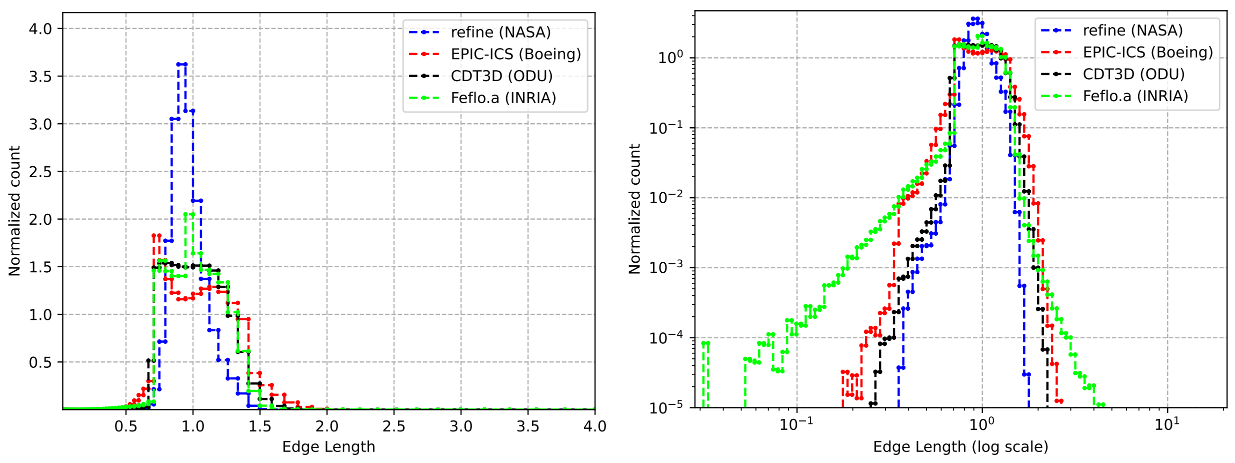


Fig. 23: Comparison of the edge lengths of the generated grids for the Cube case in linear and logarithmic scales.

The small target mesh size makes the strong scaling tests a challenge for a large number of cores since the computation time per core becomes very small and the communication overhead dominates the running time. The scaling results obtained using *refine*, *CDT3D* and *EPIC* to adapt the initial 7,600 complexity grid to conform to the 500,000 complexity as a function of number of cores is shown in Figure 24. *Feflo.a* is excluded from the scaling results since the speedup gain is too small. This is due to the high startup cost of the decomposition method which is not amortized for a small mesh. More details for this cost appear in Section 4.1.3.2. All three methods exhibit linear scaling at low number of cores. At higher core numbers, the speedup becomes constant for both *EPIC* and *refine*.

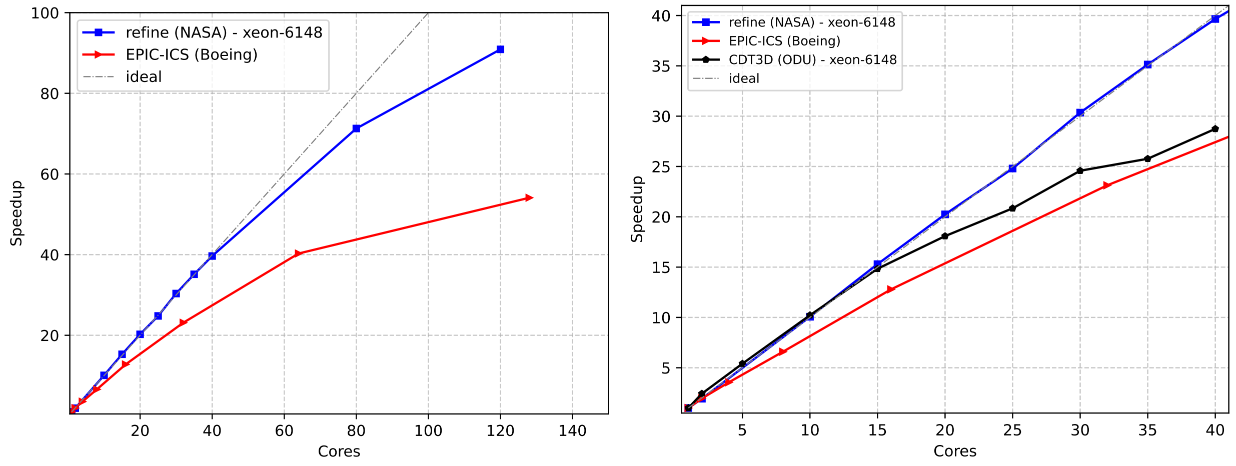


Fig. 24: Speedup results for the cube case. Left: Speedup for the cube case adapted from 7,600 complexity to 500,000 complexity. Right: Zoom-in view of the data for up to 40 cores. (Base case is the sequential time of each software.).

4.1.2 ANALYTIC METRIC FIELDS ON CURVED SURFACES

The next test case introduces CAD data. In particular, we use the `cube-cylinder` case described in [129]. The `cube-cylinder` case contains a cylinder of radius 0.5 oriented along the z-axis positioned at $x = 0, y = 0$ and subtracted from a unit cube with its lower corner positioned at $(0, 0, 0)$. Figure 25 depicts the two solids as well as the end-result. The input geometry is provided to *CDT3D* using the `.egads` file present at the repository[22] of the paper. As metric field we use `Linear`, `Polar-1` and `Polar-2`.

For these three cases, the metric field is passed to *CDT3D* as an analytic function, thus

[22]`https://github.com/UGAWG/adapt-benchmarks` (Accessed 2021-06-01).

giving the ability to obtain an exact value each time we create a new point. A metric field can be passed with the following API:

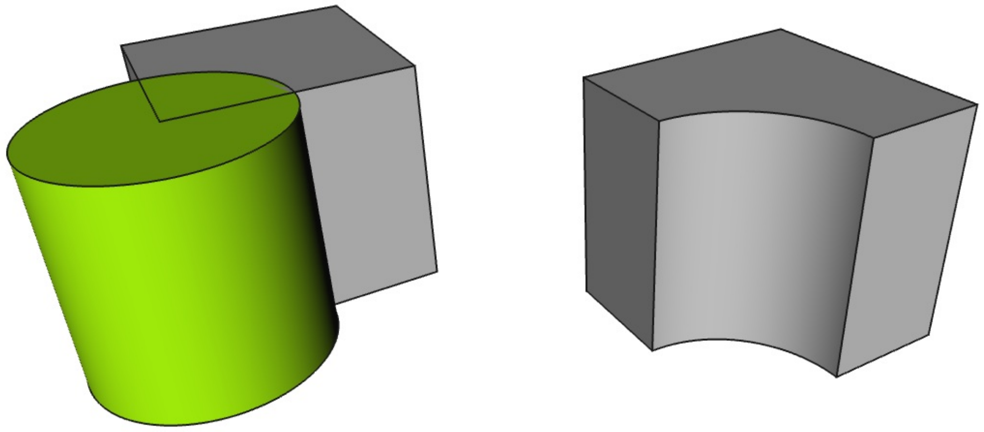


Fig. 25: Constituent solids of the cube-cylinder case along with end-result.

```
/**
* @brief API for an analytically defined metric field.
* @param x coordinates of the point
* @param M metric tensor holding the M(x)
*/
void MetricTensorOnPoint(double* x, MetricTensor* M);
```

Listing 4.1: API for providing an analytic metric field.

For all three cases we start with the same mesh containing only 286 vertices (see Figure 26a) and perform three adaptive iterations by supplying each time the mesh of the previous iteration as input.

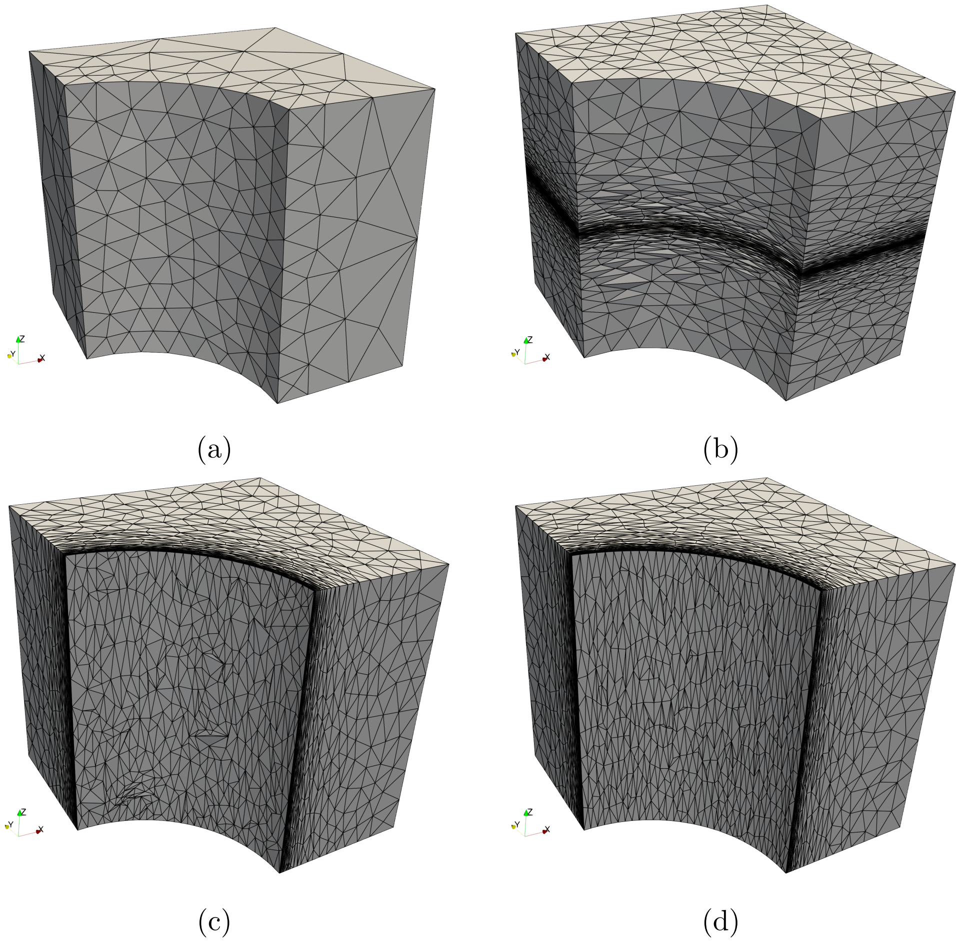

(a) (b) (c) (d)

Fig. 26: Input mesh (a) and adapted result (b) - (d) for the three analytic metrics.

4.1.2.1 Results using `Linear`

Figure 27 presents a comparison of the quality of the generated mesh in comparison to the five methods included in the first benchmark of the UGAWG. The comparison data are drawn from the paper's repository[23]. For this case we compare *CDT3D* against *EPIC*, *refine*, *Feflo.a*, *Pragmatic*, and *Omega_h*. Descriptions of all the methods appear in Section 2.1.

The linear scale of the edge-lengths reveals that the peak of the distribution for *CDT3D* is off-centered, similar to *Feflo.a*, but still within the limits of the rest of the method. The log scale reveals that the longest edge produced is closer to the end of the specturm of the

[23]`https://github.com/UGAWG/adapt-results` (Accessed 2021-05-31).

other methods while the shortest is close to the value produced by *refine*. In terms of mean ratio quality, *CDT3D* produces a value slightly above 0.1 which lies in the middle of the results of the other methods.

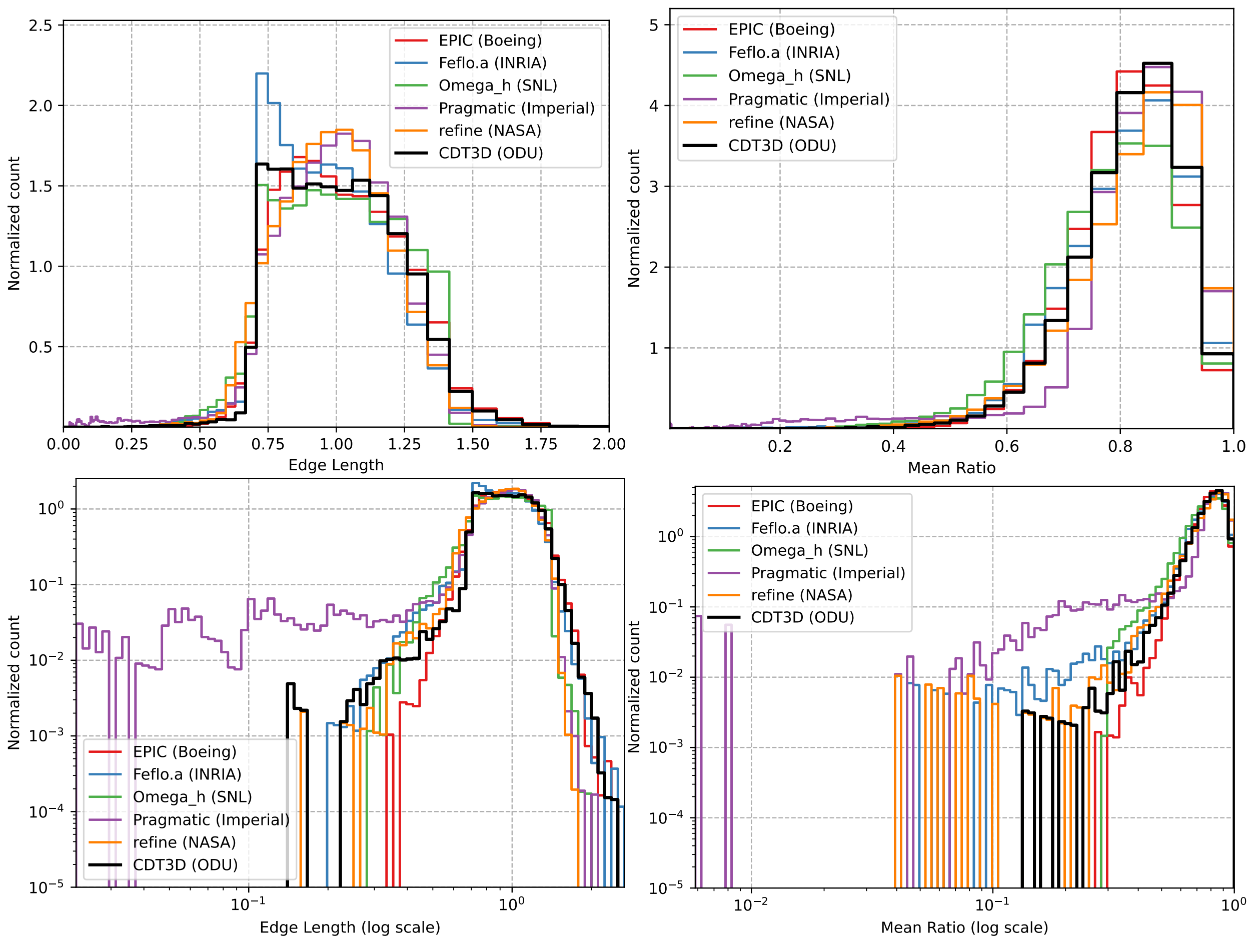


Fig. 27: Quality comparison of the `Linear` case.

4.1.2.2 Results using `Polar-1`

Figure 28 compares the quality of the generated mesh with the five methods included in the first benchmark of the UGAWG. For *refine* we replaced the mesh of the paper's repository with one generated using the latest[24] version since, the quality evaluation script of the *refine* suite detects invalid elements in the original mesh. As discussed in the original paper [129] the `cube-cylinder-polar1` case put a lot of stress to the adaptation methods due to the fact that the anisotropic layer is positioned exactly on the curved section of the mesh and has a large tangential spacing. The effect on *CDT3D* can be seen mainly on the log scale of the edge-length distribution where *CDT3D* produces produces few very long edges as well as several shorter than ideal. Still, the results are within the trends of other methods. In terms of the mean ratio measure, *CDT3D* performs better than *Feflo.a* and *Pragmatic* but falls slightly behind the other methods. The latest version of *refine* performs the best in both metrics in part due to added features since the publication of the original paper which include the introduction of pliant smoothing [134] and the use of the cavity operator [159].

4.1.2.3 Results using `Polar-2`

Finally, Figure 29 presents the results of the evaluation for the `Polar-2` case, which reduces the tangential spacing of the metric at the curved boundary. Similar to the `Linear` case we use the meshes provided in the paper's repository for this evaluation. The trends of *CDT3D* are similar to the previous two cases. The log scale of the edge length histogram reveals that *CDT3D* produces a few edges smaller than most of the other methods. On the other end of the spectrum, the longest edge of *CDT3D* is close to the one produces from *Feflo.a* but shorter that *refine*. In terms of mean ratio measure, *CDT3D* produces a distribution similar to the rest of the methods and a minimum value above 0.1, thus producing the third best value after *EPIC* and *Omega_h*.

[24]As of 2020-10-04.

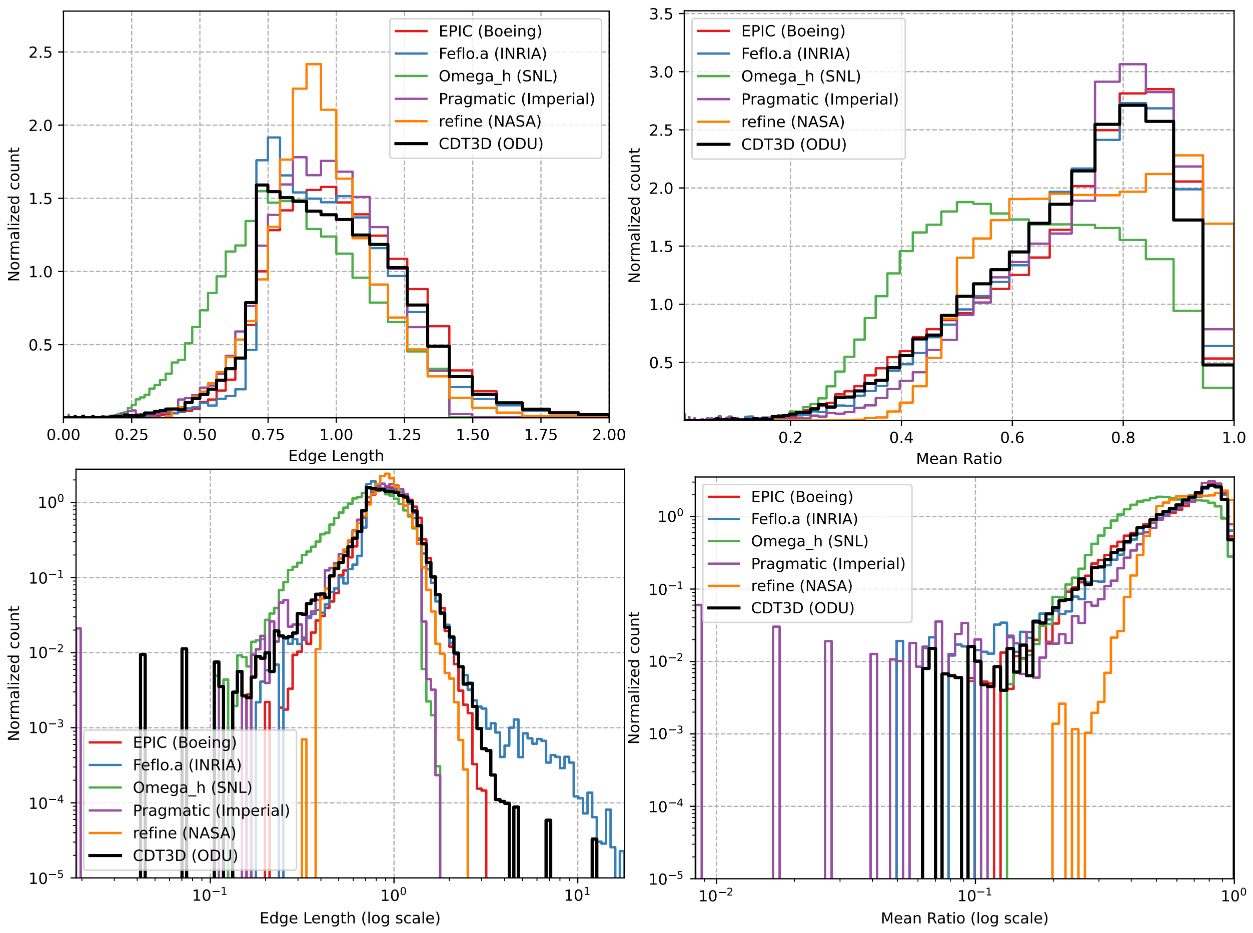


Fig. 28: Quality comparison of the `Polar-1` case.

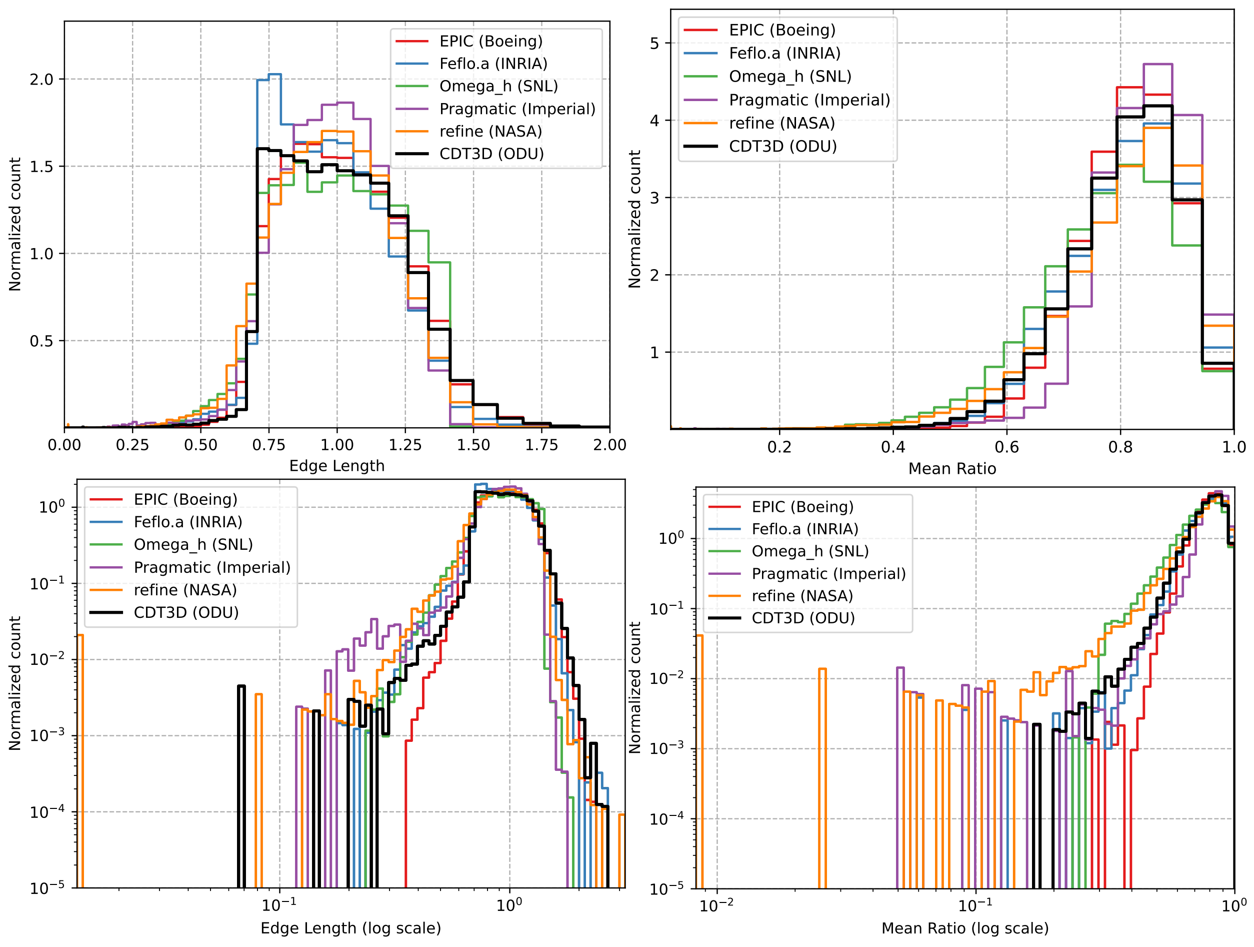


Fig. 29: Quality comparison of the `Polar-2` case in the log scale.

4.1.3 SOLUTION-BASED METRIC FIELDS ON PLANAR SURFACES

This section examines two cases, a delta wing with a solution-based metric field in laminar flow and a box-shaped domain with a solution-based metric field corresponding to a spherical blast problem.

4.1.3.1 Laminar flow over Delta wing

The next geometry, Figure 30, is a delta wing constructed of planar facets. A multiscale metric [5] is constructed based on the Mach field of this subsonic laminar flow. The initial mesh is adapted to a specified complexity of 50,000. Details of the verification of the delta wing/mesh adaptation process is provided by [199]. The multiscale metric is scaled to have a complexity of 500,000. Adapted meshes with approximately 1,000,000 vertices are expected. In the second set of data, the complexity is scaled to 10,000,000 which produces approximately 20,000,000 vertices, these mesh sizes are close to the maximum of the verification study performed in [199].

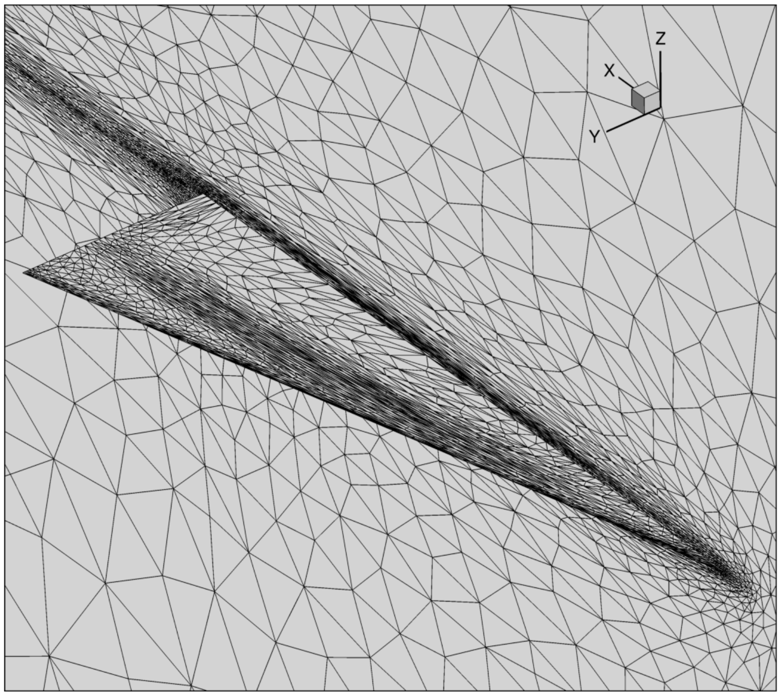


Fig. 30: Delta wing with multiscale metric in laminar flow, 50,000 complexity.

Strong Scaling: The speedup of *refine*, *CDT3D*, and *EPIC* when adapting the initial 50,000 complexity mesh to conform to a 500,000 complexity metric field as a function of the number of cores is shown in Figure 31. Similarly to Section 4.1.1, *Feflo.a* results are omitted for the lower complexity case. At high core numbers, both *EPIC* and *refine* exhibit improved scaling over the performance of the cube case due to the larger size of the initial mesh for the delta wing. At lower core counts, *refine* exhibits the best scaling while *CDT3D* falls between *EPIC* and *refine*. The superlinear scaling of *refine* is a result of the fact that *refine* has optimizations such as reordering of the nodes for cache efficiency within each partition, which have a computational complexity higher that $O(n)$ where n is the number of vertices in a partition. These optimizations favor configurations of many cores but cause significant overhead to the sequential performance. However, they allow *refine* to be within 10% of simulation time for inviscid simulations and 1% of the time for viscous simulations when coupled with FUN3D in a distributed memory setting, which is also its target configuration. Moreover, *refine* offers an "early termination" detection mechanism, which is turned off for this case since it produces a lot of noise in the results. The total time for *refine* for 1 core is 12,604 seconds and for 120 cores is 90 seconds while on a node of the same cluster *CDT3D* requires 794 seconds for 1 core and 29 seconds for 40.

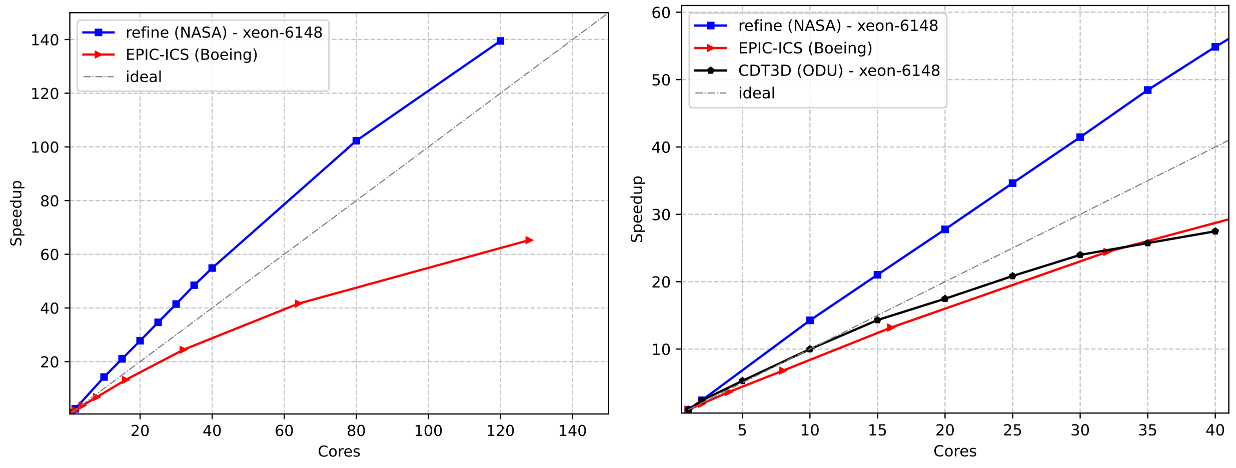


Fig. 31: Speedup data for the delta wing adapted from 50,000 complexity to 500,000 complexity. Left: Speedup data for up to 128 cores. Right: Zoom-in view of the data for up to 40 cores (Base case is the sequential time of each software.).

When the complexity of the target mesh is scaled to 10,000,000, *EPIC* retains the same scalability with the previous case as shown in Figure 32. *CDT3D* exhibits minor superlinear speedup up until 30 cores and linear between 30 and 40. The origin of the superlinear speedup is in part attributed to the increased throughput achieved by utilizing the cache memory shared among the hardware threads. *Feflo.a*'s scaling becomes constant at 8 cores, which is a result of the high startup cost of the decomposition method. More details for this cost appear in the weak scaling section below. *refine* results are omitted from the graphs as they exhibit the same issue as before with the sequential performance skewing the results to highly superlinear trends.

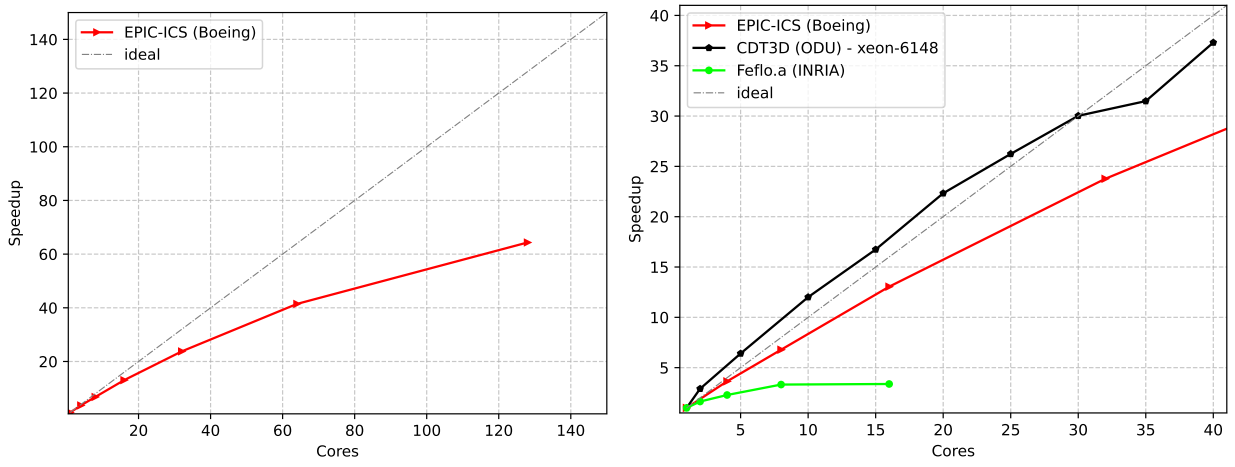


Fig. 32: Speedup data for the delta wing adapted from 50,000 complexity to 10,000,000 complexity. Left: Speedup data for up to 128 cores. Right: Zoom-in view of the data for up to 40 cores (Base case is the sequential time of each software.).

Weak Scaling: The presented timing information provides limited insight on the potential behavior of the parallel methods for extreme-scale current and emerging architectures. Amdahl's law predicts that the serial fraction of the code reduces the potential for parallel speedup as the number of cores grows. Traditionally, this issue is resolved by utilizing weak scaling, also known as *scaled speedup*, for evaluating the performance of a parallel mesh generation code by increasing the size of the mesh linearly to the number of cores, see for example [52]. However, this approach does not reflect the workflow of a simulation utilizing metric-based adaptation. A typical metric-based adaptation pass involves coarsening that decreases the number of elements and node movement, which can be crucial to improving the quality but depending on the algorithm may not affect the topology and thus the number of elements of the mesh. In an attempt to overcome these issues, we focus on the original definition of the scaled speedup, which is, that "the *problem size* scales with the number of processors" [117].

In this work, we define the problem size to be the complexity of the target metric rather than the number of elements in the mesh. Moreover, we do not use a constant step for increasing the complexity, since it is common for metric adaptive simulations to use a larger step for the first iteration [199]. Performing a fully adaptive simulation is reserved for the next section. Instead, we simulate each solver $\rightarrow$ error-estimation $\rightarrow$ metric-construction step by artificially scaling up the complexity of the input metric. In particular, we use as the input to the first "iteration" using 1 core, the 50,000 complexity metric field of this section. The input of the second iteration is created by increasing the complexity of the output mesh of the previous step by a constant amount using formula (13) of page 40 at every vertex of the mesh. The same procedure was applied for the rest of the steps. Table 29 presents the results.

refine and *CDT3D* retain an almost constant time of approximately 10,000 seconds and 1,000 seconds, respectively, as the problem size increases, which indicates a good weak scaling speedup. *Feflo.a* is the fastest among the methods even considering the difference between the machines that they were tested. On the other hand, it does not scale linearly as the size of the problem increases. Similarly to the previous case, the overhead of domain decomposition and distribution is a considerable amount for *Feflo.a* scaling from 6 seconds at 2 cores to 108 seconds at 40 cores which corresponds to 30% of the total running time.

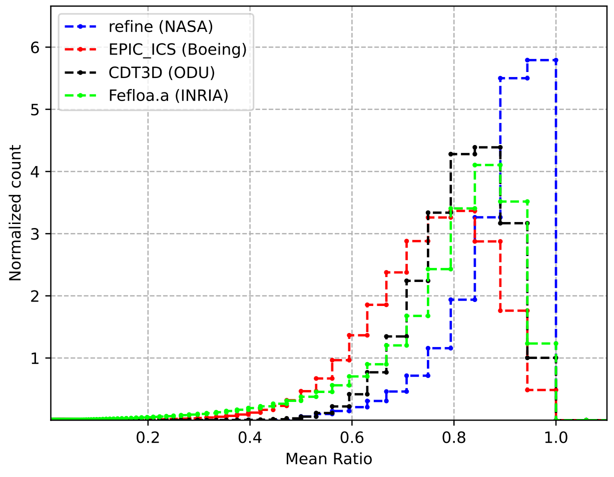
All three codes approach the expected number of elements with *refine* being closer. The difference in number of elements could is attributed in part to the different adaptation strategies as well as to the nature of the artificially scaled metric.

		refine		*CDT3D*		*Feflo.a*	
cores	complexity	# vertices	t_{e2e}	# vertices	t_{e2e}	# vertices	t_{e2e}
1	50k → 500k	927,390	9,256.41	871,402	1,211.51	835,123	64.83
2	500k → 1m	1,853,974	10,136.44	1,633,955	919.39	1,777,724	78.77
4	1m → 2m	3,694,187	10,482.89	3,271,567	1,055.28	3,516,645	101.28
8	2m → 4m	7,358,456	12,188.41	6,477,760	1,080.14	6,980,611	147.43
16	4m → 8m	14,694,593	13,915.35	12,831,874	1,190.72	13,511,085	193.31
32	8m → 16m	29,333,956	14,254.48	25,539,415	1,451.30	26,885,124	288.47
40	16m → 20m	35,767,590	10,469.66	30,539,328	1,509.98	33,498,896	340.82

TABLE 29: Weak scaling performance of *refine*, *CDT3D* and *Feflo.a* for complexities between 50,000 and 20,000,000. t_{e2e} corresponds to the end-to-end time in seconds.

Quality data: Returning to the 500,000 complexity target metric, metric conformity (characterized by element shape measure and edge length histograms of the generated meshes) is shown in Figures 33 and 34, respectively. On a linear scale, all methods appear to exhibit good overall quality. The log scale makes the differences more prevalent. *refine*'s mesh quality exhibits the best lower bound in the mean ratio measure and the distribution with the smallest deviation in the edge length measure.

Stability: The concepts of Stability and Reproducibility were introduced in Section 1.1. Adherence to these attributes is measured by evaluating the metric conformity of the same case with different numbers of cores. Histograms of edge length in the metric are evaluated for three codes for execution with different numbers of cores in Figure 35. *refine*, *CDT3D*, and *EPIC* show an almost perfect overlap of the histograms, but they do not produce the same mesh (i.e., they offer a weak form of the Reproducibility attribute). Producing metric conformity that is independent of the number of cores satisfies the requirement of Stability. The mean ratio histograms result in the same conclusion that metric conformity is independent of the number of cores for these tools and the mean ratio plot is omitted for brevity.



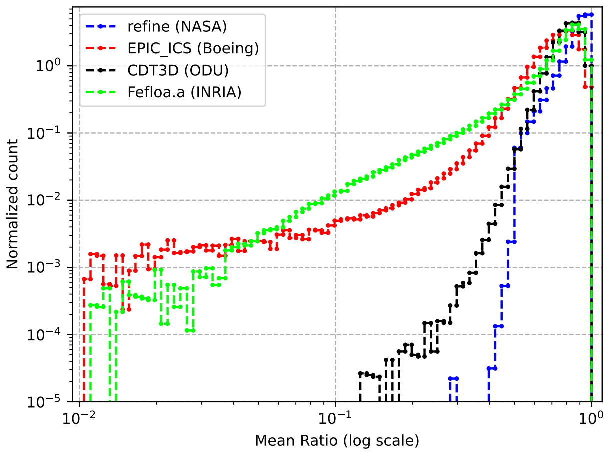
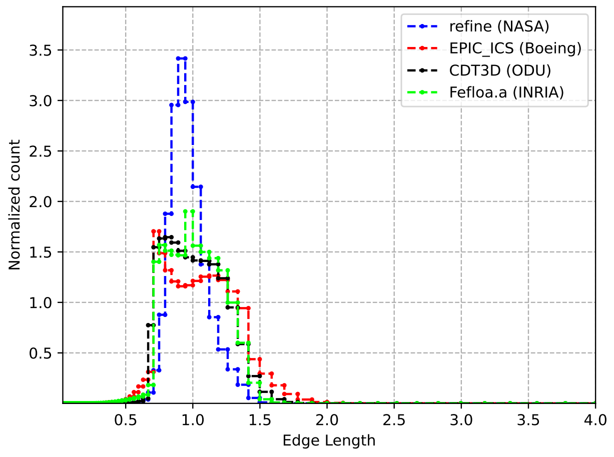


Fig. 33: Comparison of the mean ratio of the generated meshes for the delta wing 500,000 complexity case in linear and logarithmic scales.



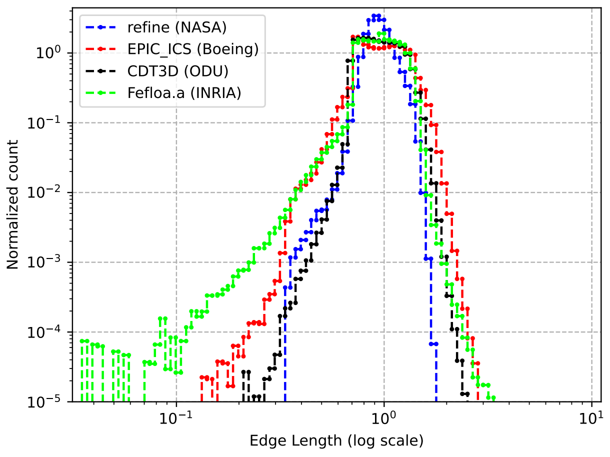


Fig. 34: Comparison of the edge lengths of the generated meshes for the delta wing 500,000 complexity case in linear and logarithmic scales.

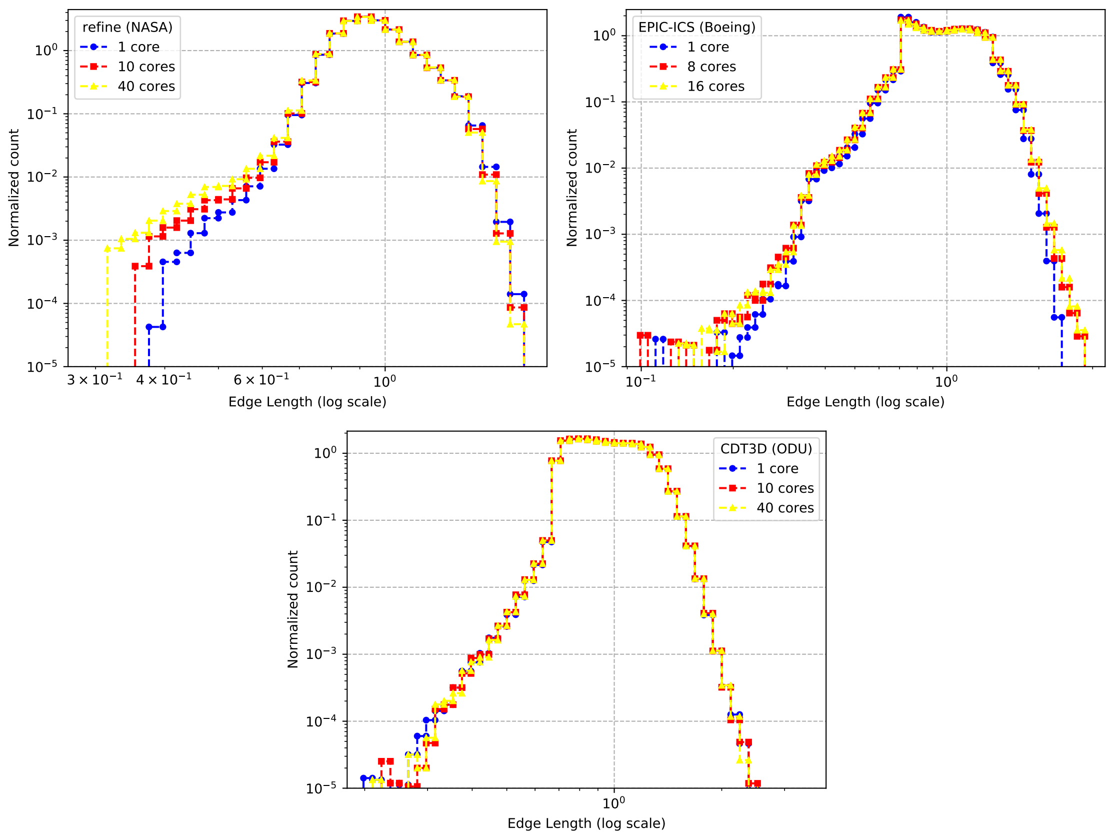


Fig. 35: Stability data for the delta wing 50,000 to 500,000 complexity case using *refine*, *EPIC* and *CDT3D*.

4.1.3.2 Spherical Blast

In order to complement the previous case where mesh refinement is the main operation, the following case focuses mainly on coarsening operations. It corresponds to the numerical solution (at one time step) of a spherical blast problem [159]. The target metric complexity is 49,013, which corresponds to about 98,000 vertices in the final mesh. As initial input, a uniform tetrahedral mesh of 1,900,000 vertices is provided. The adapted mesh is shown in Figure 36.

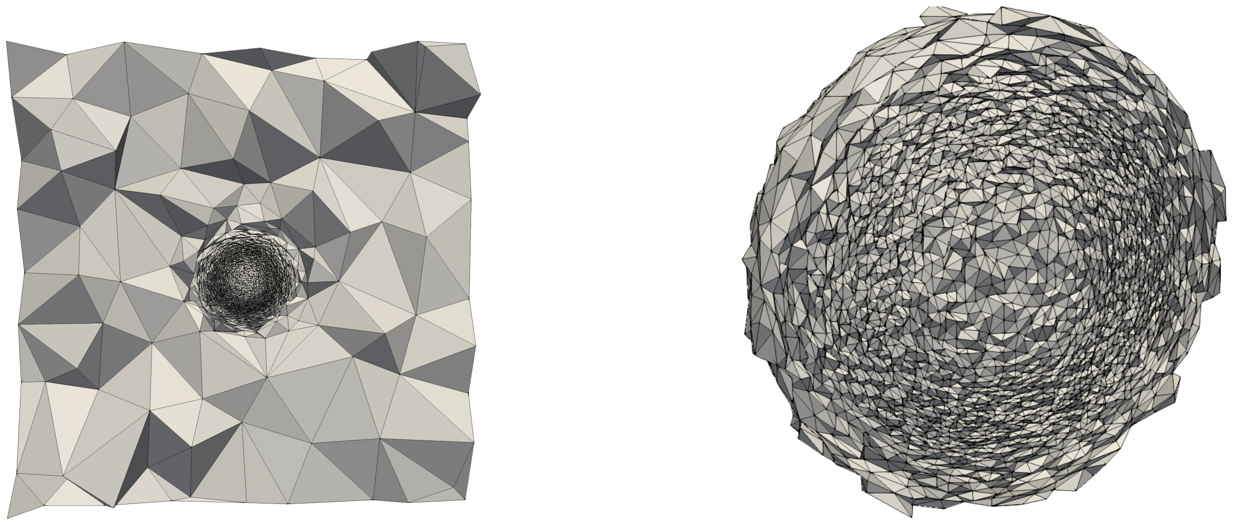


Fig. 36: Adapted mesh of the spherical blast case. Left: Cross-cut of the domain. Right: Zoom-in of the extracted part of the core.

For *refine* the number of sweeps was fixed and set to 40. This value was selected because it allowed all cores to complete the adaptation while creating less noise in the timings since no case could exit earlier, thus skewing the results. *CDT3D* was configured with a higher collapse limit for the mesh preprocessing step (see Figure 7). This configuration was selected because it gives more flexibility in the subsequent refining step and yields better quality in the final mesh. A similar approach is used in Ref. [160] for generating an *almost empty* mesh and subsequently a metric-orthogonal mesh.

Figure 37 depicts the strong scaling performance of *Feflo.a*, *CDT3D* and *refine*. *refine* exhibits superlinear scalability for low number of cores (< 80) and almost linear for the rest of the cases. In contrast, the speedup of *CDT3D* stagnates after 20 cores, which indicates that there is not enough work to keep the additional cores busy. The same issue arises in *Feflo.a* with the speedup stagnating at an earlier stage. Table 30 presents the total time for this experiment in a shared memory setting (40 cores) and for *refine* we include distributed memory results (up to 400 cores).

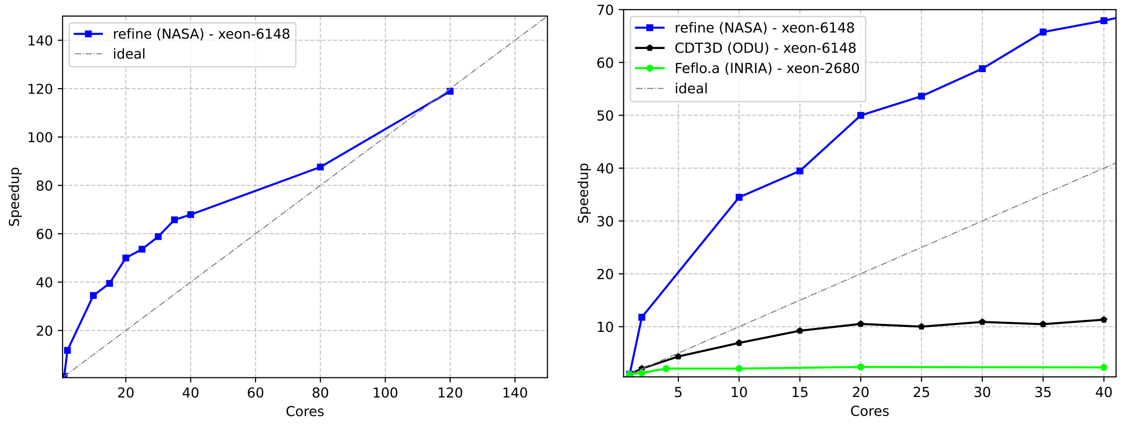


Fig. 37: Speedup data for blast case. Left: Speedup data for the blast case for up to 120 cores. Right: Zoom-in view of the data for up to 40 cores (Base case is the sequential time of each software.).

A direct comparison of the times is not possible because as it is mentioned in the beginning of the chapter the results of *refine* and *CDT3D* were collected on the same machine, while for *Feflo.a* a different machine was used. Still, the table reveals that *Feflo.a* is faster than *CDT3D* and *refine* using one core. However, using more than 10 cores *CDT3D* is 50% faster and on 40 cores is more than two times faster than *Feflo.a*. On the other hand, *refine* achieves a speedup of 67 on 40 cores and 328 on 400 cores; the superlinear speedup occurs due to the reasons discussed in the delta wing case. The breakdown of the running time of *Feflo.a* in Table 31 reveals that the main inefficiency is the subdomain creation step which

# cores	*Feflo.a* (s)	*CDT3D* (s)	*refine* (s)
1	62.82	152.41	62574.51
2	50.57	73.76	5311.00
10	30.41	22.02	1814.63
20	26.45	14.49	1252.03
40	27.42	13.46	921.39
200	-	-	332.17
400	-	-	190.36

TABLE 30: Total running times of *Feflo.a*, *CDT3D* and *refine* for the blast case.

takes a constant amount of time for all five runs. Moreover, *Feflo.a* utilizes a *cavity-based* collapse operation [159] which always results in an edge collapse, whereas the standard collapse algorithm utilized by *CDT3D* and *refine* rejects a fair amount of configurations which revisits in a subsequent step.

# cores	Total Time	Subdomain creation	Mesh Adaptation
1	62.82	-	62.82
2	50.57	11.32	38.66
10	30.41	11.29	17.13
20	26.45	11.22	12.22
40	27.42	11.26	11.49

TABLE 31: Breakdown of the total running time for *Feflo.a*.

The quality of the generated meshes is in accordance with the results of the cases discussed earlier. *refine* achieves the smallest variance in edge lengths and a mean length of 0.9. *Feflo.a* follows a similar distribution with a tighter lower limit. *CDT3D* delivers a wider distribution and few edges between 2 and 4 as well as a small number of edges below 0.1. For the mean ratio, *refine* delivers a mesh with minimum mean ratio quality of 0.3, for *Feflo.a* the minimum is 0.2, while for *CDT3D* it is 0.1.

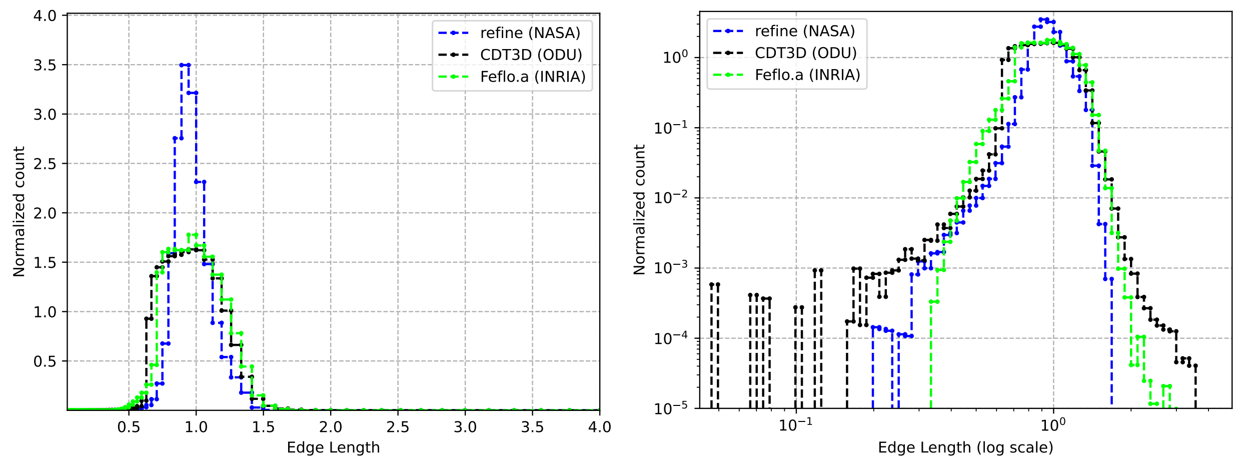


Fig. 38: Comparison of the edge lengths of the generated meshes for the spherical blast case in linear and logarithmic scales.

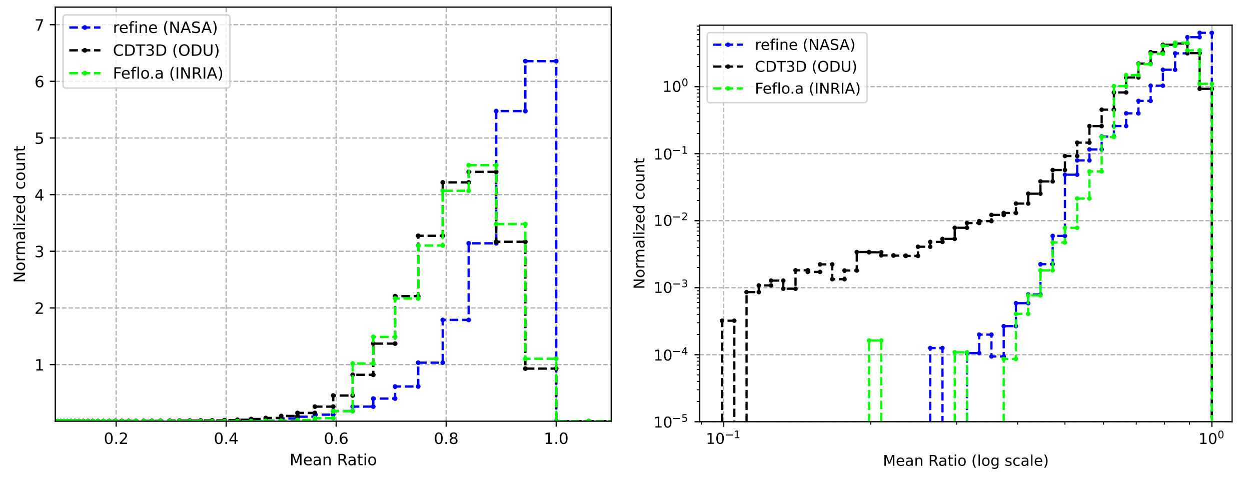


Fig. 39: Comparison of the mean ratio of the generated meshes for the spherical blast case in linear and logarithmic scales.

4.1.4 SOLUTION-BASED METRIC FIELDS ON CURVED SURFACES

In this section, we evaluate our method on two cases that utilize CAD data and fixed complexity. We compare our method with results presented in [201]. In particular, we focus on the fixed complexity cases that utilize a `hemisphere-cylinder` and the `onera-m6` geometry. The input B-rep models were drawn from the UGAWG's Github repository[25]. Figure 40 depicts the input models as well as a decomposition of the B-rep models to its constituent surfaces. The dataset of [201] contains five mesh adaptation codes *EPIC*, *refine*, *Feflo.a*, *Pragmatic*, and *Omega_h*. Descriptions of all the methods appear in Section 2.1. For *EPIC* we use only the data generated using *EPIC-ICSM*; which corresponds to *EPIC* with the widest suite of mesh operations enabled (insertion, collapse, swaps and node movement) since, it produces the highest quality among the configurations of *EPIC* presented in [201].

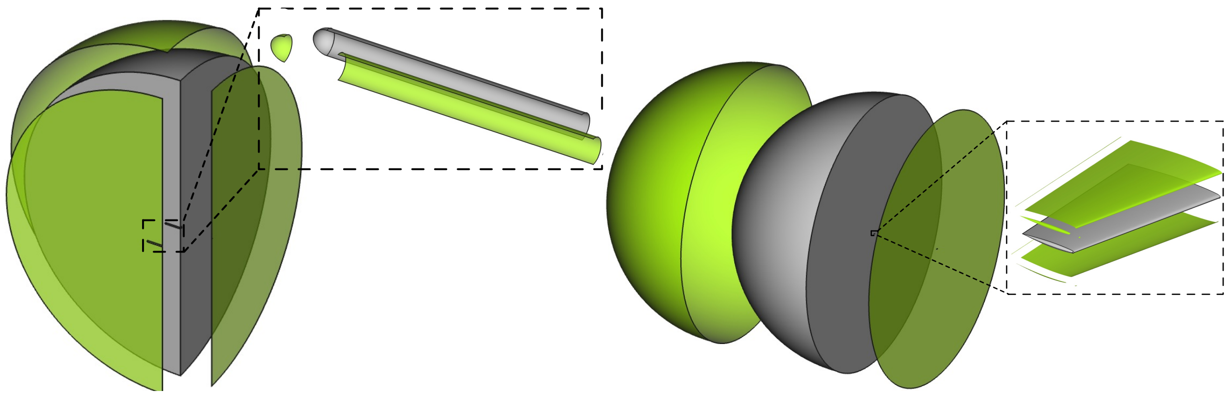


Fig. 40: Decomposition of the the input B-rep models for the `hemisphere-cylinder-fixed` and the `onera-m6` cases.

It should be noted that the graphs presented here have been generated using the meshes available at the UGAWG's Github repository. For the mean ratio measure we used the definition of equation (10) of page 39 instead of the one presented in [201], so the graphs we present and the ones in the original paper may differ slightly. This choice was done to

[25]`https://github.com/UGAWG/solution-adapt-cases` (Accessed 2021-05-31).

conform with the rest of the data in this thesis but also because the UGAWG have since then switched to the same formula for mean ratio in their latest papers.

4.1.4.1 Hemisphere-Cylinder with fixed target complexity

The hemisphere cylinder case is designed as a model for a turbulent flow over a smooth body of revolution in three dimensions. It corresponds to earlier wind tunnel studies [127] and it is part of NASA's Turbulence Modeling Resource[26]. The flow conditions are 0° angle of attack, 0.6 Mach number and 0.35 million Reynolds number based on the diameter of the cylinder. The metric field used in this case is generated by FUN3D-FV and *refine* by applying the multiscale metric construction scheme (see Section 2.2.3.3) on the local Mach number of the flow. Two target complexities are utilized at $30,000$ and $100,000$. The input (mesh, metric) pairs were drawn from UGAWG's Github repository[27]. Figure 41 depicts the adapted meshes for the two metric complexities. The output meshes have $57,114$ and $191,530$ vertices respectively.

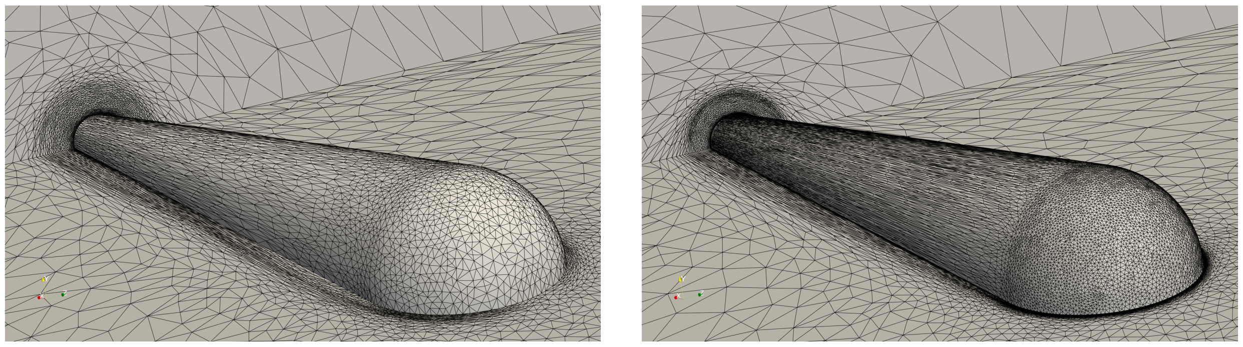


Fig. 41: Adapted meshes by *CDT3D* for the `hemisphere-cylinder` at $30,000$ and $100,000$ target complexity respectively.

Figure 42 presents quality results for the $30,000$ complexity case. The edge distribution of *CDT3D* is centered around 1 and in general follows the trends of the rest of the mesh

[26]`https://turbmodels.larc.nasa.gov` (Accessed 2021-05-31).
[27]`https://github.com/UGAWG/solution-adapt-results` (Accessed 2021-05-31).

adaptation methods. The use of the log scale for the Edge distribution reveals that *CDT3D* generates the shortest edge. On the other hand, the longest edge generated is below that of *EPIC* and *Feflo.a*. In terms of mean ratio *CDT3D*'s minimum value is around 0.1 which is higher that the minimum value for *Pragmatic* and *Feflo.a*, and very close to the values of *refine* and *EPIC*.

Figure 43 presents quality results for the 100, 000 complexity case. The edge distribution of *CDT3D* is slightly skewed to left of the ideal value 1 indicating a small amount of over-refinement. Still, it is within the trends of the rest of the mesh adaptation methods. The use of the log scale for the edge distribution reveals that *CDT3D* generates the shortest edge with *refine* and *Feflo.a* being very close. On the other end of the spectrum, it generates a maximum length smaller than *Feflo.a* and close to the rest of the methods. In terms of mean ratio, *CDT3D*'s minimum value is around 0.1 which is higher that the minimum value for *Pragmatic* and *Feflo.a* and very close to the values of *refine* and *EPIC*.

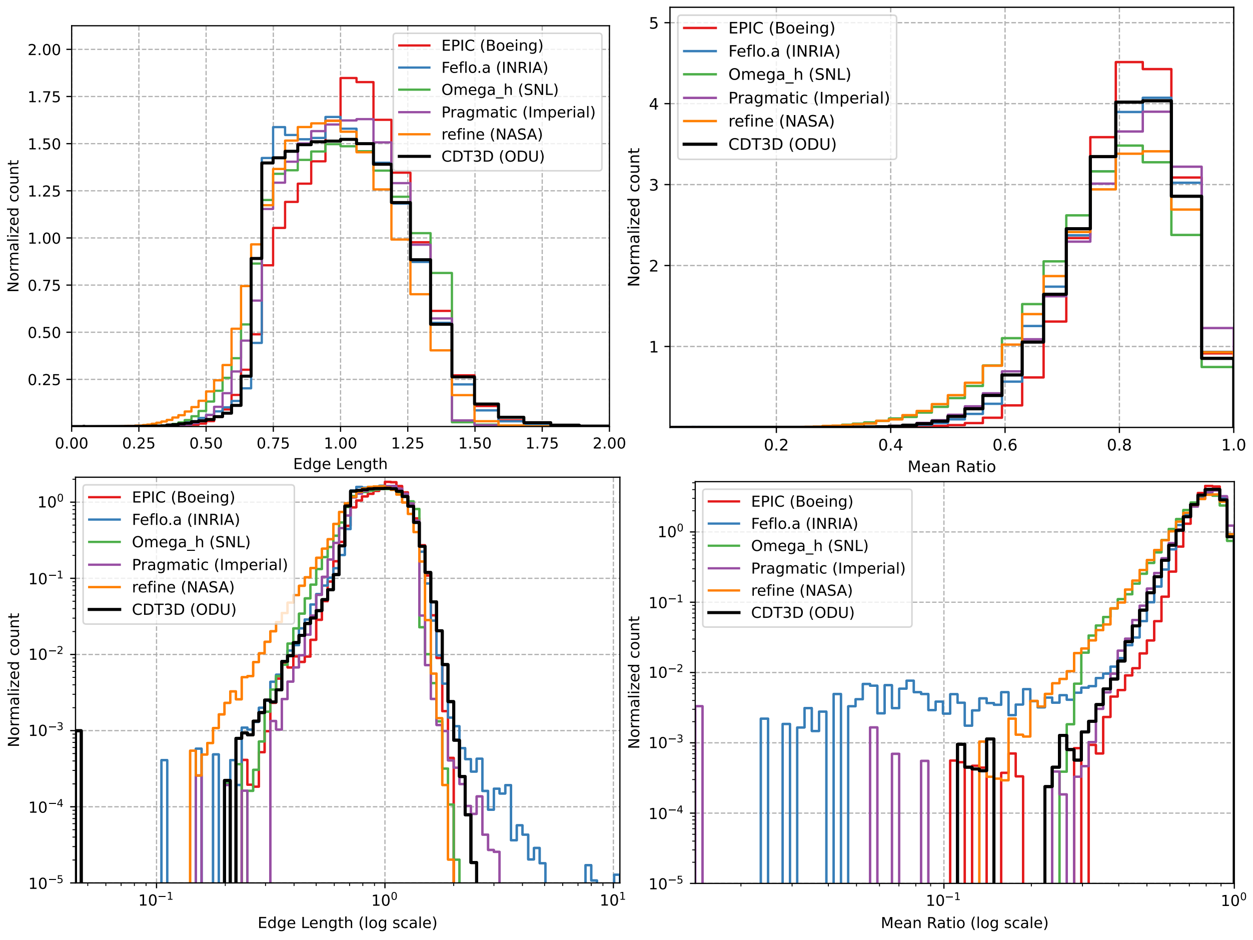


Fig. 42: Mean ratio and Edge Length quality measures for the `hemisphere-cylinder-fixed` case at 30,000 complexity. The Edge Length in linear scale is cropped to the $(0, 2)$ range. The second row uses a log scale make differences more prevalent.

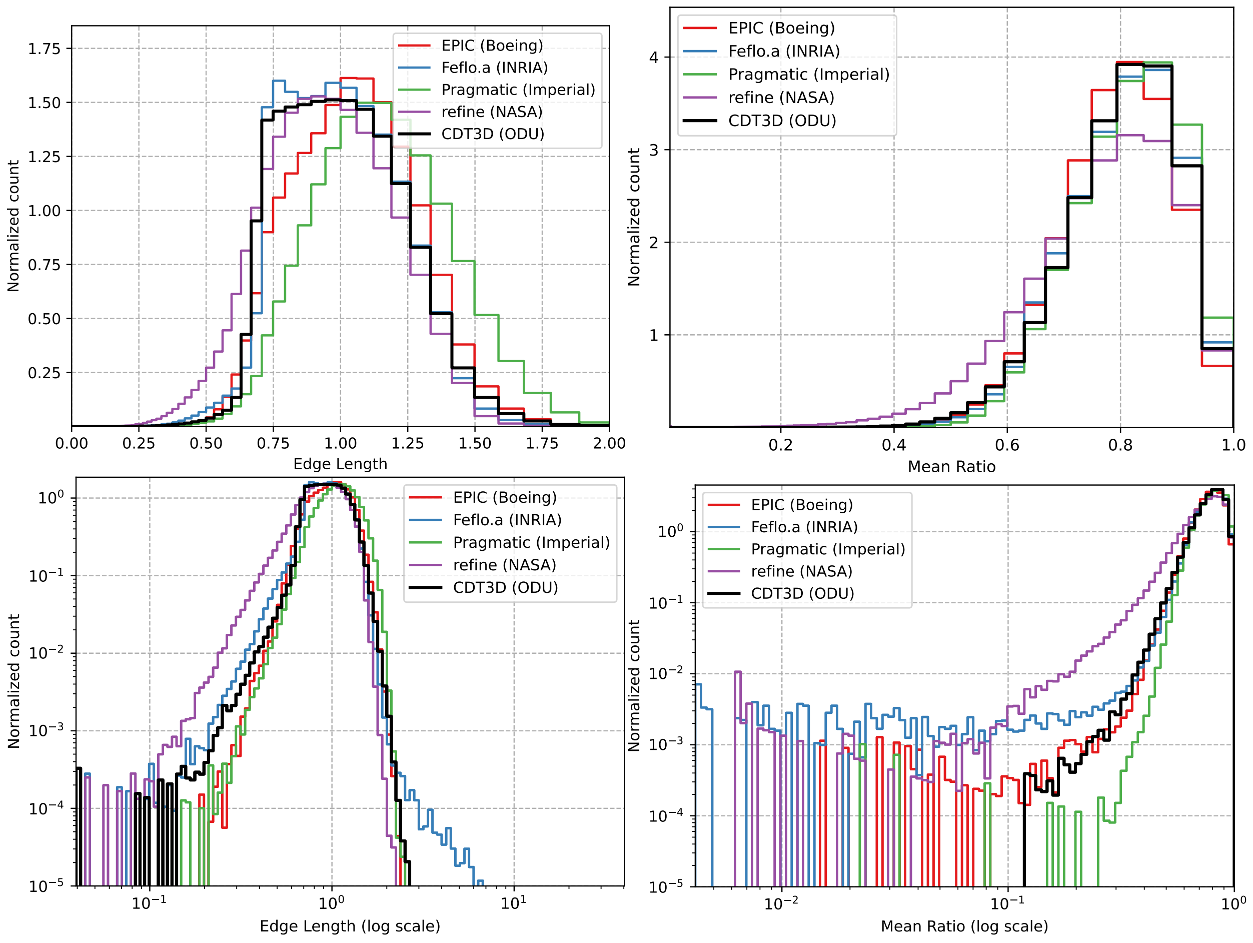


Fig. 43: Mean ratio and Edge Length quality measures for the `hemisphere-cylinder-fixed` case at 100,000 complexity. The Edge Length in linear scale is cropped to the (0,2) range. The second row uses a log scale make differences more prevalent.

4.1.4.2 ONERA M6 fixed complexity

The onera-m6 case is derived from the ONERA M6 experiment [224] and it is a well documented case in the CFD literature. It is part of NASA's Turbulence Modeling Resource[28]. The flow conditions are 3.06° angle of attack, 0.84 Mach number and 14.6 million Reynolds number based on the root chord. The metric field used in this case is generated by FUN3D-FV and *refine* applying the multiscale metric construction scheme (see Section 2.2.3.3) on the local Mach number of the flow. Two target complexities are utilized at $30,000$ and $100,000$. The input (mesh, metric), pairs were drawn from UGAWG's Github repository[29]. Figure 44 depicts the adapted meshes for the two metric complexities. The output meshes have $63,285$ and $191,089$ vertices respectively.

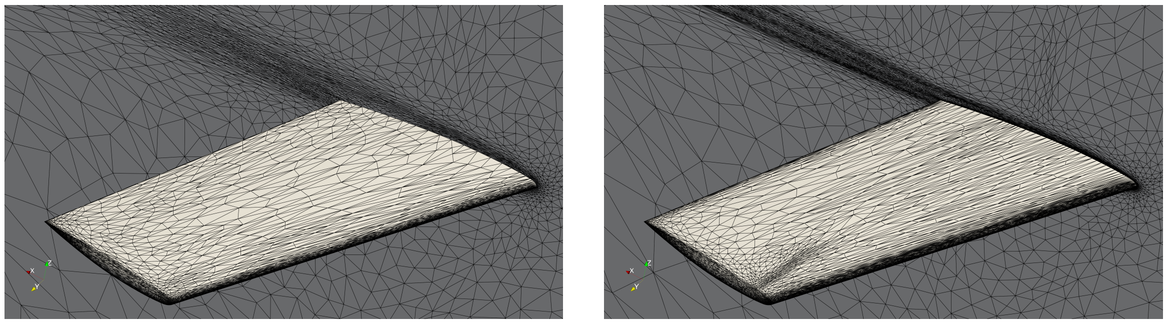


Fig. 44: Adapted meshes by *CDT3D* for the onera-m6 case at $30,000$ and $100,000$ target complexity respectively.

Figure 45 depicts the quality of the adapted mesh at $30,000$ metric complexity. In the Edge Length measure, *CDT3D* produces a histogram slightly skewed towards the left of 1 but still within the spectrum of the rest of the methods. The log scale reveals that shortest and longest edge of *CDT3D* is close to the values produced by *Feflo.a*. In terms of the mean ratio measure, the linear scale depicts similar trends. The log scale on the other hand, reveals that *CDT3D* produces a mesh with minimum value above 0.1. This comes second

[28]`https://turbmodels.larc.nasa.gov/onerawingnumerics_val.html` (Accessed 2021-05-31).
[29]`https://github.com/UGAWG/solution-adapt-results` (Accessed 2021-05-31).

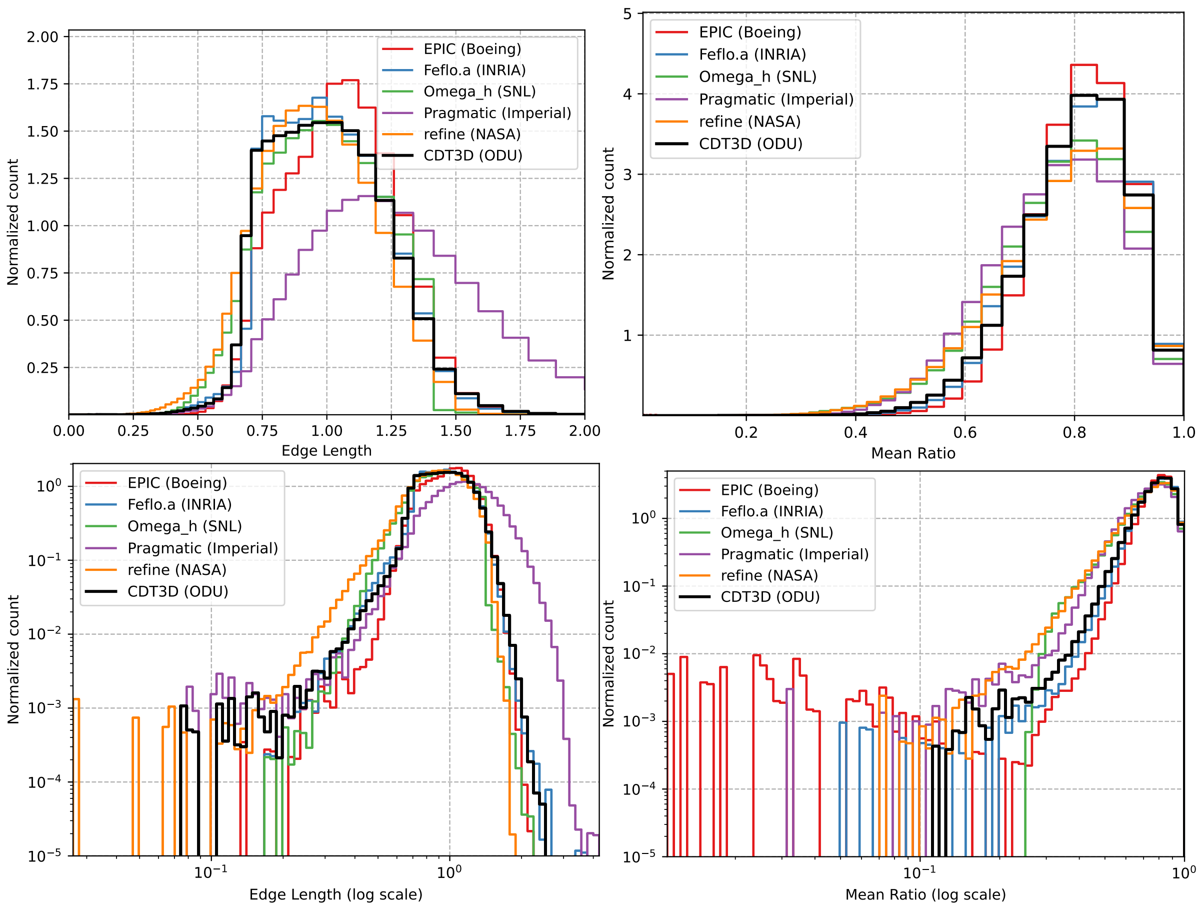


Fig. 45: Mean ratio and Edge Length quality measures for the `onera-m6` case at 30,000 complexity. The second row uses a log scale to highlight the differences better.

only to *Omega_h*'s result.

Figure 46 depicts the quality of the adapted mesh at $100,000$ metric complexity. The results are similar to the lower complexity case. The edge length distributions are closer to each other in both the linear and logarithmic scale. In terms of the mean ratio measure, the logarithmic scale reveals that *CDT3D* produces the highest minimum value which is above 0.1.

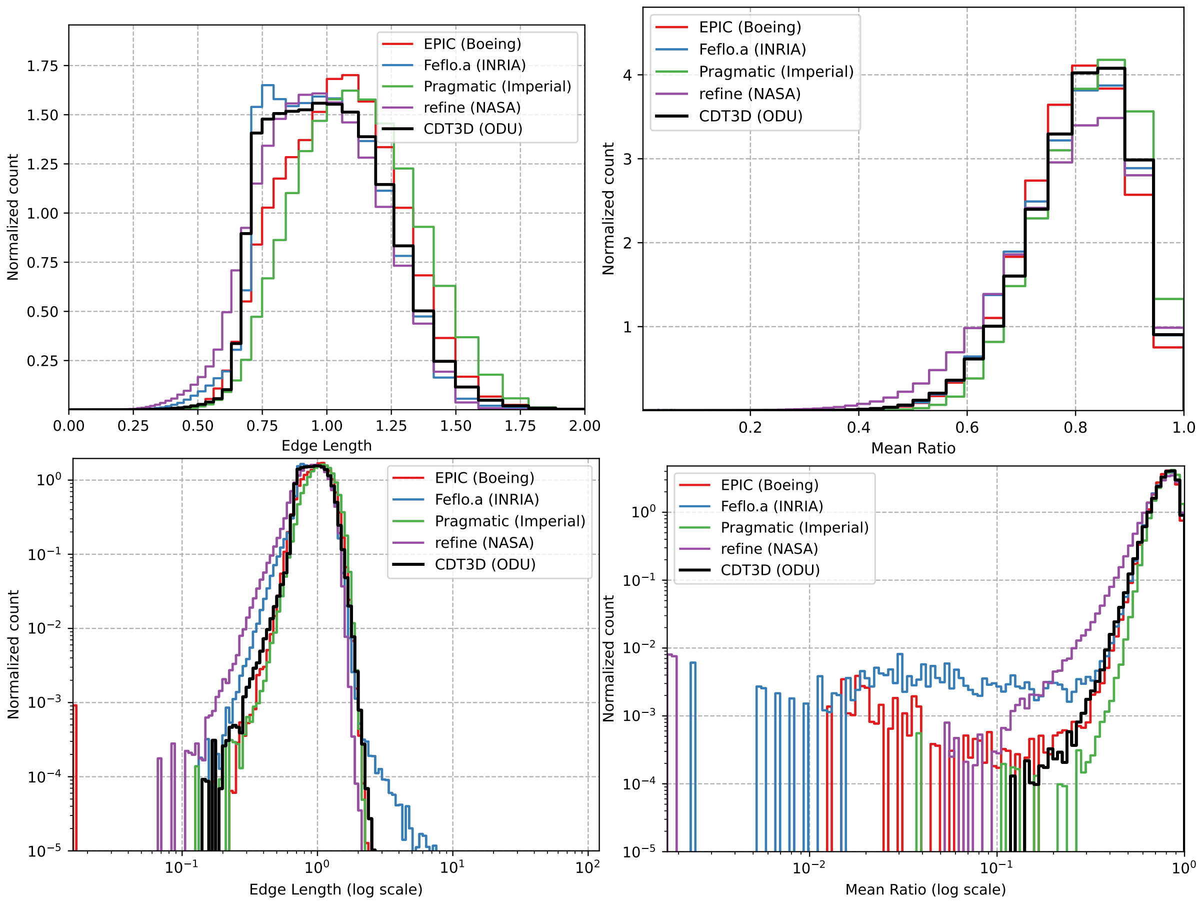


Fig. 46: Mean ratio and Edge Length quality measures for the `onera-m6` case at 100,000 complexity. The second row uses a log scale to highlight the differences better.

4.2 MESH ADAPTATION WITHIN AN ADAPTIVE PIPELINE

Evaluating our approach with fixed target meshes not only simplifies the testing process, but also gives a straightforward way to compare our method with other approaches based on quantitative and qualitative metrics. However, as mentioned in the introduction, the goal of generating meshes is to provide a domain discretization for the Finite-element solver which is the ultimate consumer of meshes. In this section, we build a software pipeline around our mesh adaptation method in order to evaluate its effectiveness for adaptive simulations.

In order to meet the ever-evolving and growing demands of the CFD community, a simulation pipeline should be able to integrate a plethora of different tools. The T-infinity project [191] demonstrates a series of different use-cases where a high-level Python interface can be used to build sophisticated pipelines. In this work, we focus on a single pipeline depicted in Figure 47 which is pertinent to mesh adaptation.

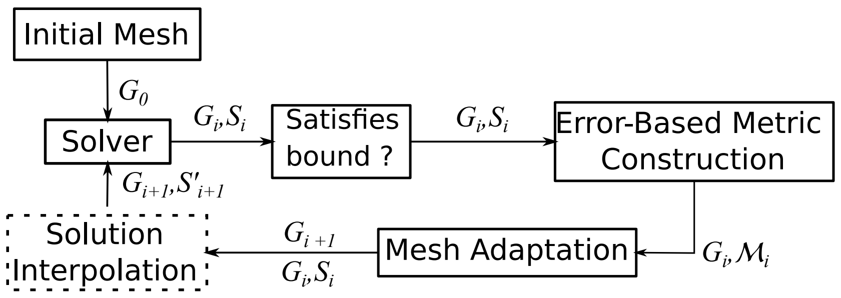


Fig. 47: Mesh Adaptation pipeline. G_i denotes the mesh at the i-th iteration. S_i, S'_i the solver solution and the interpolated solution at the vertices of G_i, respectively. $\mathcal{M}_i$ corresponds to the metric field associated with the vertices of G_i and derived from S_i.

In Figure 47, the process is initialized with a (usually coarse) mesh G_0 that captures all the geometrical features of the input model at some user-defined accuracy. The solver then evaluates a discrete solution of the problem of interest and stores it in each mesh element. For simplicity, we assume that the solver in this case is vertex-based and the solution is stored at each vertex of the mesh S_i. The next block captures a user-defined condition that controls the exit of the iterative process. It can be based on some target simulation quantity or on the total number of iterations of the adaptive loop. The Metric Construction step creates a metric field $\mathcal{M}_i$ at each vertex of G_i using S_i that drives the adaptation process. Mesh Adaptation modifies the mesh based on the provided metric field and generates a new mesh G_{i+1}. Optionally, one can interpolate the solution of the previous iteration to the new mesh thus producing S'_{i+1}. This step allows the solver to restart the calculation from a state closer to the converged solution instead of starting from the freestream conditions which is the default. Finally, the new mesh (and optionally the interpolated solution S'_{i+1}) are passed to the solver for the next iteration of the loop.

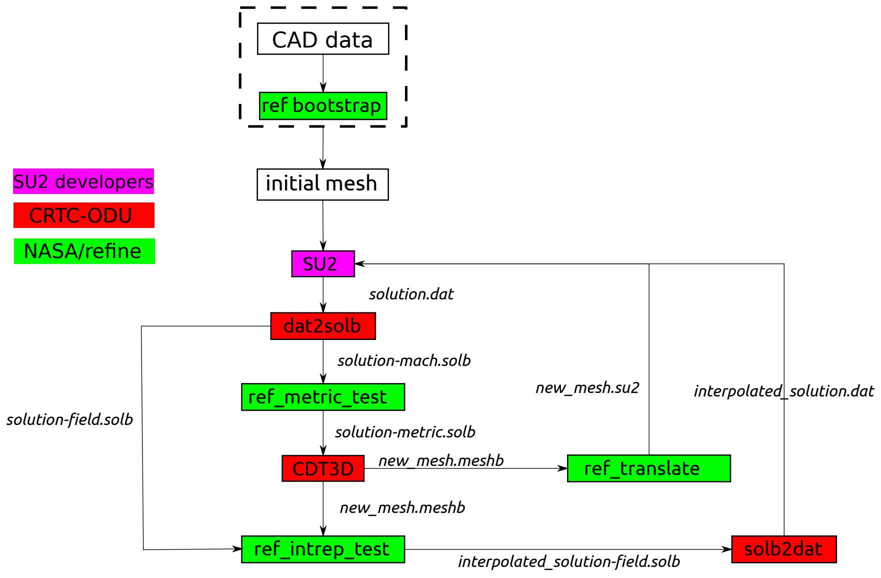


Fig. 48: Software pipeline utilized in the adaptive pipelines of this study.

The corresponding software pipeline can be seen in Figure 48. For the cases of this study, the input volume mesh is either given or created out of a CAD file using `ref bootstrap` which is part of the *refine* mesh mechanics suite [198]. `ref bootstrap` uses the EGADS [121] kernel in order to generate an initial surface triangulation of the input

CAD file. The surface mesh is then adapted based on the curvature and other geometrical features. Adapting the surface in absence of a volume mesh gives greater flexibility since the software is not constrained by the requirement of conformity to a volume after each operation. A volume mesh is then generated using an external tool such as TetGen [235] or AFLR [173] and finally the volume mesh is adapted based on a metric field derived by the geometrical features of the CAD input. `SU2` will then produce a solution file that holds values of the discrete solution at each vertex of the input mesh. `dat2solb` is used to convert the solution to a libMeshb-compatible file [174]. The extracted Mach field (`solution-mach.solb`) is then passed to `ref_metric_test` that creates a multiscale metric field based on it (`solution-metric.solb`). The multiscale metric field can be optionally intersected with a curvature- and feature-based metric built based on the geometrical features of the input model. *CDT3D* will then use the metric field along with the mesh used by `SU2` to generate an adapted mesh (`new_mesh.meshb`). If solution interpolation is utilized, we pass the new mesh along with a `.solb` version of the `SU2` solution to `ref_intrep_test` which we then convert using `solb2dat` to an SU2-compatible file (`interpolated_solution.dat`). The values of the previous solution are interpolated using linear interpolation. Finally, the adapted mesh is passed to `SU2` after being converted to a `.su2` mesh file along with the interpolated solution if this was generated.

It should be noted that the metric field can be built using any solution variable besides the local Mach number[30]. However, the use of the local Mach number is favored in the literature, since it provides a "compound" scalar variable that varies in most flow regions, thus allowing to capture most of the flow features [34, 118].

For the rest of this section, we present results utilizing the above pipeline for four different cases of increasing difficulty. First, an analytic field where the solver is replaced with an analytic function in order to verify the rest of the components of the pipeline. Then, in Section 4.2.2 we apply the pipeline on a flow over a model with planar faces. The goal of this test case is to verify our implementation in absence of curved surfaces. Section 4.2.3 presents results utilizing a flow over a simple curved model. Finally, in Section 4.2.4 we present results on one of NASA's High-lift cases.

Since the solver, a major part of the pipeline, is an external and sophisticated project, fine-tuning of its parameters and detailed convergence and error-analysis is outside the scope of this thesis. Instead, the goal of this section is to show capability of our method to function as part of an adaptive pipeline.

[30]The local Mach number is defined as the ratio of the local flow speed over the local speed of sound.

4.2.1 ANALYTIC SCALAR FIELDS

The adaptation pipeline of Figure 47 consists of many parts and the errors in each component can have accumulative and unpredictable consequences in the final calculation. In an effort to mitigate these issues, we first test *CDT3D* by replacing the CFD solver with analytic metric fields. In particular, instead of solving a flow problem at each iteration, we evaluate an analytic function at the vertices of the mesh. The adaptive iterations will create a mesh that is expected to drive the interpolation error down. For this test, we will be using the three analytic cases described in [100] and implemented in the *refine* suite. The multiscale metric implementation of *refine* has been combined with several mesh adaptation tools and verified separately in [100] and thus, we will only focus on the verification of the adaptation procedure in *CDT3D*.

For each of the three analytic scalar fields ((21),(22),(23)) the adaptation pipeline starts with a uniform tetrahedral mesh of the unit cube domain $[0,1]\times[0,1]\times[0,1]$ with 64 vertices. In contrast to the analytic metric fields of Section 4.1 that provide the metric value directly, for the cases of this section, we derive the metric field from a scalar field. In particular, in each iteration a multiscale metric field is computed using $F(x,y,z)$ as scalar field. The metric is computed in the 2-norm and the gradation value is set to 3. The metric is then passed to our method along with the mesh of the previous iteration.

$$\texttt{sinfun3} \quad := F(x,y,z) = \begin{cases} 0.1\sin(50xyz) \text{ if } xyz \leq \frac{-1}{50}\pi \\ \sin(50xyz) \text{ if } xyz \leq \frac{2}{50}\pi \\ 0.1\sin(50xyz) \text{ else} \end{cases},$$

$$\text{where } xyz = (x-0.4)(y-0.4)(z-0.4) \tag{21}$$

$$\texttt{tanh3} \quad := F(x,y,z) = \tanh\left((x+1.3)^{20}(y-0.3)^{9}z)\right) \tag{22}$$

$$\texttt{sinatan3} \quad := F(x,y,z) = 0.1\sin(50xz) + \tan^{-1}\left(0.1/(\sin(5y)-2xz)\right) \tag{23}$$

For each field, 90 adaptive iterations are performed with the complexity increased at every 10 iterations. The convergence plots in Figure 49 show the interpolation error of the last 5 iterations at each complexity with respect to the finest generated mesh. Since, the multiscale metric approximates linear interpolation error via a Hessian reconstruction, all results are expected to exhibit second order convergence rate. For comparison, the same adaptation procedure was performed using *refine*.

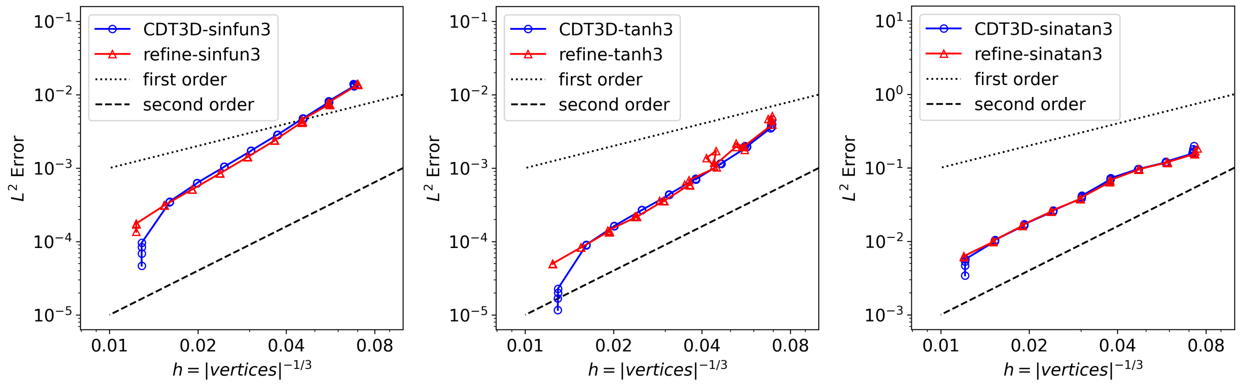


Fig. 49: Convergence rates for *CDT3D* and *refine* for the three scalar fields.

Figure 49 indicates that the convergence rate of *CDT3D* matches closely the rate of *refine* and they both exhibit 2nd-order convergence. Figure 50 presents the adapted meshes. *CDT3D* is able to recover the features of the scalar fields at both small and large scales.

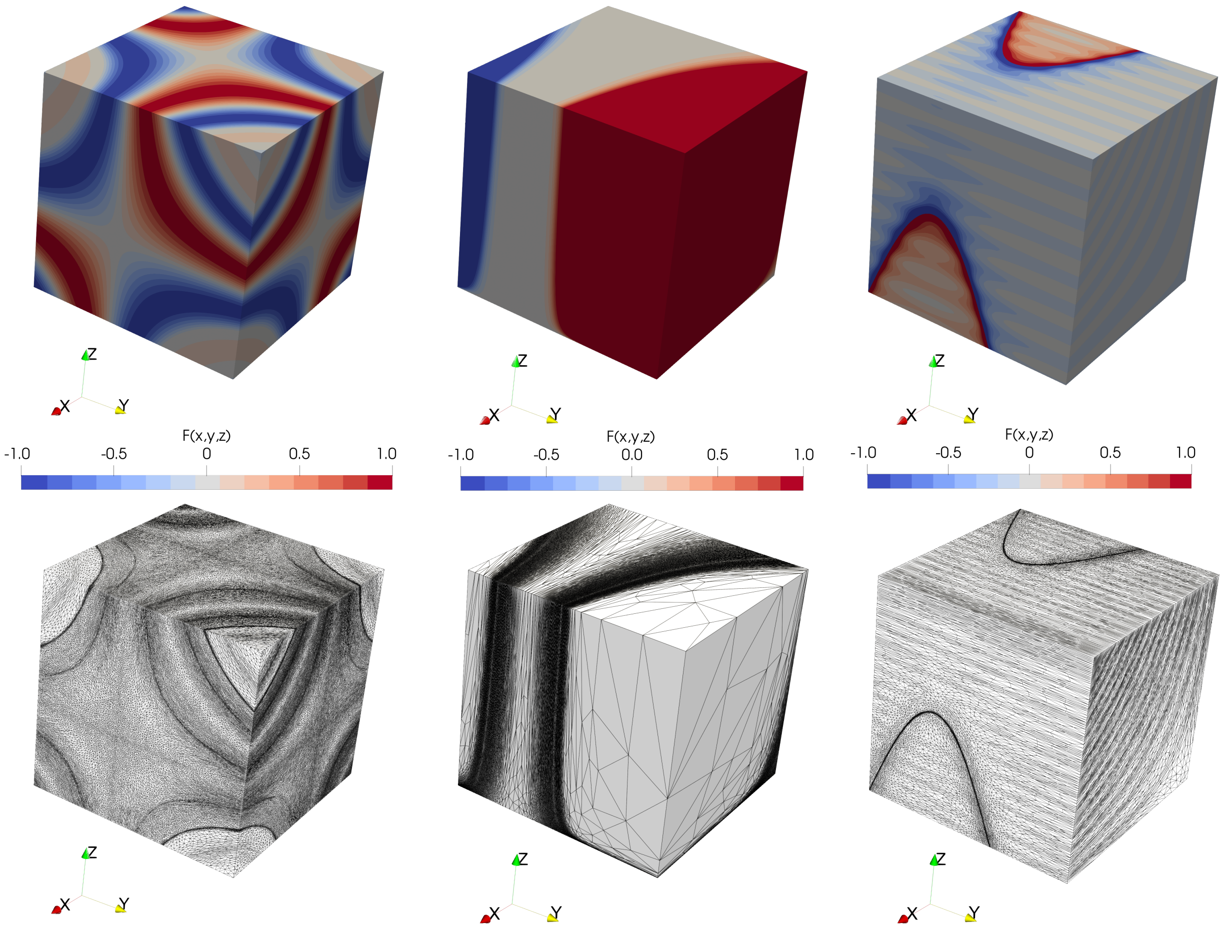


Fig. 50: Adapted meshes for the three fields. Top: `sinfun3`, `tanh3` and `sinatan3` fields. Bottom: Corresponding *CDT3D* adapted meshes at 256,000 target complexity.

4.2.2 LAMINAR SUBSONIC FLOW OVER A DELTA WING

For the next case, *CDT3D* is coupled within an adaptive pipeline that includes a CFD solver. The input geometry is a delta wing with planar faces. The 3D delta wing simulation conditions have been set so that they match the case used in the first three High-Order Workshops [256]. This case is well studied in the literature and it is preferred due its simple geometry and yet non-trivial flow features. Adaptive results in terms of mutliblock meshes appear in [148], verification results for the multiscale, MOESS and output-based metrics appear in [100] and [17].

The freestream conditions are 0.3 Mach, 4000 Reynolds number based on a unit root chord length and 12.5° angle of attack. The wing surface is modeled as an isothermal no-slip boundary with the freestream temperature equal to 273.15K. The Prandtl number is 0.72 and the viscosity is assumed constant. SU2 is configured with an initial CFL number of 1 and a final value of 5 with a ramping of 1.001. As linear solver FGMRES is used with the ILU preconditioner. The error for the linear solver is set to 10^{-10} and the number of the linear iterations to 10. The Roe convective scheme is used with MUSCL reconstruction and the Van Albada edge limiter[31].

For each iteration except the first, we also supplied an interpolated solution on the new mesh based on the solution of the previous iteration. The metric is constructed based on the local Mach field of the solution and the metric gradation value is set to 2.0. The complexity of the metric is doubled every 5 iterations. The solution-based metric is intersected also with a curvature- and feature-based metric built based on the geometrical features of the wing. Although the geometry in this case is planar, the CAD kernel is utilized to validate its implementation and coupling with *CDT3D*. We considered 7 metric complexity values for this study: $[50\,000, 100\,000, 200\,000, 400\,000, 800\,000, 1\,600\,000, 3\,200\,000]$.

Figure 51 depicts the initial surface mesh of the delta wing as well as adapted meshes at $100,000$ and $800,000$ complexity respectively. The initial mesh has 901 vertices, the middle corresponds to the 15th iteration with $377,569$ vertices and the last corresponds to the 25th iteration and has $1,467,922$ vertices. Figure 52 depicts streamlines and contour slices of the final solution.

[31]We would like to thank Jayant Mukhopadhaya from Stanford University for his help configuring SU2 for this case.

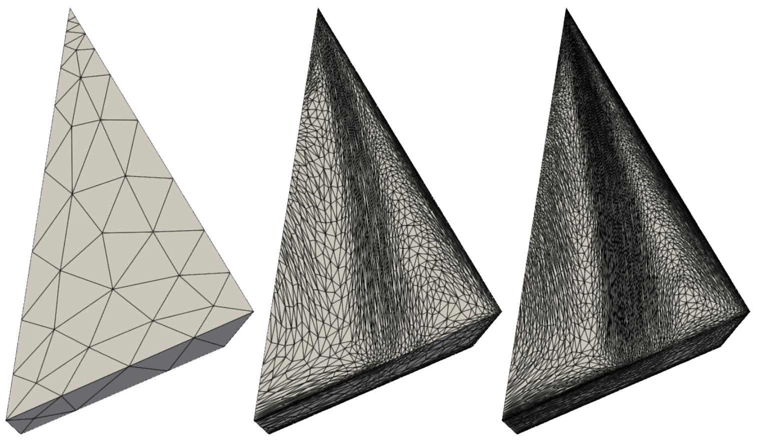


Fig. 51: Adapted mesh at three different complexities. Left: Initial mesh, Middle: mesh at $100,000$ complexity, Right: mesh at $800,000$ complexity.

To access the quality of the results of the adaptation procedure and its coupling with *CDT3D*, drag and lift coefficients are compared against the results presented in [122, 148], and [100]. Figure 53 presents the results. Both the drag and the lift coefficients are within less than 0.55% of all the reference values. The final values as evaluated by SU2 on the 35th iteration are $C_D = 0.165396$ and $C_L = 0.346937$.

Table 32 presents performance data for every 5 iterations of the adaptive pipeline. SU2 is deployed on the ODU's `turing` cluster[32] that houses nodes with a variety of different node specifications. The number of cores used by the solver was set so that it corresponds to about 10,000 vertices per core and it was constrained to 300 to reduce the waiting time in the job scheduler queue of the cluster. *CDT3D* is using one of `turing`'s nodes with two sockets each one with a Intel®Xeon®CPU E5-2698 v3 @ 2.30GHz (16 cores) for a total of 32 cores.

To ease the comparison that involves different core counts and hardware specifications, we include a *core-hours*[33] column. Using core-hours allows to evaluate the performance of the application with respect to the cost of running it on a shared cluster where charge is common to take place in terms of core-hours. The running time of *CDT3D* occupies only a small fraction of the adaptive pipeline.

[32]`https://wiki.hpc.odu.edu/en/Cluster/Turing` (Accessed 2021-04-20).

[33]core-hours = number of cores used by application * hours required for the execution.

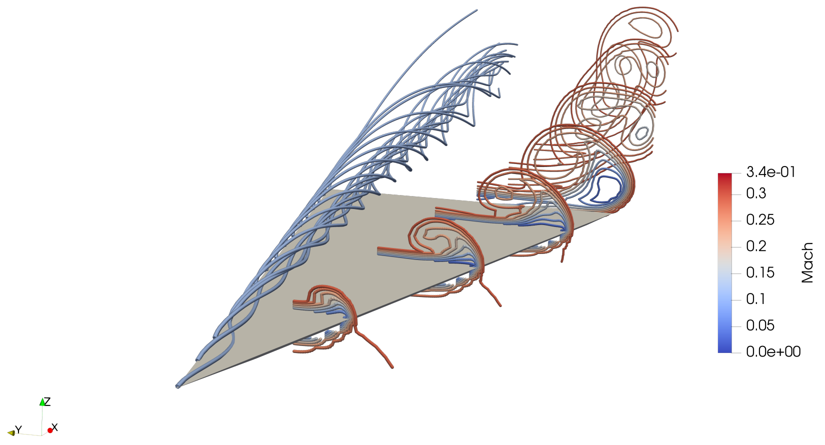


Fig. 52: Streamlines and Contour slices of the Mach number of the solution. (Simulation performed on the half model).

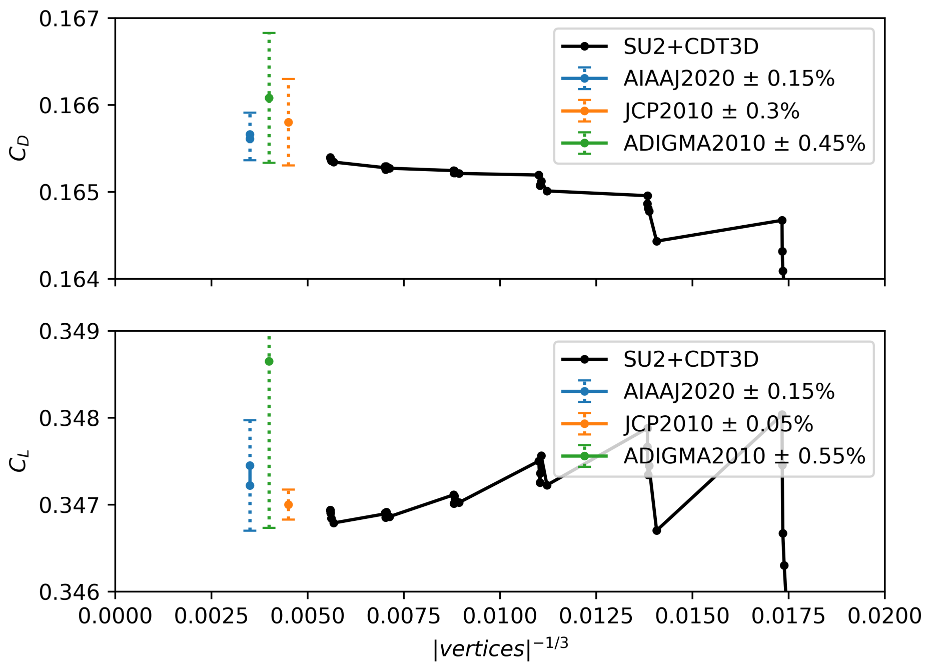


Fig. 53: Lift and drag coefficients as evaluated by SU2 compared against results presented in [100](AIAA2020) [148](JCP2010) and [122](ADIGMA2010).

iter.	vertices	tetrahedra	solver (s)	solver core-hours	*CDT3D* (s)	*CDT3D* core-hours
0	901	3,444	57.55	0.16	-	-
5	97,896	563,930	2,833.51	7.87	62.70	0.56
10	192,098	1,114,412	3,301.11	18.34	47.43	0.42
15	377,569	2,203,660	2,897.73	32.20	108.49	0.96
20	749,290	4,391,974	3,865.75	85.91	198.79	1.77
25	1,467,922	8,641,694	3,476.85	154.53	374.51	3.33
30	2,897,903	17,108,219	2,777.01	232.96	766.87	6.82
35	5,726,724	33,883,975	3,281.82	273.49	1,519.19	13.50

TABLE 32: Performance data of adaptive iterations.

4.2.3 INVISCID ONERA M6 CASE

The next case introduces CAD data to the adaptation pipeline. We use an inviscid flow based on the description of a turbulent case included NASA's Turbulence Modeling Resource (TMR)[34]. As mentioned in NASA's website[35] *"The ONERA M6 wing is a classic CFD validation case for external flows because of its simple geometry combined with complexities of transonic flow [..] It has almost become a standard for CFD codes because of its inclusion as a validation case in numerous CFD papers over the years."* The flow conditions for this study are $3.06°$ angle of attack, 0.84 Mach number and freestream temperature equal to 300K. This case utilizes the ONERA M6 wing geometry (see Figure 40) of the previous section. Figure 54 depicts the initial mesh generated by `ref bootstrap`.

SU2 is configured similarly to the previous case but using the JST as convective scheme which we found to converge faster for this case. For each iteration except the first, we also supplied an interpolated solution on the new mesh based on the solution of the previous iteration. The metric is constructed based on the local Mach field of the solution and the metric gradation value is set to 10. The complexity of the metric is doubled every 5 iterations. The solution-based metric is intersected also with a curvature- and feature-based metric built based on the geometrical features of the wing. We considered 3 metric complexity values for this study: $[50\,000, 100\,000, 200\,000]$.

[34]`https://turbmodels.larc.nasa.gov/onerawingnumerics_val.html` (Accessed 2021-05-31).
[35]`https://www.grc.nasa.gov/www/wind/valid/m6wing/m6wing.html` (Accessed 2021-05-31).

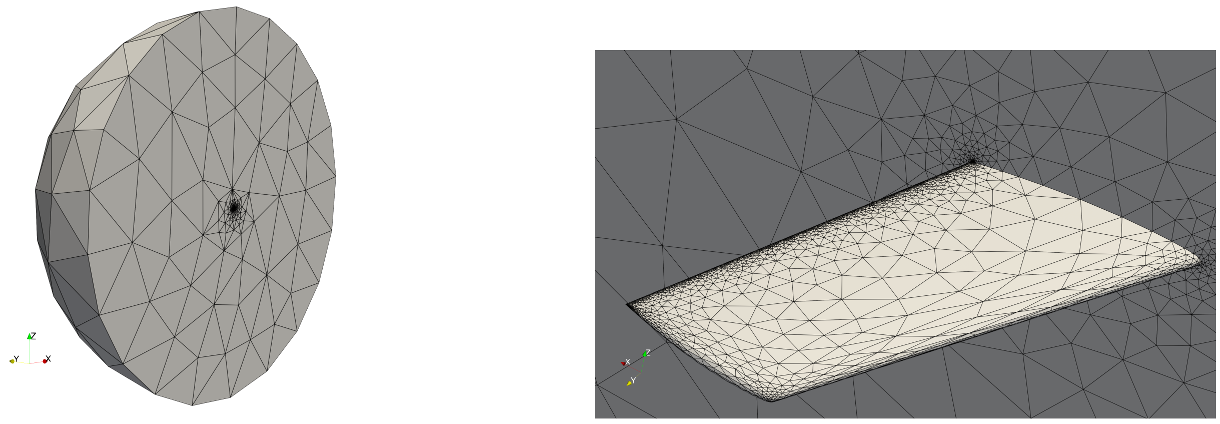


Fig. 54: Initial mesh generated by `ref bootstrap`. The mesh conforms to the geometrical features of the wing.

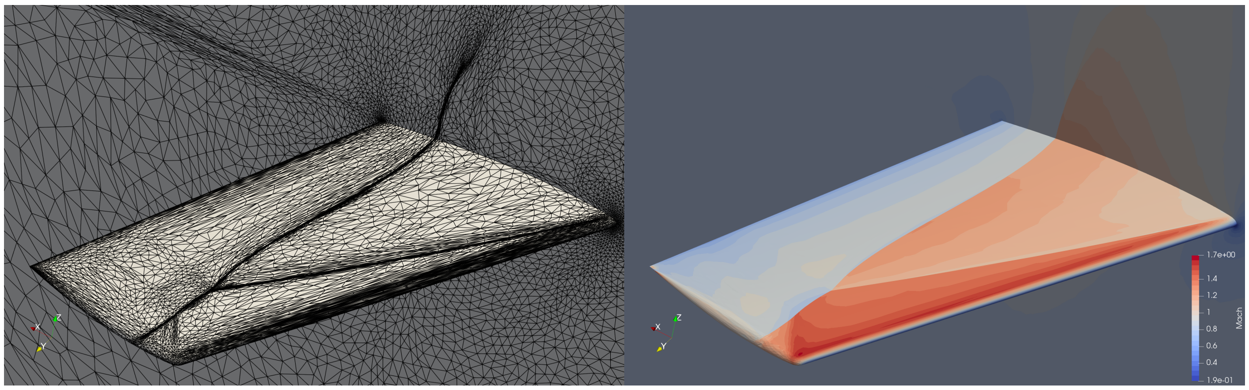


Fig. 55: Final iteration of the adaptive loop.

These flow conditions produce the typical “lambda” shock along the upper surface wing. Figure 55 depicts the mesh as well as the corresponding contour plot of the local Mach number for the final iteration of the adaptive loop.

To verify our results, we compare the pressure coefficient against two different datasets.

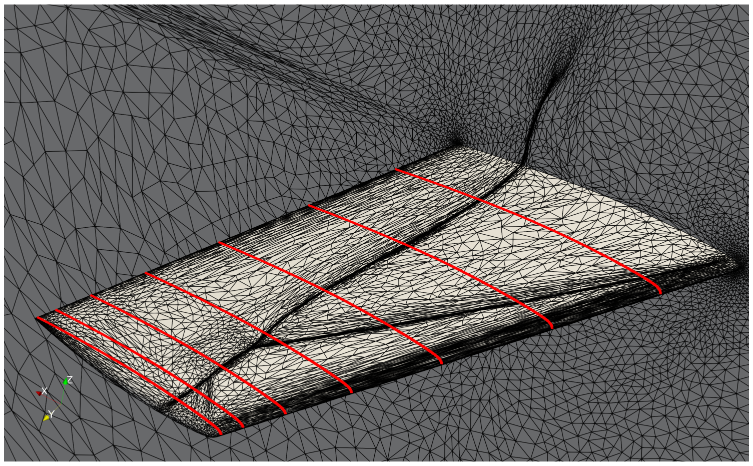


Fig. 56: Location of pressure cross section.

First, the experimental values of Case 2308 of [224] acquired from the TMR website[36] that corresponds to our configuration. Also, we executed SU2 with the same configuration on a structured grid generated using a customized mesh generation code [190] using the input parameters suggested by the TMR website[37]. In particular, we used the level 2 mesh (L2) that has $36,865$ points across the surface of the wing. For comparison, the final mesh of our pipeline has $6,898$ points across the surface of the wing. Figure 56 depicts the 7 sections along which the experimental and numerical results are compared. The rest subfigures of Figure 57 compare the results generated using *CDT3D* and the pipeline of Figure 48, the structured mesh, and the experimental values. The x axis denotes the x-coordinate of the cross-section normalized by the local cord-length of the wind. The y axis represents the local pressure coefficient which measures the pressure at a point relative to the freestream conditions. The combination of *CDT3D* with SU2 generates results very close to the experiment and the numerical solution obtained on the structured mesh. The differences with the experimental values are in part due to the inviscid method used in this simulation. We attempted to perform a viscous simulation using the same configuration but we did not succeed obtaining converged results. Still, these results indicate that the meshes produced by *CDT3D* in presence of simple curved geometries supplied as CAD data are suitable for inviscid calculations and the results are close to the reference values.

[36]`https://turbmodels.larc.nasa.gov/onerawingnumerics_val.html` (Accessed 2021-06-01).
[37]`https://turbmodels.larc.nasa.gov/onerawingnumerics_grids.html` (Accessed 2021-06-01).

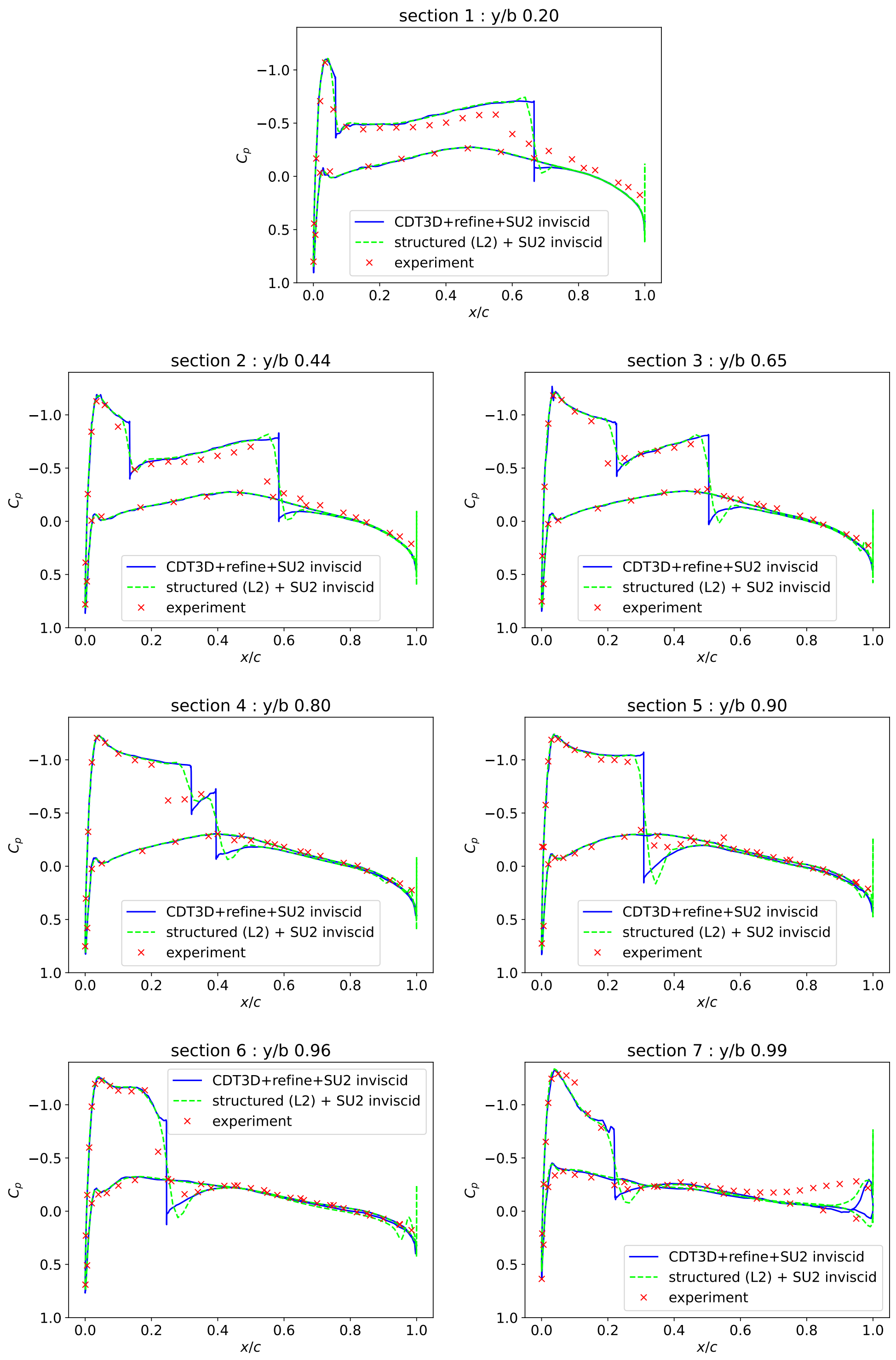


Fig. 57: Values of the pressure coefficient as evaluated by the solver versus the experiment across the 7 sections of Figure 56.

4.2.4 INVISCID FLOW OVER THE JAXA STANDARD MODEL

As a final stress-test, we use the Japan Aerospace Exploration Agency (JAXA) Standard Model (JSM). JSM was built as an attempt to study flow effects over a fairly complete configuration instead of isolated aircraft components that were commonly used. There are several experimental data available, see for example [131, 261, 262] but, we will focus on the use of the JSM in the context the 3rd AIAA CFD High-Lift Prediction Workshop[38]. A summary of the workshop's results appear in [218]. In particular, we will study the case 2b that excludes the pylon and the nacelle of the original model and uses an angle-of-attack equal to 4.36° and a Mach number of 0.172. The JSM geometry is combined out of 200+ surfaces, modeling details of the aircraft including brackets, flaps and slats (see Figure 58).

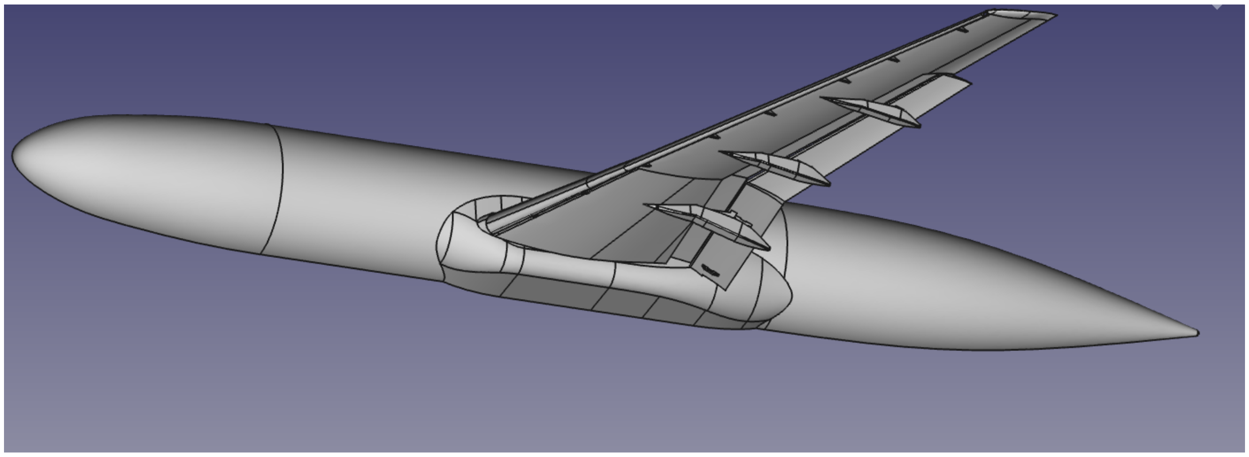


Fig. 58: The JSM geometry.

SU2 is configured similarly to the inviscid ONERA M6 case of the previous section. For each iteration except the first, we also supplied an interpolated solution on the new mesh based on the solution of the previous iteration. For the first iteration, we used the coarse mesh of Figure 59 created by `ref boostrap` of the *refine* mesh mechanics suite [198].

[38]`https://hiliftpw.larc.nasa.gov/index-workshop3.html` (Accessed 2021-06-01).

The metric is constructed based on the Mach field of the solution and the metric gradation value is set to 1.5. The complexity of the metric is doubled every 5 iterations. The solution-based metric is intersected also with a curvature- and feature-based metric built based on the geometrical features of the model. We considered 8 metric complexity values for this study: $[50\,000, 100\,000, 200\,000, 400\,000, 800\,000, 1\,600\,000, 3\,200\,000, 6\,400\,000]$. The final mesh contains $13,227,952$ vertices, $478,518$ triangles and $78,479,450$ tetrahedra.

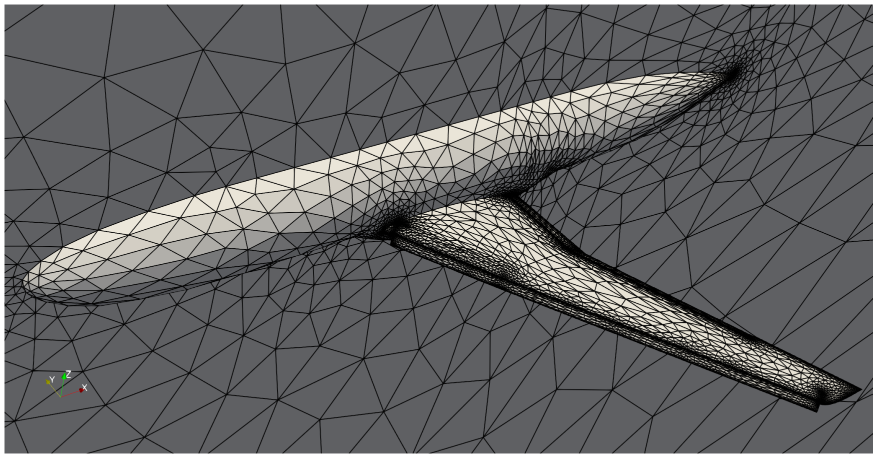


Fig. 59: Initial coarse mesh created by `ref bootstrap`. # vertices 52,265, # triangles 57,240, # tetrahedra : 219,230.

Figure 60 depicts the upper surface of the wing of the final iteration along with the distribution of the local Mach number around it. Notice that the method inserts more points around the regions of higher variability of the local Mach number as expected. In particular, the wakes of the slat brackets are resolved on the upper surface. These wakes are initiated at the sharp edges of the brackets. Figure 61 depicts the final mesh along with the final solution colored my the local Mach number. Zoom-in views of one of the generated vortices are also provided.

To verify our results we compare the pressure coefficient values as evaluated by the solver against experimental results acquired from High-lift workshop website[39]. Figure 62a depicts the locations of C_p extraction along the wing of JSM. The rest subfigures of Figure 62 present results generated using our approach and the pipeline of Figure 48. In general, the obtained results are close to the experimental values. Notice, however, that our results overpredict the C_p values on the upper surface of the wing with corresponds to the upper section of the blue datapoints. This is in part attributed to the fact that we used an inviscid simulation instead of a viscous. Viscous simulations were attempted starting from a coarse mesh but we didn't succeed in obtaining a converged solution. Still, this case indicates that the new functionality of *CDT3D* allows the method to handle fairly complicated CAD data in combination with solution-based metric derived from inviscid calculations.

[39]`https://hiliftpw.larc.nasa.gov/Workshop3/pressures.html` (Accessed 2021-06-17).

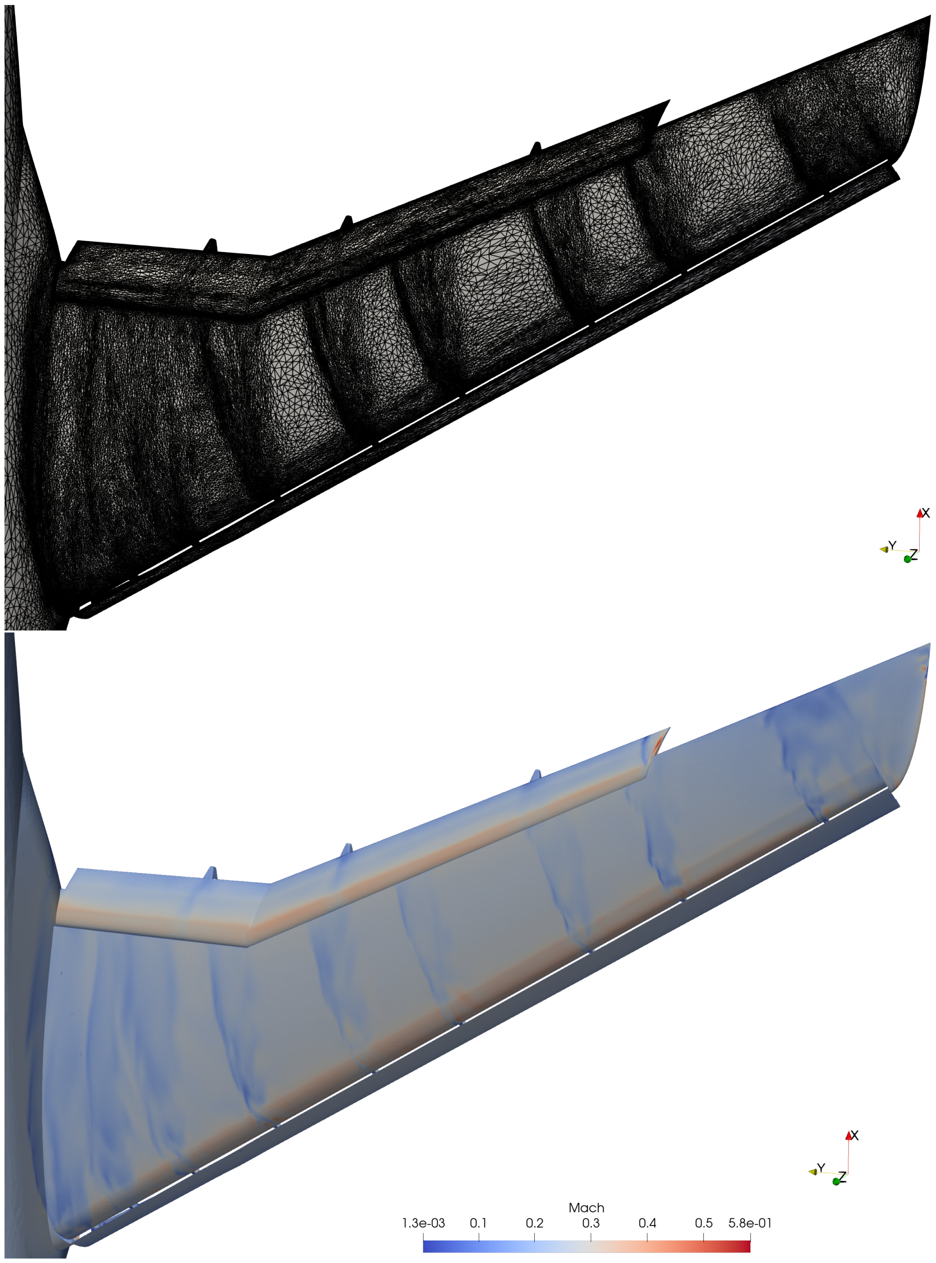


Fig. 60: Final mesh and coloring of the wing by the local Mach number.

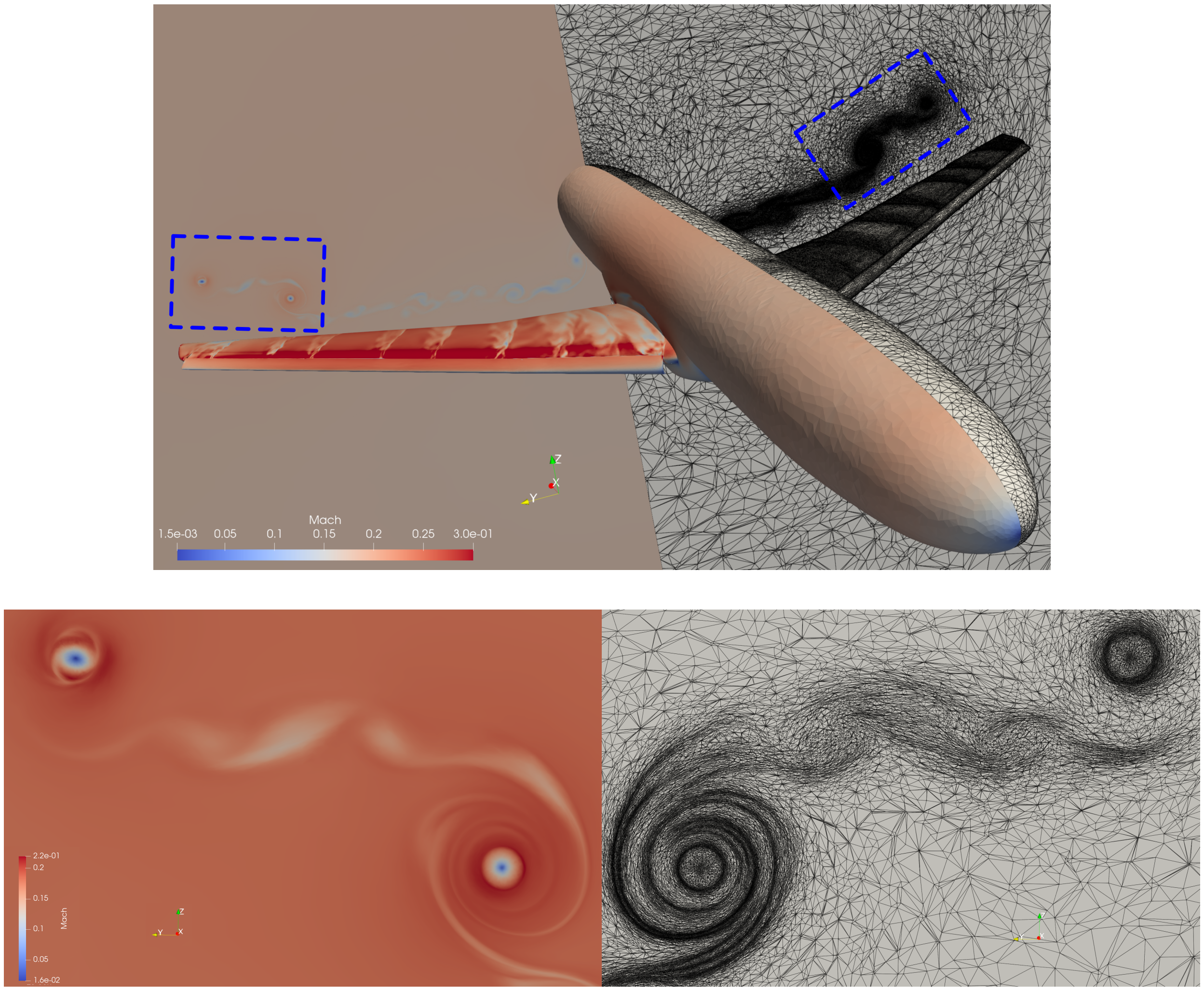
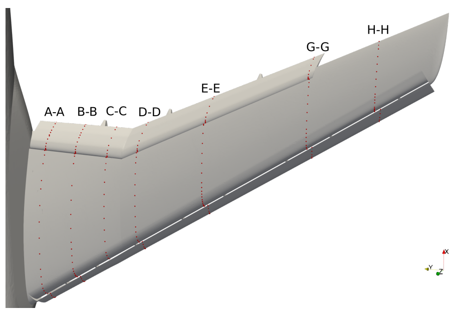
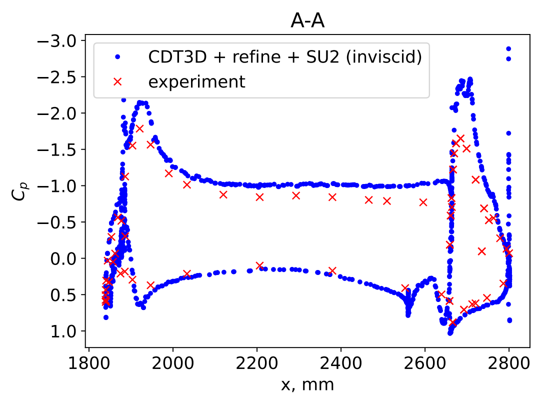
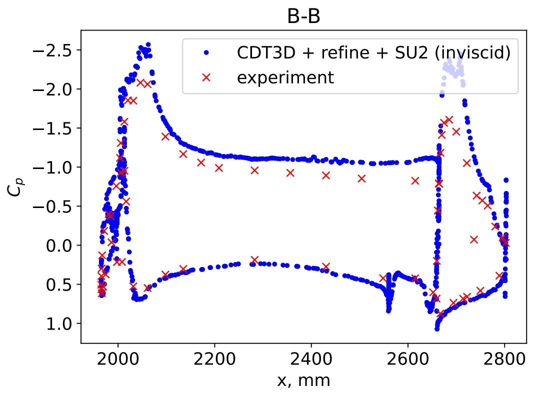


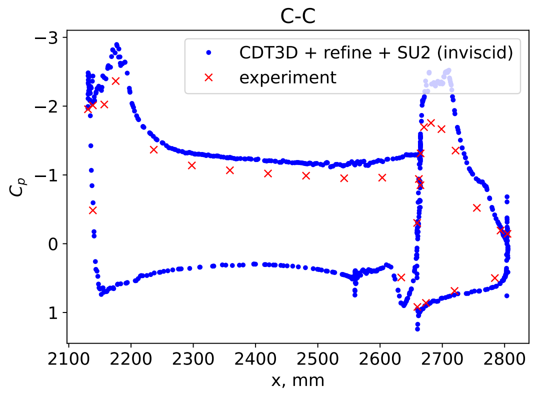
Fig. 61: Simulation results. Top: Final mesh alongside the corresponding solution. Bottom: Zoom-in of the blue regions of the top figure.

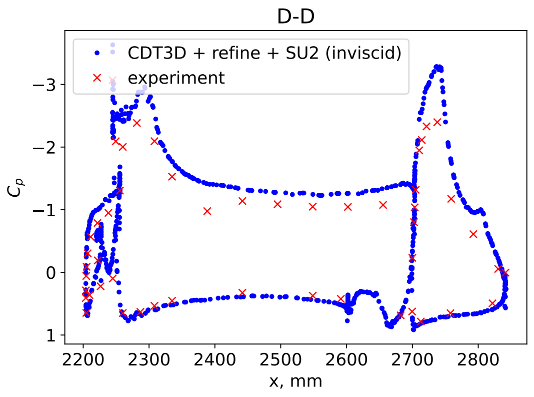


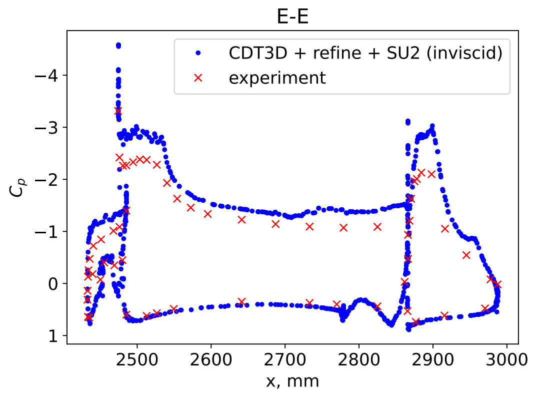
(a) Locations of experimental data measurements.

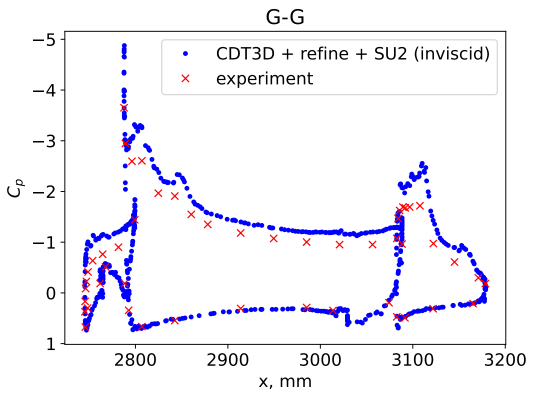












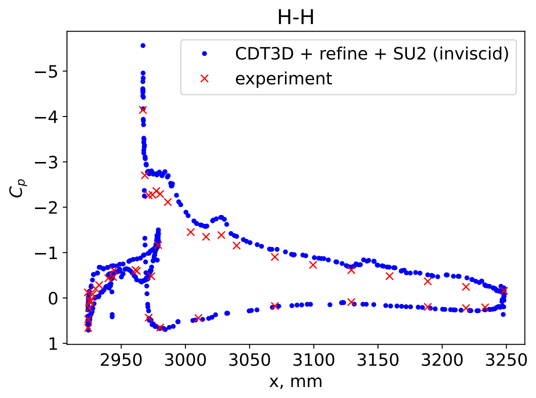


Fig. 62: Values of the pressure coefficient as evaluated by the solver versus the experiment across the 7 sections of Figure 62a.

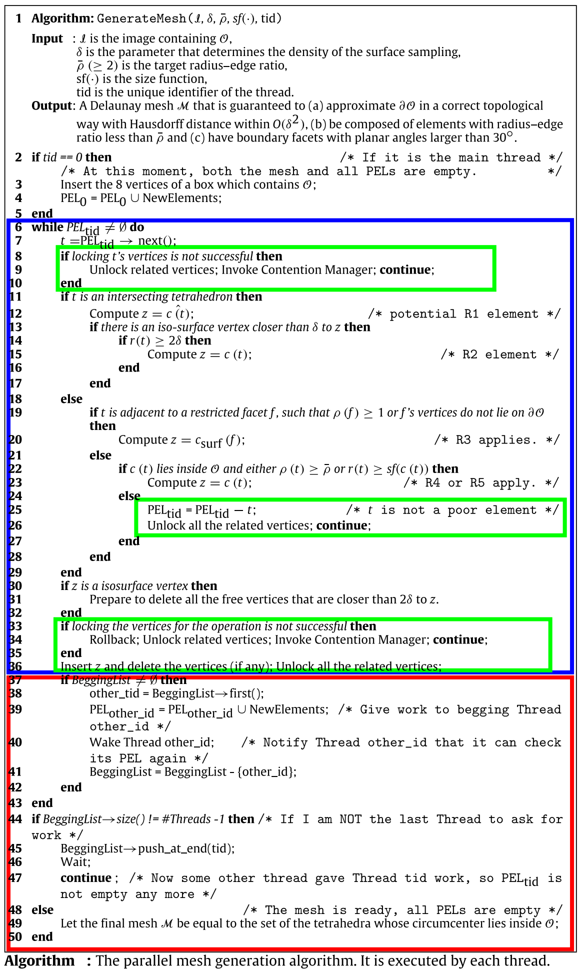
CHAPTER 5

A TASKING FRAMEWORK FOR PARALLEL MESH OPERATIONS

So far, we have presented significant advances in terms of both functionality and performance. As the complexity of the methods and subsequently of the codes increases, the need for abstracting logical parts of the method into separate modules becomes paramount. Handling the ever-increasing complexity of mesh generation codes along with the intricacies of newer hardware often results in codes that are both difficult to comprehend and maintain. Different facets of codes such as thread management and load balancing are often intertwined, resulting in efficient but highly complex codes. Although platform-specific and/or application-specific optimizations will always perform better than generic solutions, abstract interfaces can last longer and allow for better interoperability between applications. Choosing the right abstractions allows applications to build upon a generic framework while enabling low-level software substrates to offer implementations that take advantage of the underlying hardware. Moreover, abstractions provide space for future explorations; as newer hardware (e.g., in the form of accelerators) becomes available, the application developer may need to perform minimal to no changes while the underlying runtime system can add new features opaquely.

The goal of this chapter is to present an approach for separating the concerns of *functionality* and *performance*, specifically for mesh operations. Figure 63 depicts the pseudocode of two fine-grained speculative meshing operations. Note that the application developer must manage and account for the meshing kernel, parallel correctness, and load balancing, all within a single algorithm. Developing and maintaining such an application becomes challenging since the developer has to keep all three parts in mind while modifying the code. Moreover, re-using hardware-specific optimizations among different applications can only be achieved by abstracting them outside of specific applications. Examples include affinity-aware work schedulers, cache-optimized data structures, etc.

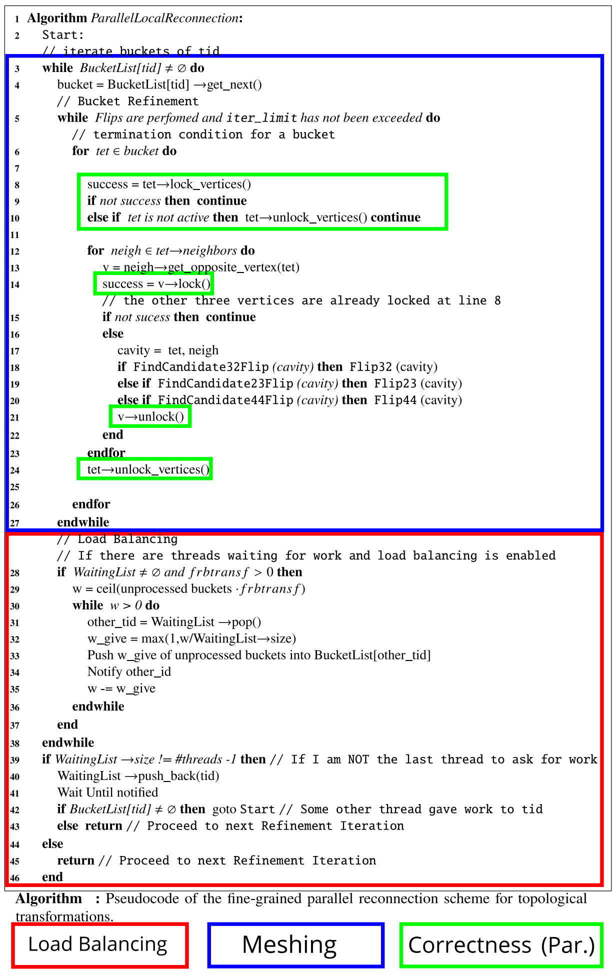
To demonstrate this method we will use two case studies, the parallel meshing operations of *CDT3D* described in Section 3 that are common in most metric-based mesh adaptation codes [250] and the Delaunay-based kernel of *PODM* [95]. For these two applications, we explore how a tasking environment can be used to express speculative mesh operations and how it allows to abstract the load balancing aspects of both case studies with small (for *PODM*) to no (for *CDT3D*) impact to functionality. The results in Sections 5.3.1



```
Algorithm: GenerateMesh($\mathcal{I}$, $\delta$, $\bar{\rho}$, $sf(\cdot)$, tid)
Input  : $\mathcal{I}$ is the image containing $\mathcal{O}$,
         $\delta$ is the parameter that determines the density of the surface sampling,
         $\bar{\rho}$ ($\geq 2$) is the target radius–edge ratio,
         sf($\cdot$) is the size function,
         tid is the unique identifier of the thread.
Output: A Delaunay mesh $\mathcal{M}$ that is guaranteed to (a) approximate $\partial\mathcal{O}$ in a correct topological
         way with Hausdorff distance within $O(\delta^2)$, (b) be composed of elements with radius–edge
         ratio less than $\bar{\rho}$ and (c) have boundary facets with planar angles larger than 30°.
if tid == 0 then                                          /* If it is the main thread */
    /* At this moment, both the mesh and all PELs are empty.                          */
    Insert the 8 vertices of a box which contains $\mathcal{O}$;
    $PEL_0$ = $PEL_0$ ∪ NewElements;
end
while $PEL_{tid} \neq \emptyset$ do
    $t$ =$PEL_{tid}$ → next();
    if locking t's vertices is not successful then
        Unlock related vertices; Invoke Contention Manager; continue;
    end
    if t is an intersecting tetrahedron then
        Compute $z = c\,(\hat{t})$;                            /* potential R1 element */
        if there is an iso-surface vertex closer than $\delta$ to z then
            if $r(t) \geq 2\delta$ then
                Compute $z = c\,(t)$;                              /* R2 element */
            end
        end
    else
        if t is adjacent to a restricted facet f, such that $\rho\,(f) \geq 1$ or f's vertices do not lie on $\partial\mathcal{O}$
        then
            Compute $z = c_{surf}\,(f)$;                           /* R3 applies. */
        else
            if $c\,(t)$ lies inside $\mathcal{O}$ and either $\rho\,(t) \geq \bar{\rho}$ or $r(t) \geq sf(c\,(t))$ then
                Compute $z = c\,(t)$;                         /* R4 or R5 apply. */
            else
                $PEL_{tid}$ = $PEL_{tid} - t$;              /* t is not a poor element */
                Unlock all the related vertices; continue;
            end
        end
    end
    if z is a isosurface vertex then
        Prepare to delete all the free vertices that are closer than $2\delta$ to z.
    end
    if locking the vertices for the operation is not successful then
        Rollback; Unlock related vertices; Invoke Contention Manager; continue;
    end
    Insert z and delete the vertices (if any); Unlock all the related vertices;
    if BeggingList $\neq \emptyset$ then
        other_tid = BeggingList→first();
        $PEL_{other\_id}$ = $PEL_{other\_id}$ ∪ NewElements;  /* Give work to begging Thread
        other_id */
        Wake Thread other_id;     /* Notify Thread other_id that it can check
        its PEL again */
        BeggingList = BeggingList - {other_id};
    end
end
if BeggingList→size() != #Threads -1 then /* If I am NOT the last Thread to ask for
work */
    BeggingList→push_at_end(tid);
    Wait;
    continue ;  /* Now some other thread gave Thread tid work, so $PEL_{tid}$ is
    not empty any more */
else                                      /* The mesh is ready, all PELs are empty */
    Let the final mesh $\mathcal{M}$ be equal to the set of the tetrahedra whose circumcenter lies inside $\mathcal{O}$;
end
```

Algorithm : The parallel mesh generation algorithm. It is executed by each thread.

(a) PODM pseudocode as presented in [95].



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

Algorithm : Pseudocode of the fine-grained parallel reconnection scheme for topological transformations.

(b) *CDT3D* pseudocode as presented in [85].

Fig. 63: Pseudocodes of the speculative approach applied on a Delaunay-based algorithm (left) and a local reconnection operation (right). Colored regions indicate the primal function of the enclosed steps.

and 5.3.2 indicate not only low overhead, but even speedup with respect to the baseline hand-optimized applications for some of the mesh operations. In summary, this chapter:

- Presents a high-level front-end that abstracts and unifies task management for adaptive and irregular applications.
- Describes the design and implementation of the front-end for three major back-ends: Intel$^{®}$'s TBB, OpenMP, and Argobots.

- Illustrates how this front-end can be applied to two different speculative parallel unstructured mesh generation codes.
- Provides an in-depth analysis of the effect of task granularity for each back-end and the advantages and disadvantages of different task creation strategies.

5.1 RELATED WORK

A complete review of the current state-of-the-art tasking environments is outside the scope of this section. A comprehensive taxonomy based on architectural characteristics and user APIs appears in [245]. In the rest of this section, we focus on methods that exploit concurrency through speculative execution. As mentioned in Section 2.1, *Speculative* execution (also known as *optimistic*) is a technique that allows for the exploitation of more concurrency out of a problem by executing steps of a procedure ahead of time, prior to resolving data dependencies between the steps themselves.

There several efforts in the literature that facilitate speculative execution utilizing higher level constructs. Among the many we list a few pertinent to this study such as the use of transactional memory at the software level [209], compiler-assisted methods [7,40,210] and libraries such as Galois [142], ParlayLib [28] and SPETABARU [36].

The Galois system [142] provides abstract set iterators, giving to the application developer the ability to extract parallelism out of the work-lists of a sequential application. Custom data structures and a runtime scheduler are responsible for detecting and recovering unsafe accesses to shared memory. The elegance of this approach is appealing but, for our use-case, it would require extensive modifications of the worklists maintained by each application. Also, to the best of our knowledge, its effectiveness for mesh generation has been demonstrated only on simple sequential mesh triangulation codes. In contrast, in this work, both use-cases build on top of an already-parallelized application that have demonstrated comparable performance to state-of-the-art methods [95,251].

In [28,29] the authors revisit the idea of expressing speculative execution as a combination of nested parallelism and commutative operations suggested in [240] and propose the use of *deterministic reservations* for dealing with a class of greedy algorithms. The main idea is to split the operation into two phases. One that attempts to reserve the data dependencies for a number of tasks speculatively, and then a commit phase that executes the tasks that successfully reserved all their dependencies. The two-phase approach is similar to the *inspector-executor* model [221]. However, an *inspector-executor* model is not suitable for our case due to the data-intensive nature of the targeted use-cases. In contrast, our approach re-uses the speculative approach step already present in both use-cases and merges the two steps in one. This approach acts directly upon touched data which improves cache utilization and allows tolerating more than 80% of system latencies [187]. Moreover, it avoids the synchronization required by a two-phase approach.

The SPETABARU tasking runtime system introduced in [36] can exploit concurrency of task graphs through speculation. The task graph is built based on the user-defined data dependencies between the tasks. The system manages the execution of tasks as well as disregarding data from failed speculative attempts upon runtime. The library was originally created for Parallel Monte Carlo Simulations and it is primarily designed for parallel applications that utilize graphs of tasks. This tasking system generates the graph utilizing a single thread in a pre-processing step that generates all the tasks and evaluates the data dependencies among them. This approach is inadequate for our target data-intensive applications for two reasons. First, dependency discovery and resolution is the most expensive step, thus rendering the pre-processing step to a major bottleneck. Moreover, the continuous generation of new elements (and therefore tasks) would require additional synchronization points which would degrade performance significantly.

In [7] the authors incorporate Thread-Level Speculation in OpenMP. OpenMP is extended with the `speculative` directive which annotates a variable as the target of speculative execution. A thread-local version of such variables is created for each thread. The respective runtime monitors such variables and guarantees that all read accesses will return the most up-to-date value. When a thread consumes an outdated version of a `speculative` variable, it is stopped and restarted in order to consume the correct value of the variable. For both of our applications the speculative execution is already part of the application code and modifying it is outside the scope of this work. Moreover, the abstract front-end of our approach gives access to additional back-ends beyond OpenMP.

Although many of the above approaches are close to our goals, most of them will require nontrivial changes to the code required for the *Parallel Correctness* steps (see Figure 63) of the algorithm which for this study we chose to keep as part of the meshing task. Moreover, in contrast to all the presented approaches the starting point in this work is applications that are already parallel instead of sequential. This comes with the benefit of having thread-safe mechanisms in place for memory allocation and speculative locking, but also with higher complexity due to their legacy nature. Also, the proposed approach can utilize a number of different back-ends, including Argobots [227]. Argobots provides lightweight User Level Threads (ULTs), capable of context switching with low overhead. Among others, ULTs are employed to tolerate latencies in cases such as failing to acquire a lock or calling synchronous MPI operations where regular tasks would block, along with the underlying hardware thread. Instead, a ULT will implicitly release the hardware thread it runs on, allowing other ULTs to use it. Argobots is an integral part of the PREMA runtime system [244] which in turn is a building block of the *Telescopic Approach* [61].

In the context of mesh generation, speculative execution was introduced in [187] where the authors execute the same meshing kernel across multiple processes without restricting them on their local data. Instead, the meshing kernel is launched *optimistically* and the data dependencies are discovered and captured on the fly. If some dependency cannot be satisfied, the operation releases any captured dependencies, aborts its execution (rollback), and reenters the scheduler's pool. The speculative approach has already been utilized in Delaunay triangulation methods [23,27,93], Delaunay Refinement [95] and Advancing-Front [85] methods. However, in each case the approach was application- and method-specific. In contrast, in this work we provide an abstract interface allowing the framework to be applied not only across different mesh operations but also between different mesh applications. Moreover, the approach in this work is method-agnostic, thus rendering the framework extremely useful not only for other meshing methods but also for adaptive and irregular applications in general.

5.2 METHOD

The proposed approach builds upon the observation that the speculative meshing operations of the two case studies can be decomposed into three components: *Meshing*, *Parallel Correctness* and *Load Balancing* (see Figure 63). The Load Balancing part is handled by the generic tasking framework presented in the following sections. *Parallel Correctness* is expressed through the use of atomic locks upon the cavity (data dependencies) of each operation. For this study, the parallel correctness steps will remain part of the meshing task. The implementation of generic tasking framework is composed of an application-facing front-end which is agnostic to target hardware, and a back-end, which provides custom implementations for individual substrates.

5.2.1 FRONT-END: A GENERIC TASKING FRAMEWORK

The general approach used in this work is to decompose any given operation into non-interruptible tasks that execute to completion. The framework supports both blocking calls for creating many tasks (see Listing 5.1) and a fine-grained API for single task creation (see Listing 5.2). This allows for the utilization of a range of tasking paradigms, from simple fork-join models to hierarchical or recursive task creation (see Figure 64).

```
/**
* @brief launch tasks and wait until they all finish
* @param user_task_args vector of task arguments
* @param user_func the user function to be executed,
* type should be void func(UserTaskArgs&),
* @param grainsize number of elements of user_task_args to group
* into a single task
*/

template<typename UserTaskArgs, typename FunctionType>
void task_for(std::vector<UserTaskArgs>& user_task_args,
              FunctionType user_func,
              int grainsize = 10)
```

Listing 5.1: Interface for launching several tasks at once.

```
/**
* @brief add a task to the internal queue
* @param user_task_args arguments of the task
* @param user_func the user function to be executed,
* type should be void func(UserTaskArgs&)
*/

template<typename UserTaskArgs,typename FunctionType>
void create_and_schedule(UserTaskArgs& user_task_args,
                         FunctionType user_func)

/**
* @brief wait until all generated tasks have completed.
*/
void wait_for_all()
```

Listing 5.2: Interface for creating a single task.

`task_for` in Listing 5.1 is similar to the parallel-for paradigm. It accepts a function `user_func` that implements the task operation as well as a vector `user_task_args` of the different arguments for each task. Optionally, it can accept a grainsize which controls the number of terminal tasks that will be generated. The number of terminal tasks is `user_task_args.size()/grainsize`. Similar to `std::for_each`, `task_for` will apply `user_func` on each element of the `user_task_args` vector. However, in contrast to `std::for_each` not all invocations of `user_func` will be completed successfully. Some will abort due to rollbacks. In this study, re-applying the operation on aborted tasks is handled by the application logic since it was already present before the introduction of this framework.

`create_and_schedule` in Listing 5.2 is a simple wrapper around the corresponding backend that generates a task and places it in the internal queue of the framework. This call is not blocking and the execution of the task may start immediately on a different thread. Finally, `wait_for_all` suspends the calling thread until the internal task queues are empty.

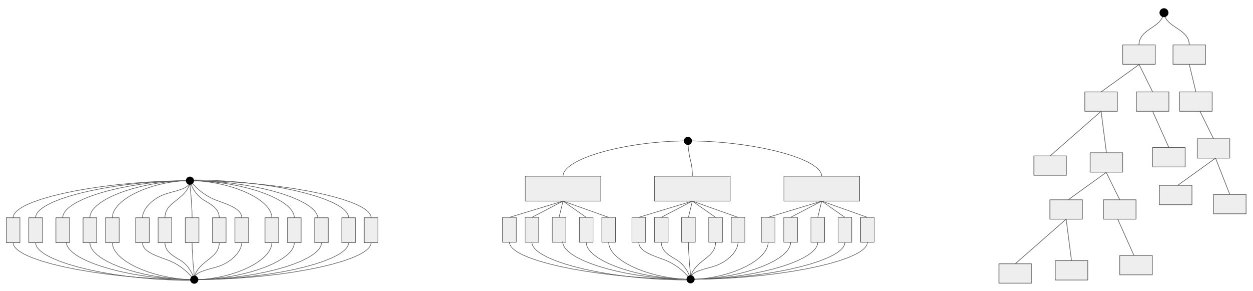


Fig. 64: Different tasking paradigms employed in this work. Left: flat model, Middle: two-level task creation, Right: hierarchical task creation.

The tasking framework also provides the user with a unique `thread_id` $\in [0, nthreads)$. This id is not pinned to any hardware thread, but it is guaranteed to stay fixed and unique for the duration of the task execution. The `thread_id` is required by both applications of this study for two main operations. First, data dependency acquisition is implemented utilizing atomic locks that hold the id of the owning thread [85, 95]. Also, both pieces of software utilize the thread-aware memory management method described in [13] that uses a `thread_id` in order to access the appropriate thread-local memory pools that allow for the allocation and deallocation of elements in a thread-safe manner.

5.2.2 TASK GENERATION STRATEGIES

One of the considerations of explicitly creating tasks is the overhead of task creation. In the current implementation of `task_for`, there is support for all three task creation strategies of Figure 64. The `flat` model implements a basic fork-join paradigm [66]. It creates all tasks sequentially and waits for them to complete. As a first attempt to reduce the overhead, a `2-level` task creation strategy was introduced. For this strategy, the application thread will spawn sequentially $2 \cdot nthreads$ tasks that partition the range of the `user_task_args` vector in equal parts. Each level 2 task will then iterate the assigned range of the task vector and spawn a task for each task-argument. Finally, the `hierarchical` model employs a divide-and-conquer scheme; it creates tasks recursively by creating two child tasks that bisect the task range up to the point where the assigned range is smaller or equal to the target grainsize. When the framework is used sequentially, no tasks are created independently of the chosen strategy. Instead, the application thread will apply `user_func` sequentially on each item of the `user_task_args` vector.

5.2.3 IMPLEMENTATION

The above framework is implemented with three different back-ends: the Argobots runtime system [227], Intel's TBB framework [258] and OpenMP [69]. For each of the three implementations we have incorporated the three task creation strategies of Figure 64 as well as high level constructs specific to each back-end such as `tbb::parallel_for`, `#pragma omp parallel for` and `#pragma omp taskloop` for a total of 12 different execution back-ends. We will use the notation `backend-strategy` to refer to tasking strategy `strategy` implemented on top of the back-end `backend`.

5.2.3.1 Argobots back-end implementation details

Argobots is a low-level tasking framework developed to support higher level runtime systems, so it does not provide optimized schedulers for fork-join parallelism out of the box. To implement an optimized tasking framework for our needs, we developed custom scheduling mechanisms using the interfaces provided. Each thread (execution stream in Argobots' terminology) in the parallel environment is associated with a circular double-ended queue (deque) which is thread-safe and lock-free [46]. Every new task is pushed to the top of the deque of the thread that created it. When a thread finishes with the execution of a task, it first checks its own deque; if there are tasks available, it pops the one residing at the top of the deque and executes it. If its deque is empty, it will randomly pick one of the remaining threads and try to steal the task at the bottom of its deque. By picking the task at the top of the owned deque first, tasks that are hot in the cache are given priority. On the other hand, stealing the task at the bottom of other threads' deques: increases the chance of picking tasks that will create more child tasks, allows more work to become available for the stealing thread and results in a decreased number of steal attempts. We provide two tasking flavors for this implementation - User Level Threads (ULTs) that can yield explicitly and Tasklets that run to completion and can only block waiting for another tasklet created using this framework. In this work, both case studies use tasklets as we only need to wait for other tasks to complete and no other blocking operation is performed. Each task is created using a `abt::task_create` function call that asynchronously schedules a new task and immediately returns a task handle. The task handle can then be used to check or wait for the completion of the respective task's execution. Internally, the call to wait for a task completion will result in calling the scheduler and popping/stealing some other task.

5.2.3.2 TBB back-end implementation details

Intel®Thread Building Blocks (TBB)[40] is a library that enables parallel programming across different applications and architectures. It provides high level constructs such as `tbb::parallel_for` but also gives access to the lower level tasking queues. TBB uses tasks to express parallelism thus making it a good candidate for this study. Tasks are expected to be non-preemptive which is the case for both applications of this study and for speculative operations in general. The scheduler switches the running thread only when a task is waiting for its spawned children. For the `hierarchical` and the `2-level` task creation strategy, each level is enclosed in a `tbb::task_group` that allows to wait until all tasks of the group are completed. When using the lower level `create_and_schedule`, all generated tasks are added to the same global `tbb::task_group` thus allowing termination to be detected in a convenient manner while still enabling work stealing among all threads. For comparison, we also implemented a wrapper that passes the arguments of `task_for` directly to the higher

[40]Recently Intel®Threading Building Blocks was renamed to Intel®oneAPI Threading Building Blocks (oneTBB) to highlight that the tool is part of the oneAPI ecosystem.

level `tbb::parallel_for` function.

5.2.3.3 OpenMP back-end implementation details

OpenMP is an API that enables parallel shared-memory programming with the use of `#pragma`s making it easily accessible directly through the compiler. It is included in this study since it is often the first step towards introducing parallelism for many scientific applications. Tasks are created using `#pragma omp task` and they are declared as `untied` which gives them the opportunity to be scheduled on any available thread. For the `hierarchical` strategy, it was also advantageous to prepend `#pragma omp taskyield` right before the recursive step. This created an extra scheduling opportunity for the back-end. Without it, it was noticed that a single thread would tend to run all the tasks it created, affecting performance and greatly increasing the recursion tree size. For comparison, we also implemented two more wrappers using higher level constructs. The first passes the arguments of `task_for` directly to `#pragma omp for` while the second passes to `#pragma omp taskloop`. For the `#pragma omp for`, we chose the `dynamic` scheduler because it performs on average better across the different mesh operations covered in this work.

5.3 CASE STUDIES

As case studies we use the parallel meshing operations present in *CDT3D* [82,85] and the Delaunay-based kernel of *PODM* [95]. These two applications share a lot of common ideas when it comes to parallel execution, but they also have some differences. As mentioned in Figure 63, both applications utilize speculative execution for their meshing operations which they implement similarly; the meshing kernel (blue sections) uses atomic locks speculatively to guarantee correctness (green sections). This feature fits well with our approach since we assume that each task should be non-interruptible and should execute to completion. Also, they both integrate load balancing and thread management with the mesh application (red sections), that our framework can abstract away.

PODM is built around a single mesh operation for modifying the mesh, thus reducing the amount of code changes required. On the other hand, when it comes to parallel execution, it has a number of optimizations that complicate the use of the tasking framework. In particular, it uses a Hierarchical Load Balancing that ties worklists to specific threads in order to improve data affinity and takes into account the cost of memory access when moving load between threads. These optimizations result in a tight coupling between the mesh operations and the load balancing parts of the code.

In contrast, in *CDT3D* the coupling between the threads and their data is lower. At a high level it follows a fork-join pattern where sequential steps prepare global data structures for parallel execution. This structure matches well with the `task_for` API and reduces the places where the code needs to be modified. Moreover, it utilizes a set of different mesh operations that use both hand-optimized and generic work sharing methods which results in varying impact on performance when transitioning to the tasking framework.

5.3.1 CASE STUDY I : PARALLEL MESH ADAPTATION SOFTWARE (*CDT3D*)

CDT3D is composed of many modules depicted in Figure 65. With proper re-arrangement of the modules one could implement different meshing applications as described in [253]. The configuration chosen for this study is optimized for metric-based adaptation and has been already compared against state-of-the-art mesh adaptation codes in [251].

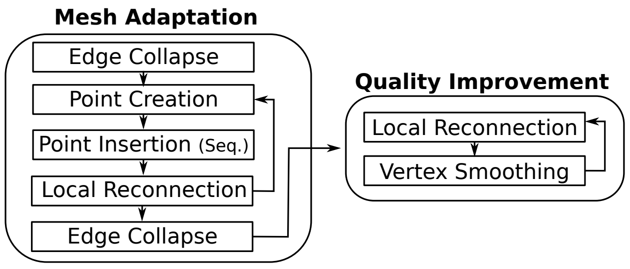


Fig. 65: Mesh operations in *CDT3D*.

For this case-study, the focus will be: Point Creation, Local Reconnection, Edge Collapse, and Vertex Smoothing. The common first step for porting the operations is to express them in a way that is compatible with the API of the front-end presented in Figure 5.1. The most natural choice is as an operation applied to an element. However, the baseline implementation of *CDT3D* already uses "buckets" (i.e, lists of elements) for some of its operations (see, for example, Figure 63b and references [82, 85]). In the context of the presented mesh operations, "buckets" are used as simple strip partitioning method similar

Operation	Work-unit	Operator	Baseline implementation
Local Reconnection	"Bucket"	Apply local reconnection between an element and its face neighbors for each element of a bucket	custom scheduling ([85])
Point Creation	"Bucket"	Generate candidate points for each element of the bucket	custom scheduling ([85])
Edge Collapse	Vertex	Collapse small edges attached to a mesh vertex	`omp for schedule(guided)`
Vertex Smoothing	Vertex	Improve the quality of the elements attached to a mesh vertex by smoothing	`omp for schedule(static)`

TABLE 33: Characteristics of the baseline implementation of the parallel mesh operations ported to the tasking framework.

to the `chunk-size` parameter of the `#pragma omp parallel for` scheduler. In an effort to maximize code re-use of the application, we opted for the conventions of Table 33.

One important feature of the *Operator* in each case, is that it is built using the speculative/optimistic approach. In practice, it means that no data or domain decomposition is applied to the mesh, but the operator will attempt to acquire its dependencies through some exclusive locking mechanism upon execution. Failure to do so will result in unlocking any acquired resources and exiting.

5.3.1.1 Performance Evaluation

For this evaluation, the code was recompiled picking the appropriate back-end implementation each time. The experiments were performed on the `wahab` cluster of Old Dominion University using dual socket nodes equipped with two Intel®Xeon®Gold 6148 CPU @ 2.40GHz (20 slots) and 368 GB of memory. The compiler is `gcc 7.5.0` and the compiler flags `-O3 -DNDEBUG -march=native`. `gcc 7.5.0` comes with support of OpenMP version `4.5`. For TBB, version `2021.1.1` was used. Each configuration was executed 10 times. All times are normalized based on the performance of the baseline application unless otherwise stated. The graphs below use the geometric mean [91] to summarize the results for each configuration. The evaluation in the following paragraphs proceeds as follows: First, we compare higher-level constructs (`#pragma omp parallel for`, `#pragma omp taskloop` and `tbb::parallel_for`) to the `-flat` strategy implemented using the three back-ends. Next, we compare the `-flat`, `-2-level`, and `-hierarchical` strategies as implemented in our framework. We then analyze and optimize the grainsize for each back-end and strategy in order to derive the optimal grainsize for each operation. Finally, we compare our framework using the optimal grasinsizes with the baseline application.

Higher-level parallel constructs and the `flat` model: For the first benchmark, the `flat` tasking creation model is employed for each back-end and compared against higher-level constructs such as `#pragma omp parallel for`, `#pragma omp taskloop` and `tbb::parallel_for`. As expected, all back-ends exhibit an overhead when using the `flat` strategy compared to higher-level constructs due to the cost of sequentially creating all the tasks. Figure 66 presents the running time normalized with respect to the baseline implementation. `omp-flat` back-end suffers from the highest overhead especially when more than 20 cores are used, which is the size of the socket for this machine. This trend is in part attributed to the fact that the naive creation of tasks in the `flat` model along with the `untied` specification allows any task to run on any core without any consideration about the affinity of data with respect to the cores. The higher level constructs perform better than their `-flat` counterparts. `#pragma omp taskloop` improves significantly over `omp-flat` by merging multiple loop iterations into a single task, thus, decreasing the number of tasks that need to be created and scheduled. Moreover, by creating and scheduling fewer tasks, the number of context switches and cache-line invalidations is also reduced. `#pragma omp parallel for` equipped with the `dynamic` scheduler performs even better thanks to the absence of the overhead of task creation. `tbb::parallel_for` outperforms the rest by dynamically adjusting the loop ranges assigned to each task, based on the number of threads, the time for each task execution, and hardware occupancy.

Comparison between the `flat`, `2-level` and `hierarchical` task creation strategies: In Figure 67, the three task creation strategies are compared with each other. Both the `2-level` and the `hierarchical` strategies reduce significantly the overhead in comparison to the `flat` strategy. In the `2-level` strategy, $2 \cdot nthreads$ level 1 tasks that partition the `user_task_args` vector are created sequentially. Then, each level 1 task generates tasks that apply *Operator* to the appropriate unit of work based on Table 33. For this dataset, the `grainsize` is set to 1, which results in creating a level 2 task for each unit of work. Both the `2-level` and the `hierarchical` strategies exhibit higher overhead at 2 cores due to the fact

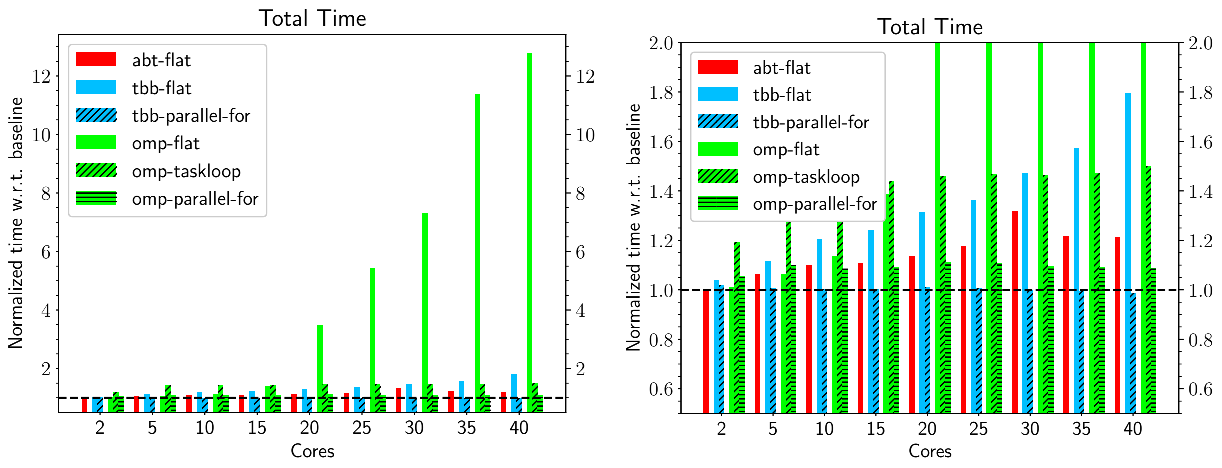
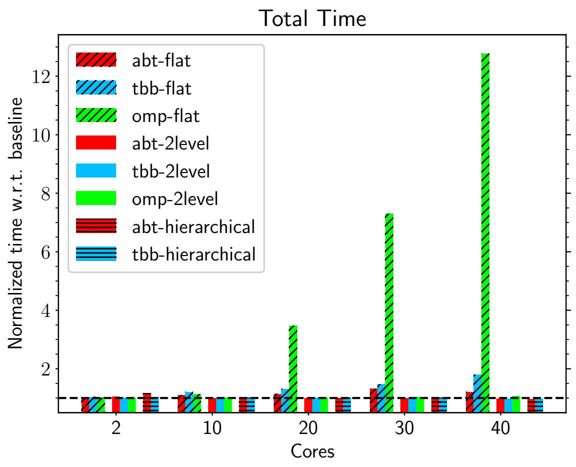


Fig. 66: Comparison of high level constructs and the `flat` model. Left: Normalized total running time of high level constructs and the `flat` model. Right: zoom-in at the range 0.5-2.0.

that more tasks are created in total. However, this overhead is amortized at a higher number of cores. The dual socket nature of the machine affects the system by a smaller amount, in comparison to the `flat` strategy, with the `omp` back-end suffering from the highest overhead at about 7% on 40 cores. On the other hand, the `abt` and `tbb` back-ends achieve a small improvement when using 40 cores.

The `hierarchical` task creation strategy creates tasks recursively by bisecting the `user_task_args` vector and creating two child tasks each time. The algorithm continues up until the target range reaches the grainsize, which is 1 in this dataset. `abt-hierarchical` and `tbb-hierarchical` exhibit a higher overhead at 2 cores possibly due to the larger number of generated tasks. For more than 2 cores, the `hierarchical` strategy performs slightly better than the `2-level`. This is attributed, in part, to the fact that the `hierarchical` strategy gives more flexibility in scheduling by having many smaller tasks running concurrently (versus the `2-level` which combines them in larger ones). This also creates more work-steal opportunities for idle threads, while at the same time avoids the overhead of creating tasks sequentially (contrary to the `flat` strategy). Results of applying the `hierarchical` strategy with a grainsize of 1 using the `omp` back-end are omitted due to their high overhead which reaches up to a 160x slowdown on 40 threads.



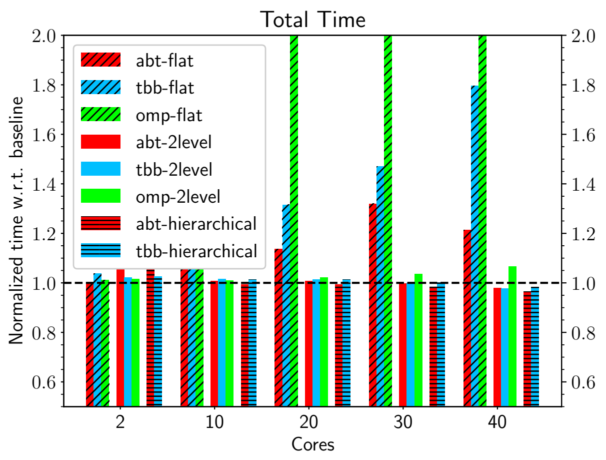


Fig. 67: Comparison of the three task creation strategies. Left: Normalized total running time of the three task creation strategies implemented across the three different back-ends. The `grainsize` is fixed to 1. Right: zoom-in at the range 0.6-2.0.

Effect of `grainsize` for each task creation strategy: In the next dataset, we demonstrate that the tasking framework in addition contributes towards automating the process of performance tuning. Since the tasking framework uses the same scheduler across the four different operations of this case-study, running the application repeatedly while scanning through a set of different `grainsize` values and the available back-ends, we can obtain optimal values for each operation.

The grainsize controls how many applications of *Operator* will be bundled into a single task. In general, creating a high number of tasks (smaller grainsize) gives more flexibility for load balancing by the scheduler. However, a high number of small tasks increases the cost of load balancing. Previous studies on *CDT3D* [85] revealed a significant dependence of the running time on the number of buckets created during local reconnection. In this study, instead of targeting a fixed number of buckets, we fix the size of each bucket to 150 tetrahedra which was found to be ideal for the baseline application. Figures 68, 69 and 70 compare the effect of different grainsizes for each operation using the `2-level` task creation strategy.

The running time in each case is normalized based on the time achieved using a fixed grainsize of 1. Overall, there are similar trends among the different back-ends. Point Creation and Local Reconnection perform better with a smaller grainsize. This is due to

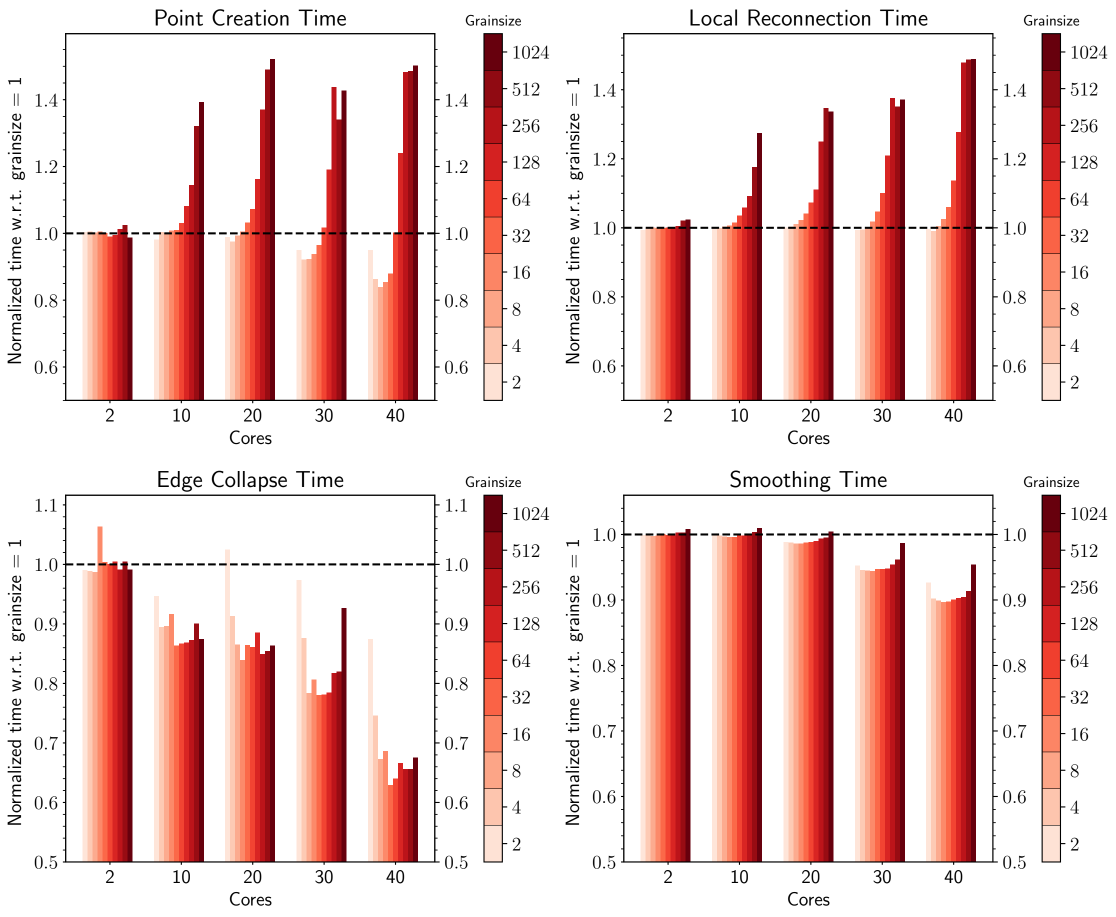


Fig. 68: Effect of grainsize for each operation for `omp-2-level`. Times are normalized based on the time taken using grainsize = 1.

the fact that these operations already decompose their data into "buckets" (see Table 33) and each "bucket" offers enough workload to amortize the cost of creating and handling tasks. Using a higher grainsize creates fewer tasks, thus constraining the load balancer and causes a loss in performance. On the other hand, Edge Collapse and Vertex Smoothing, where the *Operator* is designed to accept a single vertex, benefit significantly from increasing the grainsize. In particular, a grainsize of 128 for the Edge Collapse offers more than 30% speedup in comparison to a value of 1 for the `omp` back-end and about 20% for the other two back-ends. The gains for Vertex Smoothing are lower, but they also appear in the middle of the range which we experimented.

The same analysis was also performed for the `hierarchical` strategy. `abt-hierarchical`

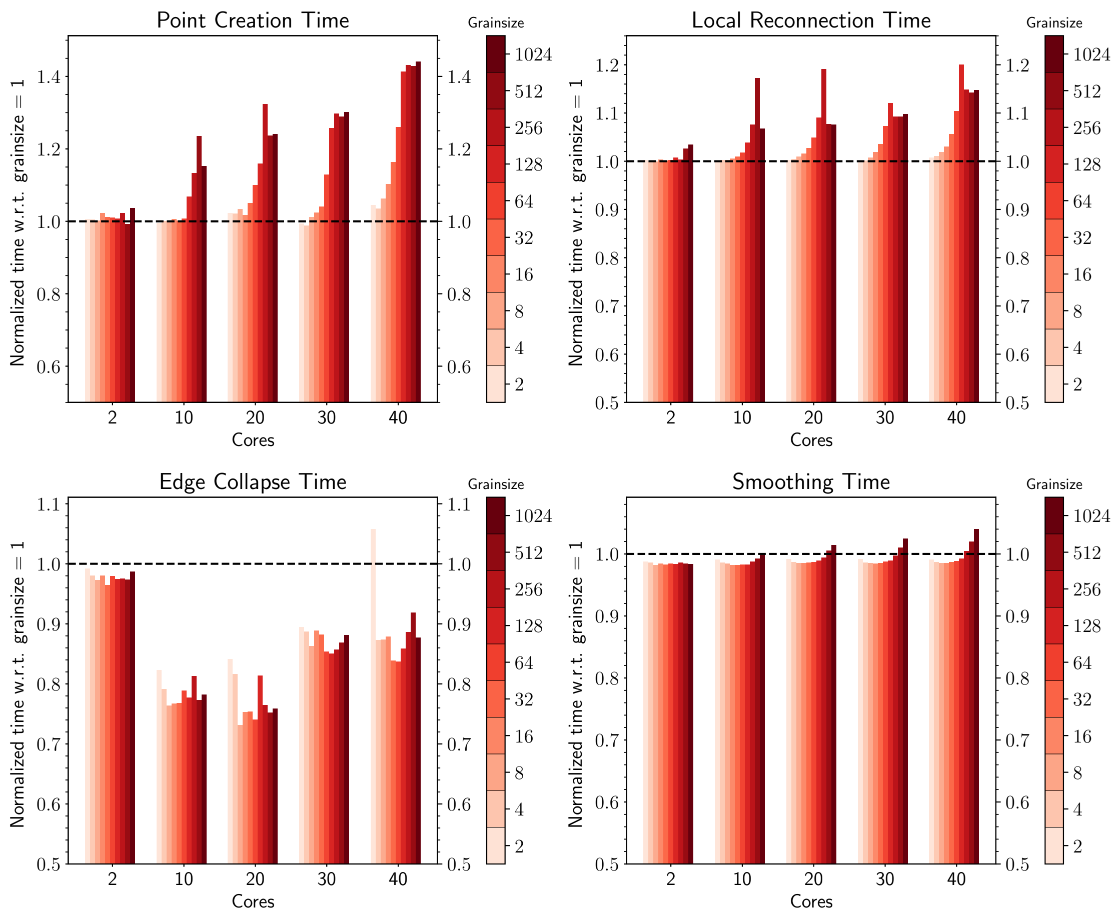


Fig. 69: Effect of grainsize for each operation for `tbb-2-level`. Times are normalized based on the time taken using grainsize = 1.

and `tbb-hierarchical` obtain optimal performance for the same grainsize values, while for `omp-hierarchical` the optimal values are 8192 for Edge Collapse and Vertex Smoothing, 32 for Vertex Creation and 1 for Local Reconnection. The graphs for the `hierarchical` strategy are omitted for brevity. It should be noted that these values may not be the ideal for a different configuration (hardware, meshing problem, etc.).

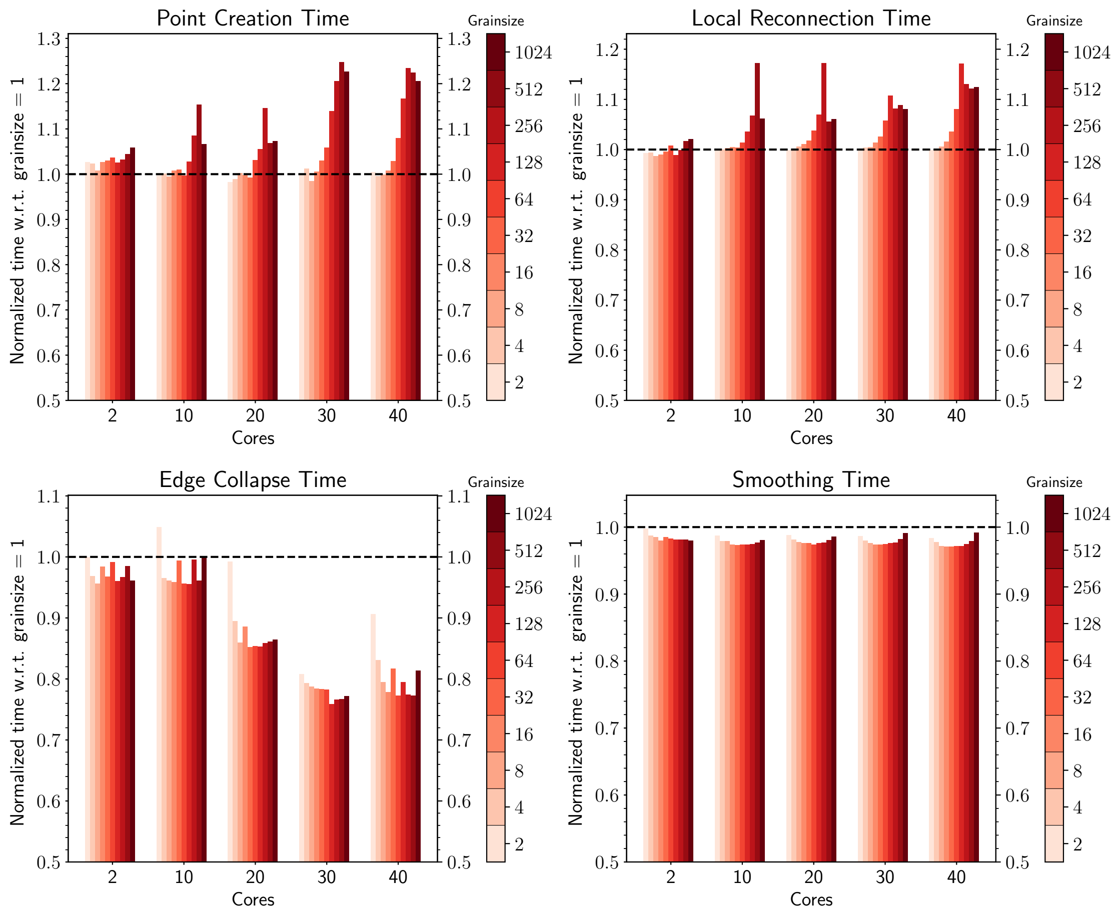
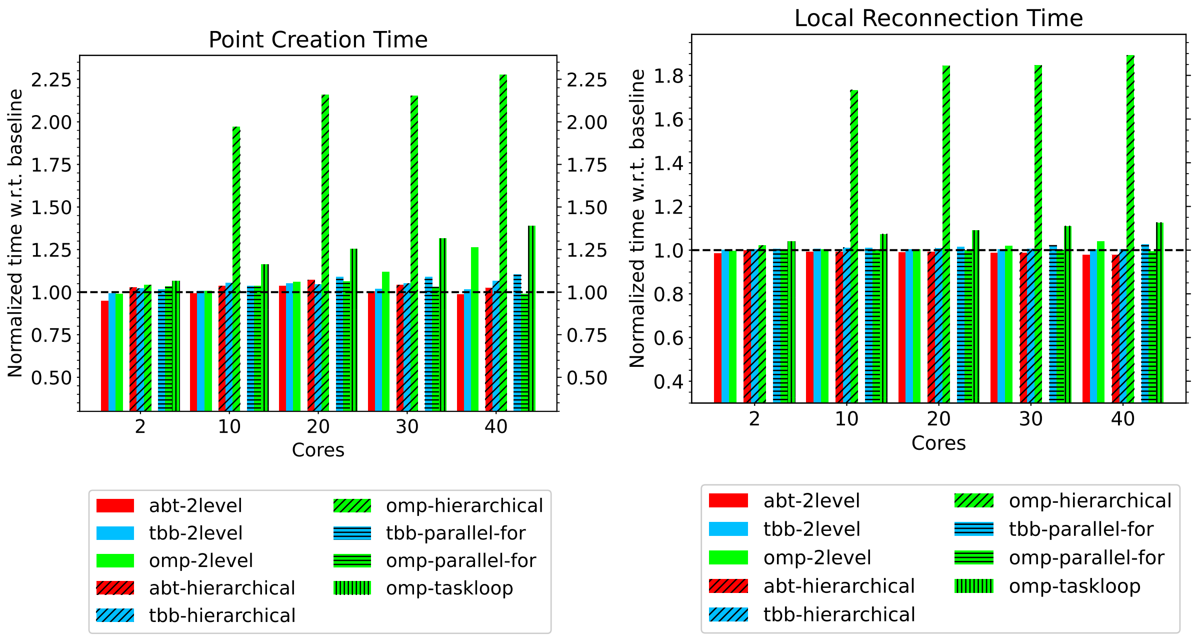


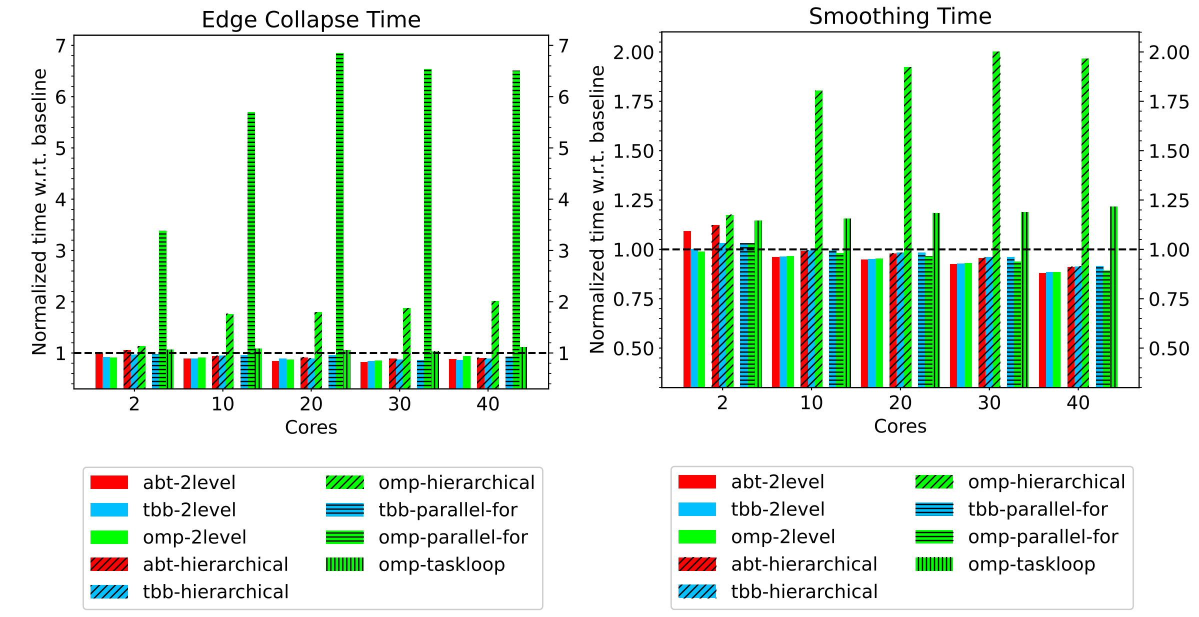
Fig. 70: Effect of grainsize for each operation for `abt-2-level`. Times are normalized based on the time taken using grainsize = 1.

Performance of `2-level` and `hierarchical` task creation strategies utilizing optimal `grainsize`: Finally, we compare the `2-level` and `hierarchical` task creation strategies utilizing the three different back-ends and the optimal `grainsize` values derived in the previous paragraph. The performance data indicate significant improvements for some back-ends especially for the least optimized mesh operations (Edge Collapse, Vertex Smoothing). Figure 71 depicts the performance gains replacing the baseline implementation with the tasking framework for each of the operations.

The grainsize is set to 1 for the Vertex Creation and Local Reconnection, 128 for Edge Collapse and 64 for Vertex Smoothing when utilizing `*-2-level`, `abt-hierarchical` or



(a) Percent (%) improvement over `baseline` for the Vertex Creation and Local Reconnection.



(b) Percent (%) improvement over `baseline` for the Edge Collapse and Smoothing Operations.

Fig. 71: Performance improvements over the baseline implementation for using the `2-level` and `hierarchical` strategies and optimal grainsizes.

`tbb-hierachical`. For `omp-hierarchical`, we used the grainsizes mentioned in the previous paragraph. Tables 34 and 35 present the percent (%) improvement over the baseline implementation.

	Point Creation			Local Reconnection		
	Cores			Cores		
	2	10	40	2	10	40
abt-2level	5.03	0.82	1.47	1.45	0.79	2.01
tbb-2level	0.68	-0.85	-1.69	-0.13	-0.66	-0.35
omp-2level	1.16	-0.81	-26.42	0.25	-0.49	-4.11
abt-hierarchical	2.69	-3.54	-2.47	-0.02	0.67	2.04
tbb-hierarchical	-2.22	-5.30	-6.65	-0.40	-1.20	-0.32
omp-hierarchical	-4.24	-97.11	-127.64	-2.19	-73.31	-89.30
tbb-parallel-for	-1.58	-3.79	-10.29	-0.59	-1.07	-2.78
omp-taskloop	-6.64	-16.30	-38.80	-4.07	-7.35	-12.67
omp-parallel-for	-3.39	-3.65	1.01	-0.22	-0.27	0.53

TABLE 34: Percent (%) improvement of running time with respect to the baseline implementation for the Point Creation and Local Reconnection operations.

The Point Creation and Local Reconnection operations benefit the least. The difference in performance gains between the two pairs of operations is related to the fact that not all operations use the same back-end in the baseline implementation (see Table 33). In particular, the Point Creation and Local Reconnection operations utilize a custom work-sharing approach described in [85] which has been optimized for the these operations. On the other hand, the Edge Collapse and Vertex Smoothing operation were parallelized using simple OpenMP primitives. Also, the first two operations operate on "buckets" (i.e., lists of elements) instead of single elements thus introducing *a-priori* data decomposition which may limit the effect of using different scheduling techniques.

`abt-2level` performs the best, offering up to 1.47% and 2.01% improvement on 40 cores for the Point Creation and Local Reconnection operations respectively. The Edge

	Edge Collapse			Smoothing		
	Cores			Cores		
	2	10	40	2	10	40
abt-2level	-0.46	10.88	12.05	-9.18	3.98	12.04
tbb-2level	7.74	10.60	13.42	-0.07	3.68	11.60
omp-2level	8.69	8.87	5.88	0.98	3.26	11.45
abt-hierarchical	-5.44	5.75	9.67	-12.32	0.77	8.92
tbb-hierarchical	3.02	5.26	10.58	-3.17	0.49	8.62
omp-hierarchical	-13.43	-76.16	-101.07	-17.42	-80.47	-96.60
tbb-parallel-for	2.28	3.86	6.86	-3.31	0.54	8.49
omp-taskloop	-6.53	-8.32	-11.44	-14.47	-15.59	-21.59
omp-parallel-for	-238.23	-469.65	-550.96	-3.16	1.89	10.67

TABLE 35: Percent (%) improvement of running time with respect to the baseline implementation of the Edge Collapse and Smoothing operations.

Collapse and Smoothing operations benefit more. `tbb-2level` performs the best for the Edge Collapse operation delivering more than 13% improvement on 40 cores while for Smoothing the best performing is `abt-2level` with up to 12% improvement over the baseline implementation. For comparison, we also append data from the higher-level constructs (`tbb::parallel_for`, `#pragma omp parallel for`, `#pragma omp taskloop`) (i.e., from Figure 66) which should serve as a reference point, since they provide the simplest way to introduce tasks within an application. The higher-level constructs fail to improve the performance for the operations that use custom scheduling and only some of them deliver small gains for Edge Collapse and Vertex Smoothing. In particular, `tbb::parallel_for` delivers improvements for Edge Collapse and Smoothing and `#pragma omp parallel for` exhibits some gains for Smoothing. However, the gains using the same back-ends within the proposed approach are higher.

Figure 72 and Table 36 depict the overall results while utilizing the optimal grainsize for each back-end and task creation strategy. Overall, `abt-2level` performs the best with up to 5.81% improvement on 40 cores. `tbb-2level` offers a slightly smaller improvement (4.72%) while `omp-2level` adds a small overhead (-0.52%) on 40 cores. Although, the `hierarchical` strategy is able to exploit concurrency at an earlier stage, it does not perform as well as the `2-level` strategy. This is attributed, in part, to the fact that the `2-level` strategy generates almost half the number of tasks in comparison to the `hierarchical` strategy. In particular, the `2-level` strategy generates $2 \cdot nthreads + n/grainsize$ tasks while the hierarchical generates $2^{\log_2(n/grainsize)+1} - 1 = 2(n/grainsize) - 1$ tasks where, n number of work-units passed to `task_for` (i.e., the length of vector `user_task_args` in Listing 5.1)[41]. Since, in general, $2 \cdot nthreads \ll n$ and $grainsize \ll n$, the `2-level` strategy produces about half the number of tasks.

	Total Time		
	Cores		
	2	10	40
abt-2level	-2.81	2.31	5.81
tbb-2level	-0.31	1.25	4.72
omp-2level	0.62	0.91	-0.52
abt-hierarchical	-15.86	0.91	4.39
tbb-hierarchical	-1.62	-0.31	2.79
omp-hierarchical	-21.0	-83.01	-99.71
tbb-parallel-for	-1.81	-0.30	1.40
omp-taskloop	-19.17	-42.88	-49.97
omp-parallel-for	-5.48	-8.57	-8.73

TABLE 36: Percent (%) improvement of total running time with respect to the baseline implementation.

[41] $h = \log_2(n/grainsize)$ is the depth of a perfect binary tree with $n/grainsize$ terminal nodes. $2^{h+1} - 1$ is the number of nodes for a perfect binary tree with depth h.

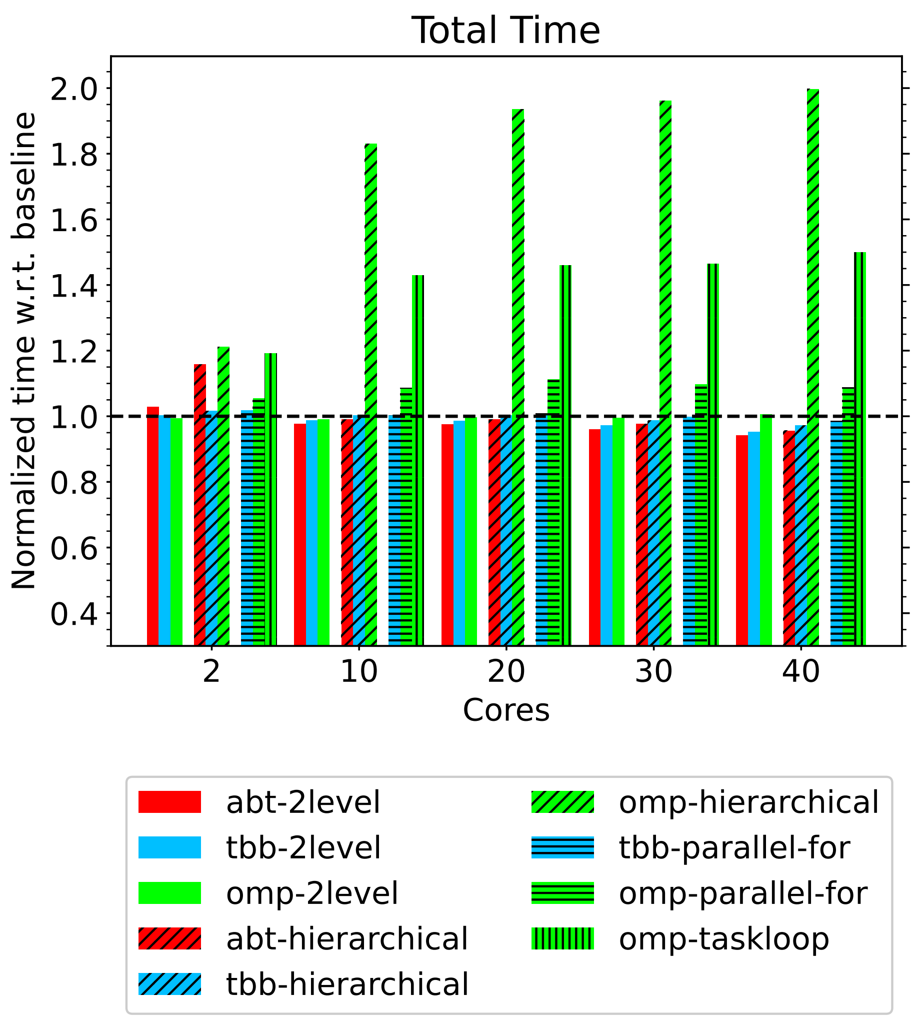
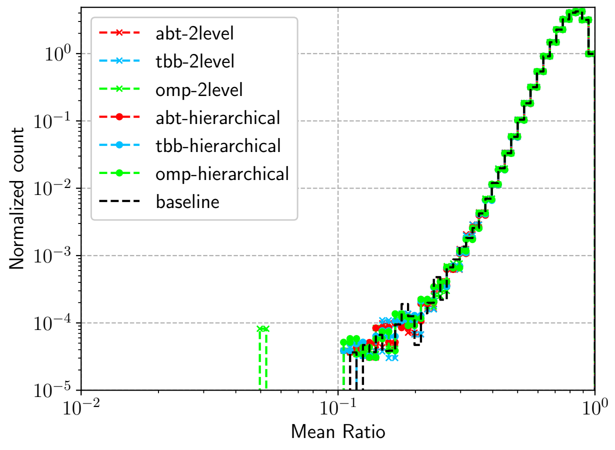


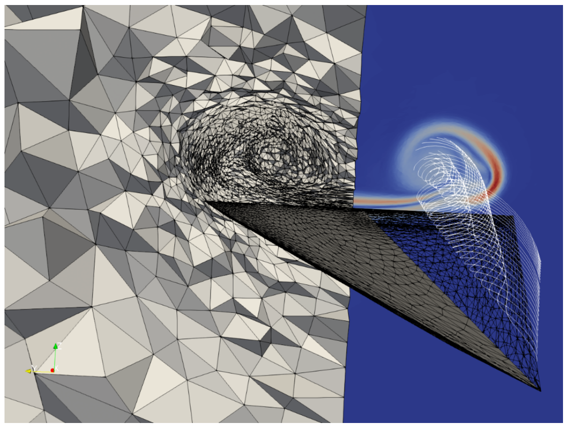
Fig. 72: Total running time with optimal grainsizes for each back-end and task creation strategy.

5.3.1.2 Stability of the Tasking Approach

Among the requirements for a parallel mesh generation code as presented in Section 2.1 is the one of *stability* which requires that a mesh generated in parallel has comparable quality with one generated sequentially by the same application. The stability of the baseline application has been already demonstrated in [251]. In Figure 73a, we compare a mesh quality measure among the different back-ends and task creation strategies of the previous section. In particular, the histograms are built using the meshes generated at 40 cores in Figure 71 and averaging the data over the 10 runs of the experiment. Even when using a logarithmic scale there is no significant difference between the different back-ends with the exception of the `omp-2level` back-end that produced slightly lower minimum value. Still, the results are within the range (> 0.01) produced by other state-of-the-art approaches as presented in [251].



(a)



(b)

Fig. 73: Stability data and visualization of the generated mesh of the *CDT3D* case-study. Left: Comparison of the mean-ratio quality metric for the different back-ends. Right: Visualization of the mesh generated by the experiments in this section: Metric-adapted mesh to a laminar flow over a delta wing.

5.3.2 CASE STUDY II: PARALLEL OPTIMISTIC DELAUNAY MESHING (*PODM*)

The Parallel Optimistic Delaunay Meshing (*PODM*) method presented in [95] delivers good parallel performance on DSM machines and high mesh quality along with provable fidelity guarantees. In terms of meshing operations, *PODM* initializes the meshing procedure with only 6 elements that decompose the bounding box of the input image. The mesh is incrementally refined by inserting points generated based on rules that guarantee the quality and fidelity of the mesh with respect to the input image. For more details, see Figure 63a. The point insertion procedure is built around the Bowyer-Watson kernel [35, 257] which introduces new points in the mesh while simultaneously preserving the invariant that after each point insertion, the mesh retains the Delaunay property. There are many ways to decompose the Bowyer-Watson kernel into tasks. In the past, it has been decomposed into *compute data dependencies (cavity)*, *collect data dependencies* and *update connectivity* tasks [60]. In higher dimensions (> 3), it is advantageous to decompose the data dependency evaluation (i.e., cavity expansion) into many tasks [94]. Other approaches [93, 176], transform the problem of Delaunay Mesh Refinement into two tasks: one of generating the vertices to be added and one that updates the current triangulation by inserting the vertices. In this study, in an effort to keep the problem complexity low and introduce only a small amount of code changes, only two types of tasks will be used; one for scheduling an element and one for refining it.

PODM caries many years of optimization for DSM machines [92]. However, as it happens with highly optimized codes, viewing them from a new perspective may reveal new challenges. The optimizations and design decisions that made *PODM* very efficient put constraints on the tasking implementation. The most important one, is that threading is managed explicitly by the application and the Load Balancing section of Figure 63a is responsible for populating the work-queue of each thread. In other words, the workload distribution is explicit and tightly integrated with the application.

To overcome this issue, we use the `thread_id` obtained by the threading environment in order to access the appropriate queue in a thread-safe manner. Listing 5.3 presents a high level pseudocode of the tasking version of *PODM*. It implements the `flat` model of Figure 64 by decomposing the algorithm of Figure 63a into two tasks. `ScheduleTask` creates tasks for a number of elements from a thread queue. Notice that the task created on line 19 can run with any `thread_id` which implicitly enables work distribution between different threads. Moreover, each thread will push the newly created elements into its private queue in line 31. `RefineBadElement` encapsulates the blue section of Figure 63a and it will generate the point to be inserted, calculate and lock its cavity (i.e., data dependencies) and apply the Bowyer-Watson kernel as well as release any acquired locks in the end.

Figure 74 depicts a high level view of the execution flow of the tasking version. Initially, `ScheduleTask` will span a task for each of the 6 elements of the initial mesh. Since the initial mesh is very small, only some of the initial 6 tasks will be completed successfully due to rollbacks. In the second round, `ScheduleTask` will create a task for each of the newly created elements and the process continues until all `thread_Queues` are empty. Notice that this simple implementation has two major issues: Possibility of livelocks, occurring when two tasks lock themselves in an infinite cycle trying to acquire different parts of overlapping cavities, and the algorithm termination depending on empty thread-local queues. Thread-local queues are accessed based on the `thread_id` acquired from the tasking environment which is, in general, random. Thus, there is a possibility that a non-empty `thread_Queue` may never get accessed. In these experiments, we didn't notice any of the aforementioned issues but there is still a chance that they might occur. In a follow-up study, we could integrate our previous work on contention managers [95] that can treat both issues efficiently.

Line 16 of Listing 5.3 includes a limit on the number of elements to be scheduled at a time. This is necessary since many of the generated tasks will be invalid by the time they run because their corresponding element will have been deleted as part of an operation executed on another cavity. Therefore, scheduling all available elements at once will generate a high number of aborted tasks.

```
1  main()
2  {
3    while(not all thread_Queues are empty){
4      // Launch enough ScheduleTasks to keep all cores
5      // busy
6      for(tid : thread_ids)
7        task::create_and_schedule_task(ScheduleTask);
8    task::wait_for_all();
9    }
10 }
11 ScheduleTask()
12 {
13   int tid = task::get_thread_id();
```

```
  int scheduled = 0;
  while(scheduled < schedule_limit &&
  !(thread_Queue[tid].empty()))
  {
    el = thread_Queue[tid].pop();
    task::create_and_schedule_task(el,RefineTask);
    scheduled++;
  }
}

RefineTask(el)
{
  int tid = task::get_thread_id();
  success = el.lock_vertices()
  if(success) {
    RefineBadElement(el);
    for(el : newly_created_elements)
      thread_Queue[tid].push(el)
  }
```

Listing 5.3: High level tasking-based pseudocode of *PODM*.

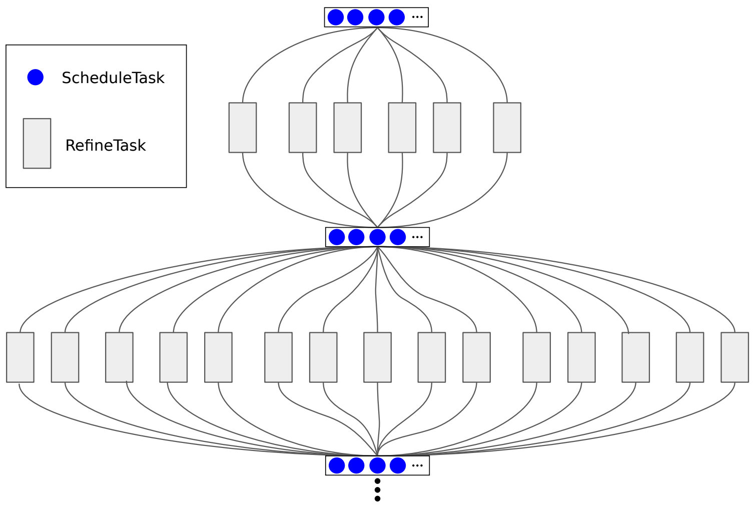


Fig. 74: Flowchart of the tasking version of *PODM*.

5.3.2.1 Performance Evaluation

The hardware and compiler configuration is the same as in Section 5.3.1.1. Figure 75 depicts the effect of the different values of `schedule_limit` to the runtime with respect to the baseline application. Notice that for 1 thread the ideal value is low. Any limit below 128 performs equally well, while for 40 a value of 512 performs better since it provides the system with more concurrency, albeit at the expense of more aborted tasks.

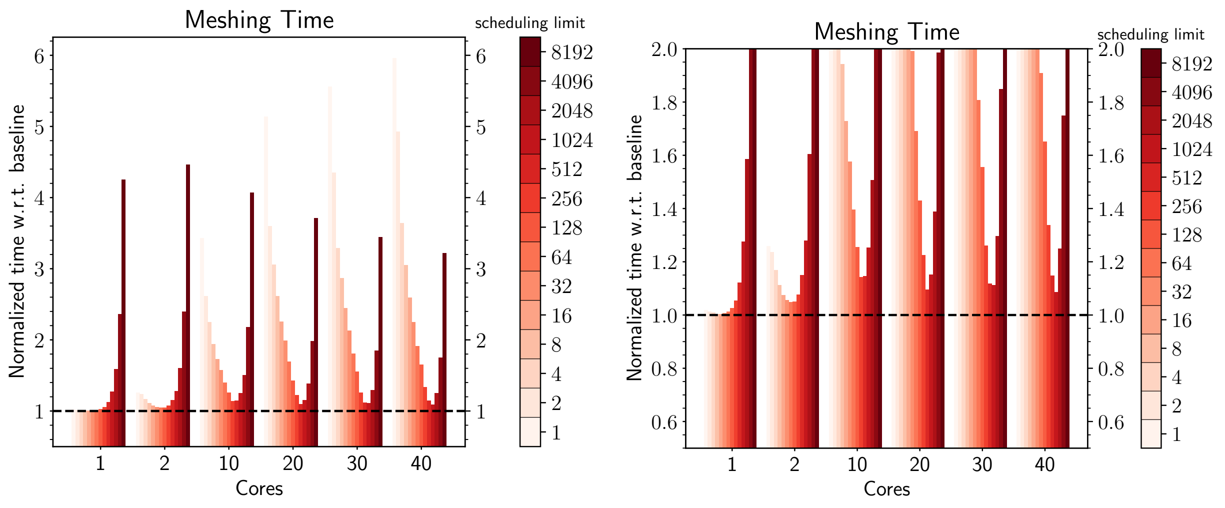


Fig. 75: Effect of scheduling limit using the `tbb` back-end. Right zoom-in in range 0.5-2.0.

Finally, Figure 76 presents the best values among our experiments. The use of the tasking framework adds only between $5\% - 14\%$ overhead with respect to the highly optimized baseline application across the different number of cores. Of course, these results come with the shortcomings mentioned above; but based on our previous experience, resolving them should not negatively impact the performance.

The `abt` and `omp` back-ends exhibit much higher overheads in this case. Scheduling decisions and the internal optimizations of `tbb` could be one of the reasons. Investigating the cause of this overhead and optimizing the `abt` and `omp` back-end implementations of the generalized framework could be investigated in the future.

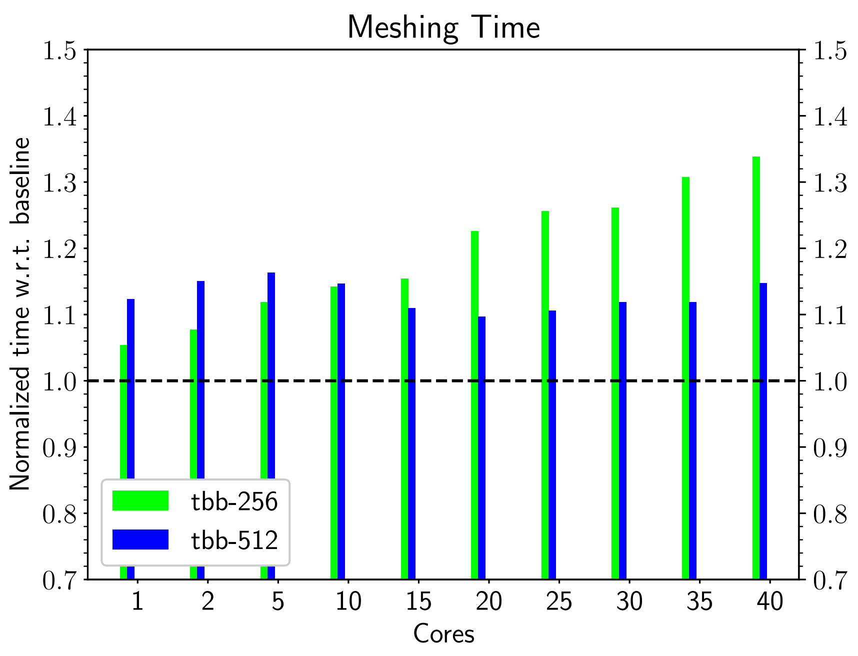
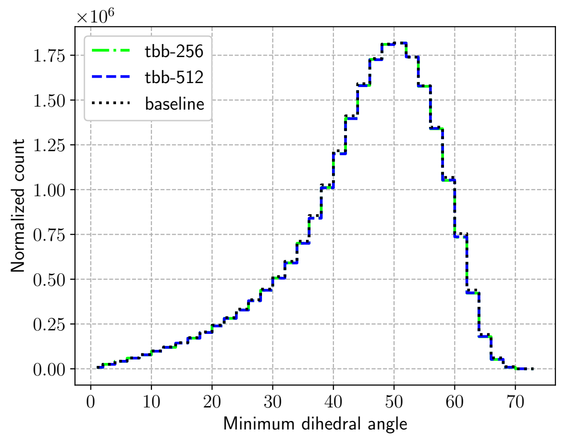


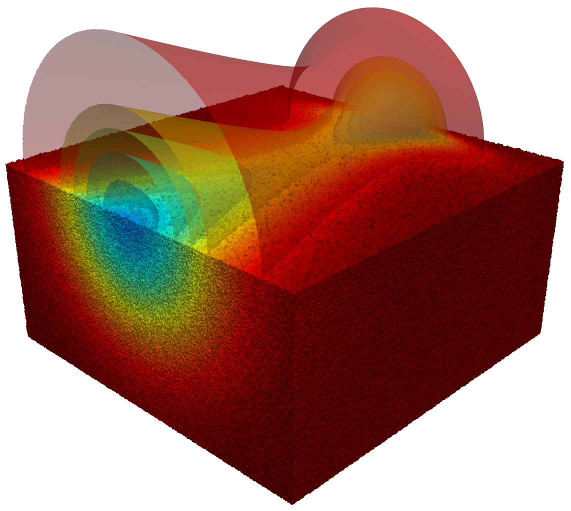
Fig. 76: Normalized meshing time of the tasking version of *PODM* for two different values of the `schedule_limit`.

5.3.2.2 Stability of the Tasking Approach

Similar to Section 5.3.1.2, Figure 77a compares a mesh quality measure (minimum dihedral angle in this case) among the different tasking approaches of this case-study. In order to satisfy the *stability* requirement the quality among the different execution back-ends should be comparable. The histograms of Figure 77a are built using the meshes generated at 40 cores in the previous section and averaging the data over the 10 runs of the experiment. The difference between the tasking approach and the baseline is marginal. The deviation from the baseline is lower that of the previous case-study due to the different meshing method used.



(a)



(b)

Fig. 77: Stability data and visualization of the generated mesh of the *PODM* case-study. (a): Comparison of the minimum dihedral angle between the different back-ends. (b): Visualization of the generated mesh along with contours of the dataset.

CHAPTER 6

CONCLUSION

In this work, we introduced a new parallel metric-based mesh adaptation method that can serve as the parallel optimistic mesh adaptation module of the *Telescopic Approach* in the context of CFD simulations. In particular, we extended the *CDT3D* library by adding new parallel mesh operations, incorporating metric adaptivity (Section 3.2) and the ability to interface with a CAD kernel (Section 3.3). The data of Section 3.4.4 indicate a well-optimized implementation attaining more than 92% end-to-end efficiency on a single node. Our implementation scales well within the shared node in comparison to state-of-the-art methods as it is indicated by Section 4.1.3.1. Although, it falls behind in pure speed when compared to well-optimized industrial codes such as *Feflo.a* from INRIA (see for example Table 29), it exhibits better performance in terms of weak scalability. Moreover, it has a faster sequential and shared-memory implementation than the distributed-memory-optimized *refine*. In terms of quality, our results indicate that our method produces meshes of comparable quality to state-of-the-art methods for a variety of configurations covering: (i) analytically prescribed metric fields over both planar (Section 4.1.1) and curved (Section 4.1.2) domains, (ii) solution-derived metric fields specified over planar geometries (Section 4.1.3) and in the presence of CAD data (Section 4.1.4). Moreover, our data in Section 4.2 suggest that *CDT3D* can be used in metric-based adaptive pipelines effectively.

Finally, our tasking framework presented in Chapter 5 was successfully applied to a number of different meshing operations employing a speculative approach. The results of Section 5.3.1 indicate performance improvements for *CDT3D*. These improvements are significantly higher when compared to the straight-forward use of tasks which can lead up to 1200% slowdown for `omp-flat` in Figure 67 and higher than higher-level constructs that are already present in some of the back-ends (`#pragma omp parallel for`, `#pragma omp taskloop`, `tbb::parallel_for`). In particular, Tables 34 and 35 indicate performance improvements of up to 13% for some meshing operations while, the results in Table 36 suggest an improvement of up to 5.81% over the entire end-to-end application without any compromise over the correctness and quality of the result. Moreover, the abstract front-end gives a platform to explore multiple execution back-ends; Figures 66 and 67 show results over 12 different `strategy-backend` combinations which are accessible to the application developer through a compile-time parameter. More importantly, it manages to separate *functionality*

and *performance*; a crucial step to the implementation of the *Telescopic Approach*. The generalized tasking framework facilitates the integration with the PREMA runtime system [244] at the shared memory level by handing control of thread management and load balancing from the application to the runtime system. This decoupling is expected to speedup the implementation due to the improved encapsulation of the different methods and PREMA's more efficient management of hardware resources.
Revisiting the attributes we set in the introduction we have:

1. **Stability:** The stability of the parallel metric-adaptive method is studied and compared to state-of-the-art methods in Section 4.1.3 and is summarized in Figure 35. The quality of the generated mesh in parallel is comparable to the one generated sequentially. Moreover, the stability holds even within the tasking framework as Section 5.3.1.2 suggests.

2. **Reproducibility:** Our method offers weak reproducibility which is sufficient for most flow solvers and adaptive processes. Indeed, the adaptive pipeline of Section 4.2 restarts the computation at each iteration and *CDT3D* is forced to reconstruct its internal data-structures.

3. **Robustness:** Interfacing with a CAD kernel allows to accept a new class of inputs that was impossible before. We cannot assert that our method offers industrial strength robustness for CAD data, we have however introduced a flexible interface that can handle inputs of varying complexity as Sections 4.1.4, 4.2.3 and 4.2.4 indicate.

4. **Scalability:** The strong and weak scaling data of Section 4.1.3 indicate that the performance of our implementation for a single node is comparable to state-of-the-art methods. Moreover, the parallel efficiency results of Section 3.4.4 indicate promising scalability at higher number of cores in the context of the *Telescopic Approach*. Also, based on the data of Table 32 the time required by *CDT3D* is only fraction of the time of an adaptive iteration.

5. **Code Reuse:** Our tasking framework improves the encapsulation by hiding thread management and load balancing away from the developer. Moreover, it offers easy access and in a portable manner to a plethora of back-end and tasking strategies.

In summary, the goal of this dissertation was to create a new parallel anisotropic mesh adaptation method that can serve as building block for scalable parallel mesh generation.

Our method is designed to be the speculative tightly-coupled parallel mesh adaptation component of the *Telescopic Approach*, that exploits concurrency at the chip level. In this dissertation we have demonstrated the following contributions:

(C_1) A parallel mesh adaptation method with high parallel efficiency on a single multi-core node (Sections 3.4.4 and 4.1.3).

(C_2) The method exhibits comparable quality and performance against state-of-the-art methods (Section 4.1).

(C_3) The parallel mesh adaptation method can interface with a CAD kernel allowing to accept a wider variety of inputs (Sections 4.1.4, 4.2.3 and 4.2.4).

(C_4) Validation of the method within an adaptive pipeline (Section 4.2).

(C_5) A General Tasking Framework that aids towards separating the concerns of functionality from performance for speculative parallel mesh generation methods (Section 5).

CHAPTER 7

FUTURE WORK

CDT3D is a shared memory software and it is designed to be a building block of the *Telescopic Approach* for scalable parallel mesh generation and adaptivity. The next layer is the Parallel Data Refinement layer which has been already implemented with both Delaunay-based [89] and Advancing-front based methods [106]. Based on our group's previous experience it is expected to achieve good scalability by combining the two approaches. There are however challenges towards this direction related to minimizing the amount data movement as identified by [88] and reducing the effect of the constrained faces to the final mesh quality as discussed in [105]. An alternative approach would be to revisit the distributed speculative approach presented in [187] and adapt it for *CDT3D*. In [187] a cavity could expand across many subdomains due to the unconstrained nature of the method. In *CDT3D* however, a cavity of a vertex (for smoothing and edge collapse) contains only elements attached to the vertex itself, and a cavity of a flip or a point insertion operation contains face/edge neighbors of a tetrahedron. In other words, all types of cavities include only immediate neighbors derived by vertex/edge/face adjacency. This observation with the appropriate software modifications could be used to create a pre-processing step that locks the interface vertices of a subdomain and implicitly the elements attached to them (see for example Figure 16). Such a pre-processing should cause *CDT3D* to avoid these elements and thus artificially constrain the modifications within the subdomain. Interface elements could then be exchanged using one of the element migration techniques utilized by the partially coupled methods that employ discrete domain decomposition discussed in Section 2.1.3.2.

The generalized tasking framework facilitates the integration with the PREMA runtime system [244] at the shared memory level by handing control of thread management and load balancing from the application to the runtime system. This decoupling is expected to speedup the implementation due to the improved encapsulation of the different methods and PREMA's more efficient management of hardware resources. For example, the application independent tasking pools that the tasking framework offers can provide load balancing across different instances of the same application occupying a common shared memory space. This scenario fits well with our previous work [89] and the Parallel Data Refinement layer of the *Telescopic Approach*.

The vertex smoothing operation can be further improved by incorporating a more complete search space for the optimal node position such as the methods presented in [97, 140].

Moreover, it could be extended to all vertices and not just the ones attached to low quality elements providing an overall smoothed result which may provide better convergence rates for the solver. Also, CAD information such as local curvature and local feature size could be incorporated in order to optimize the quality of curved surfaces.

GPUs (Graphic Processing Units) are common in today's supercomputers however, currently, *CDT3D* make no use of them. Extending the presented meshing operations so that they can take advantage of the accelerators is expected to improve the running speed of certain operations significantly. Figure 18 reveals that almost 95% of the total time is spent on just two operations: the local reconnection and the vertex smoothing operation. Although they both exhibit more than 90% efficiency, they can still be improved by the use of accelerators. In particular, porting the inner floating-point-heavy kernels such as the predicates of the Delaunay criterion and the min-max edge-weight measure to GPUs could potentially reduce the running time significantly. Such a transition is not straightforward since most of the data-structures of *CDT3D* are pointer-based. However, extracting the combinatorial steps of the flip operations and the line search of the vertex smoothing into asynchronous tasks that can be executed on the GPU could offer a compromise between keeping the current data structures and taking advantage of the GPUs. Also, in the context of the tasking framework there is a number of back-end systems that can utilize both homogeneous and heterogeneous platforms including GPUs. Generic heterogeneous frameworks such as SYCL[42] and Kokkos[43] provide already support for launching and managing tasks on GPUs. Combining them with the tasking framework is expected to assist in hiding latencies related to data transfers to and from the device as well as delays launching kernels. Evaluating such a framework would also require the addition or extension of current mesh operations for heterogeneous architectures.

Boundary layer mesh generation: As mentioned in Section 4.2 we didn't succeed in obtaining viscous results with our pipeline. This is in part attributed to the absence of boundary layer mesh that many numerical methods expect. Although, fully unstructured results have been reported with other solvers (see for example [201]), to the best of our knowledge, SU2 has not been tested thoroughly within this context. Boundary layer can be provided as an external procedure and integrated similarly to our work in [267]. A similar approach that combines state-of-the-art boundary layer generation with adaptive

[42]`https://www.khronos.org/sycl` (Accessed 2021-05-27).
[43]`https://kokkos.org` (Accessed 2021-05-27).

anisotropic method appears in [171]. Another path to explore is the generation of metric-aligned meshes such as the ones presented in [160, 170]. Moreover, the use of specialized metrics may also be suitable. For example, in [242] the authors report success by combining wall-distance to their metric creation while, in [90] the authors review various output-based metric construction schemes that could be evaluated.

Regarding our tasking approach, both case studies introduce a parameter (*grainsize* for *CDT3D* and *schedule_limit* for PODM) that controls the number of tasks created. Analyzing the effect of this parameter in an application-independent manner allows to optimize the parallel execution with no deep knowledge of the application. Still, in this study the mesh size was constant and thus a more thorough study is needed in order to identify the optimal values. As with any parameter optimization study, this could be performed utilizing machine learning on pre-generated data. Moreover, the underlying tasking framework could be equipped with an online machine learning method [124] that can choose the optimal parameters based on runtime data.

Utilizing tasks in conjunction with the speculative approach could be further improved by abstracting the Parallel Correctness sections of Figure 63. This could be achieved with high level (but still application-specific) abstractions that lock and unlock the cavity of an operation automatically. Also, aborted (due to rollbacks) tasks could be captured by the framework and rescheduled (if they are still applicable).

The back-end of the `task_for` front-end of Listing 5.1 is currently a compile-time parameter. However, there is no technical constrain that would prevent it from being an extra argument of the front-end API. Providing the back-end as a parameter would allow to combine the benefits of each back-end based on the operation being parallelized. For example, although the accumulative gains of `abt` back-end in Table 36 are higher that the rest of the back-ends, the data of Table 35 suggest that for certain operations (Edge Collapse) the `tbb` back-end performs better. Giving the user the ability to select the appropriate back-end at each `task_for` invocation would allow to get the best performance at each operation.

REFERENCES

[1] F. Alauzet, "Adaptation de maillage anisotrope en trois dimensions. Application aux simulations instationnaires en mécanique des fluides," Ph.D. dissertation, Université Montpellier II, Montpellier , France, 2003. https://tel.archives-ouvertes.fr/tel-00363511

[2] F. Alauzet, "Size gradation control of anisotropic meshes," *Finite Elements in Analysis and Design*, vol. 46, no. 1–2, pp. 181–202, Jan. 2010, DOI: 10.1016/j.finel.2009.06.028

[3] F. Alauzet, "A changing-topology moving mesh technique for large displacements," *Engineering with Computers*, vol. 30, no. 2, pp. 175–200, Apr. 2014, DOI: 10.1007/s00366-013-0340-z

[4] F. Alauzet, X. Li, E. S. Seol, and M. S. Shephard, "Parallel anisotropic 3D mesh adaptation by mesh modification," *Engineering with Computers*, vol. 21, no. 3, pp. 247–258, May 2006, DOI: 10.1007/s00366-005-0009-3

[5] F. Alauzet and A. Loseille, "High-order sonic boom modeling based on adaptive methods," *Journal of Computational Physics*, vol. 229, no. 3, pp. 561–593, 2010, DOI: 10.1016/j.jcp.2009.09.020

[6] F. Alauzet and A. Loseille, "A decade of progress on anisotropic mesh adaptation for computational fluid dynamics," *Computer-Aided Design*, vol. 72, pp. 13–39, Mar. 2016, DOI: 10.1016/j.cad.2015.09.005

[7] S. Aldea, A. Estebanez, D. R. Llanos, and A. Gonzalez-Escribano, "An OpenMP extension that supports thread-level speculation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 78–91, 2016, DOI: 10.1109/TPDS.2015.2393870

[8] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485, DOI: 10.1145/1465482.1465560

[9] American National Standards Institute, "Initial graphics exchange specification iges 5.3 (ansi-1996)," ANS US PRO/IPO-100-1996, Sep. 1996.

[10] J. D. Anderson, *Computational fluid dynamics the basics with applications*, ser. McGraw-Hill series in mechanical engineering. New York: McGraw-Hill, 1994. ISBN 0-07-001685-2

[11] F. Angrand, A. Dervieux, V. Billey, J. Periaux, and C. Pouletty, "2-D and 3-D Euler flow calculations with a second-order accurate Galerkin finite element method," in *18th Fluid Dynamics and Plasmadynamics and Lasers Conference*. American Institute of Aeronautics and Astronautics, 1985, DOI: 10.2514/6.1985-1706

[12] C. D. Antonopoulos, F. Blagojevic, A. N. Chernikov, N. P. Chrisochoides, and D. S. Nikolopoulos, "A multigrain Delaunay mesh generation method for multicore SMT-based architectures," *Journal of Parallel and Distributed Computing*, vol. 69, no. 7, pp. 589–600, Jul. 2009, DOI: 10.1016/j.jpdc.2009.03.009

[13] C. D. Antonopoulos, X. Ding, A. Chernikov, F. Blagojevic, D. S. Nikolopoulos, and N. Chrisochoides, "Multigrain parallel delaunay mesh generation: Challenges and opportunities for multithreaded architectures," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ser. ICS '05. ACM, 2005, pp. 367–376. ISBN 1-59593-167-8 DOI: 10.1145/1088149.1088198

[14] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, "Log-Euclidean metrics for fast and simple calculus on diffusion tensors," *Magnetic Resonance in Medicine*, vol. 56, no. 2, pp. 411–421, Aug. 2006, DOI: 10.1002/mrm.20965

[15] S. B. Baden, D. B. Gannon, M. L. Norman, and N. P. Chrisochoides, *Structured Adaptive Mesh Refinement (Samr) Grid Methods*. Berlin, Heidelberg: Springer-Verlag, 1999. ISBN 978-0-387-98921-1 DOI: 10.1007/978-1-4612-1252-2

[16] T. J. Baker, "Mesh adaptation strategies for problems in fluid dynamics," *Finite Elements in Analysis and Design*, vol. 25, no. 3, pp. 243–273, Apr. 1997, DOI: 10.1016/S0168-874X(96)00032-7

[17] A. Balan, M. A. Park, S. Wood, and W. K. Anderson, "Verification of Anisotropic Mesh Adaptation for Complex Aerospace Applications," in *AIAA Scitech 2020 Forum*, ser. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2020, DOI: 10.2514/6.2020-0675

[18] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley,

D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, “PETSc users manual,” https://www.mcs.anl.gov/petsc, Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.15, 2021.

[19] K. Barker, A. Chernikov, N. Chrisochoides, and K. Pingali, “A load balancing framework for adaptive and asynchronous applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 183–192, Feb. 2004, DOI: 10.1109/TPDS.2004.1264800

[20] N. Barral, “Time-accurate anisotropic mesh adaptation for three-dimensional moving mesh problems,” Ph.D. dissertation, UPMC, Nov. 2015. https://hal.inria.fr/tel-01284113

[21] N. Barral, M. G. Knepley, M. Lange, M. D. Piggott, and G. J. Gorman, “Anisotropic mesh adaptation in Firedrake with PETSc DMPlex,” in *25th International Meshing Roundtable*, 2016, Research Note.

[22] T. J. Barth, “Numerical aspects of computing high Reynolds number flows on unstructured meshes,” in *29th Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 1991, DOI: 10.2514/6.1991-721

[23] V. H. F. Batista, D. L. Millman, S. Pion, and J. Singler, “Parallel geometric algorithms for multi-core computers,” *Computational Geometry*, vol. 43, no. 8, pp. 663–677, Oct. 2010, DOI: 10.1016/j.comgeo.2010.04.008

[24] D. Benítez, E. Rodríguez, J. M. Escobar, and R. Montenegro, “Performance Evaluation of a Parallel Algorithm for Simultaneous Untangling and Smoothing of Tetrahedral Meshes,” in *Proceedings of the 22nd International Meshing Roundtable*, J. Sarrate and M. Staten, Eds. Springer International Publishing, 2014, pp. 579–598. ISBN 978-3-319-02335-9 DOI: 10.1007/978-3-319-02335-9_32

[25] M. J. Berger and J. Oliger, “Adaptive mesh refinement for hyperbolic partial differential equations,” *Journal of Computational Physics*, vol. 53, no. 3, pp. 484–512, Mar. 1984, DOI: 10.1016/0021-9991(84)90073-1

[26] B. Berthou, D. Binosi, N. Chouika, L. Colaneri, M. Guidal, C. Mezrag, H. Moutarde, J. Rodríguez-Quintero, F. Sabatié, P. Sznajder, and J. Wagner, “PARTONS: PARtonic Tomography Of Nucleon Software,” *The European Physical Journal C*, vol. 78, no. 6, p. 478, Jun. 2018, DOI: 10.1140/epjc/s10052-018-5948-0

[27] D. K. Blandford, G. E. Blelloch, and C. Kadow, "Engineering a Compact Parallel Delaunay Algorithm in 3D," in *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, ser. SCG '06. New York, NY, USA: ACM, 2006, pp. 292–300. ISBN 978-1-59593-340-9 DOI: 10.1145/1137856.1137900

[28] G. E. Blelloch, D. Anderson, and L. Dhulipala, "ParlayLib - A Toolkit for Parallel Algorithms on Shared-Memory Multicore Machines," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 507–509. ISBN 978-1-4503-6935-0 DOI: 10.1145/3350755.3400254

[29] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and J. Shun, "Internally deterministic parallel algorithms can be fast," in *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '12. New York, NY, USA: Association for Computing Machinery, Feb. 2012, pp. 181–192. ISBN 978-1-4503-1160-1 DOI: 10.1145/2145816.2145840

[30] G. E. Blelloch, G. L. Miller, and D. Talmor, "Developing a Practical Projection-based Parallel Delaunay Algorithm," in *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, ser. SCG '96. New York, NY, USA: ACM, 1996, pp. 186–195. ISBN 978-0-89791-804-6 DOI: 10.1145/237218.237357

[31] H. Blum, *A Transformation for Extracting New Descriptors of Shape.* Cambridge: MIT Press, 1967, pp. 362–380. ISBN 978-0262230261

[32] J.-D. Boissonnat, K.-L. Shi, J. Tournois, and M. Yvinec, "Anisotropic Delaunay Meshes of Surfaces," *ACM Transactions on Graphics*, vol. 34, no. 2, pp. 14:1–14:11, Mar. 2015, DOI: 10.1145/2721895

[33] H. Borouchaki, P. L. George, F. Hecht, P. Laug, and E. Saltel, "Delaunay mesh generation governed by metric specifications. Part I. Algorithms," *Finite Elements in Analysis and Design*, vol. 25, no. 1, pp. 61–83, Mar. 1997, DOI: 10.1016/S0168-874X(96)00057-1

[34] Y. Bourgault, M. Picasso, F. Alauzet, and A. Loseille, "On the use of anisotropic a posteriori error estimators for the adaptative solution of 3D inviscid compressible flows," *International Journal for Numerical Methods in Fluids*, vol. 59, no. 1, pp. 47–74, 2009, DOI: 10.1002/fld.1797

[35] A. Bowyer, "Computing Dirichlet tessellations," *The Computer Journal*, vol. 24, no. 2, pp. 162–166, Jan. 1981, DOI: 10.1093/comjnl/24.2.162

[36] B. Bramas, "Increasing the degree of parallelism using speculative execution in task-based runtime systems," *PeerJ Computer Science*, vol. 5, p. e183, Mar. 2019, publisher: PeerJ Inc. DOI: 10.7717/peerj-cs.183

[37] J. H. Bucklow, R. Fairey, and M. R. Gammon, "An automated workflow for high quality CFD meshing using the 3D medial object," in *23rd AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2017, DOI: 10.2514/6.2017-3454

[38] W. R. Buell and B. A. Bush, "Mesh Generation - A Survey," *Journal of Engineering for Industry*, vol. 95, no. 1, pp. 332–338, Feb. 1973, DOI: 10.1115/1.3438132

[39] C. Burstedde, L. Wilcox, and O. Ghattas, "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1103–1133, Jan. 2011, DOI: 10.1137/100791634

[40] J. M. M. Caamaño, A. Sukumaran-Rajam, A. Baloian, M. Selva, and P. Clauss, "APOLLO: Automatic speculative POLyhedral Loop Optimizer," in *IMPACT 2017 - 7th International Workshop on Polyhedral Compilation Techniques*, Stockholm, Sweden, Jan. 2017, p. 8.

[41] P. C. Caplan, R. Haimes, D. L. Darmofal, and M. C. Galbraith, "Anisotropic geometry-conforming d-simplicial meshing via isometric embeddings," *Procedia Engineering*, vol. 203, pp. 141–153, Jan. 2017, DOI: 10.1016/j.proeng.2017.09.798

[42] J. G. Castaños and J. E. Savage, "Parallel refinement of unstructured meshes," in *Proc. of IASTED International Conference Parallel and Distributed Computing and Systems*, Boston,MA, 1999.

[43] P. A. Cavallo, N. Sinha, and G. M. Feldman, "Parallel Unstructured Mesh Adaptation Method for Moving Body Applications," *AIAA Journal*, vol. 43, no. 9, pp. 1937–1945, Sep. 2005, DOI: 10.2514/1.7818

[44] W. M. Chan and J. L. Steger, "Enhancements of a three-dimensional hyperbolic grid generation scheme," *Applied Mathematics and Computation*, vol. 51, no. 2, pp. 181–205, 1992, DOI: 10.1016/0096-3003(92)90073-A

[45] S. Chandra, X. Li, T. Saif, and M. Parashar, "Enabling scalable parallel implementations of structured adaptive mesh refinement applications," *The Journal of Supercomputing*, vol. 39, no. 2, pp. 177–203, Feb. 2007, DOI: 10.1007/s11227-007-0110-z

[46] D. Chase and Y. Lev, "Dynamic circular work-stealing deque," in *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 21–28. ISBN 1581139861 DOI: 10.1145/1073970.1073974

[47] J. Chen, Z. Xiao, Y. Zheng, J. Zou, D. Zhao, and Y. Yao, "Scalable generation of large-scale unstructured meshes by a novel domain decomposition approach," *Advances in Engineering Software*, vol. 121, pp. 131–146, Jul. 2018, DOI: 10.1016/j.advengsoft.2018.04.005

[48] A. Chernikov and N. Chrisochoides, "Parallel Guaranteed Quality Delaunay Uniform Mesh Refinement," *SIAM Journal on Scientific Computing*, vol. 28, no. 5, pp. 1907–1926, Jan. 2006, DOI: 10.1137/050625886

[49] A. Chernikov, C. Antonopoulos, N. Chrisochoides, S. Schneider, and D. Nikolopoulos, "Experience with Memory Allocators for Parallel Mesh Generation on Multicore Architectures," in *International Conference on Numerical Grid Generation in Computational Field Simulations*, Forth, Crete, Greece, Sep. 2007.

[50] A. N. Chernikov and N. P. Chrisochoides, "Practical and Efficient Point Insertion Scheduling Method for Parallel Guaranteed Quality Delaunay Refinement," in *Proceedings of the 18th Annual International Conference on Supercomputing*, ser. ICS '04. New York, NY, USA: ACM, 2004, pp. 48–57. ISBN 1-58113-839-3 DOI: 10.1145/1006209.1006217

[51] A. N. Chernikov and N. P. Chrisochoides, "Parallel 2D Graded Guaranteed Quality Delaunay Mesh Refinement," in *Proceedings of the 14th International Meshing Roundtable*, B. W. Hanks, Ed. Springer Berlin Heidelberg, 2005, pp. 505–517. ISBN 978-3-540-29090-2 DOI: 10.1007/3-540-29090-7_30

[52] A. N. Chernikov and N. P. Chrisochoides, "Algorithm 872: Parallel 2d constrained delaunay mesh generation," *ACM Transactions on Mathematical Software*, vol. 34, no. 1, pp. 6:1–6:20, 2008, DOI: 10.1145/1322436.1322442

[53] A. N. Chernikov and N. P. Chrisochoides, "Three-dimensional Delaunay Refinement for Multi-core Processors," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 214–224. ISBN 978-1-60558-158-3 DOI: 10.1145/1375527.1375560

[54] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering," *Parallel Computing*, vol. 34, no. 6, pp. 318–331, Jul. 2008, DOI: 10.1016/j.parco.2007.12.001

[55] L. P. Chew, Nikos Chrisochoides, and F. Sukup, "Parallel constrained Delaunay meshing," *ASME APPLIED MECHANICS DIVISION-PUBLICATIONS-AMD*, vol. 220, pp. 89–96, 1997. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.8836&rep=rep1&type=pdf (Accessed 2015-05-13).

[56] N. Chrisochoides, K. Barker, D. Nave, and C. Hawblitzel, "Mobile object layer: a runtime substrate for parallel adaptive and irregular computations," *Advances in Engineering Software*, vol. 31, no. 8-9, pp. 621–637, 2000, DOI: 10.1016/S0965-9978(00)00032-6

[57] N. Chrisochoides, *Numerical Solution of Partial Differential Equations on Parallel Computers*, ser. Lecture Notes in Computational Science and Engineering. Springer-Verlag, 2006, vol. 51, ch. Parallel Mesh Generation, pp. 237–264, DOI: 10.1007/3-540-31619-1_7

[58] N. Chrisochoides, E. Houstis, and J. Rice, "Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers," *Journal of Parallel and Distributed Computing*, vol. 21, no. 1, pp. 75–95, Apr. 1994, DOI: 10.1006/jpdc.1994.1043

[59] N. Chrisochoides and D. Nave, "Parallel Delaunay mesh generation kernel," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 161–176, Sep. 2003, DOI: 10.1002/nme.765

[60] N. Chrisochoides and F. Sukup, "Task Parallel Implementation of the Bowyer-Watson Algorithm," in *Proceedings of Fifth International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, 1996, pp. 773–782.

[61] N. P. Chrisochoides, "Telescopic Approach for Extreme-Scale Parallel Mesh Generation for CFD Applications," in *46th AIAA Fluid Dynamics Conference.* American Institute of Aeronautics and Astronautics, 2016, DOI: 10.2514/6.2016-3181

[62] N. P. Chrisochoides, A. Chernikov, T. Kennedy, C. Tsolakis, and K. M. Garner, "Parallel Data Refinement Layer of a Telescopic Approach for Extreme-scale Parallel Mesh Generation for CFD Applications," in *2018 Aviation Technology, Integration, and Operations Conference*. American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-2887

[63] L. Cirrottola and A. Froehly, "Parallel Unstructured Mesh Adaptation Based on Iterative Remeshing and Repartitioning," *14th WCCM-ECCOMAS Congress 2020*, Mar. 2021, DOI: 10.23967/wccm-eccomas.2020.270

[64] L. Cirrottola and A. Froehly, "Parallel unstructured mesh adaptation using iterative remeshing and repartitioning," INRIA Bordeaux, équipe CARDAMOM, Tech. Rep. RR-9307, Nov. 2019. ttps://hal.inria.fr/hal-02386837

[65] F. Commandeur, J. Velut, and O. Acosta, "A VTK Algorithm for the Computation of the Hausdorff Distance," *The VTK Journal*, p. 839, Sep. 2011. http://hdl.handle.net/10380/3322 (Accessed 2021-06-29).

[66] M. E. Conway, "A multiprocessor system design," in *Proceedings of the November 12-14, 1963, fall joint computer conference*, ser. AFIPS '63 (Fall). New York, NY, USA: Association for Computing Machinery, Nov. 1963, pp. 139–146. ISBN 978-1-4503-7883-3 DOI: 10.1145/1463822.1463838

[67] T. Coupez, H. Digonnet, and R. Ducloux, "Parallel meshing and remeshing," *Applied Mathematical Modelling*, vol. 25, no. 2, pp. 153–175, Dec. 2000, DOI: 10.1016/S0307-904X(00)00045-7

[68] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th National Conference*, ser. ACM '69. New York, NY, USA: ACM, 1969, pp. 157–172, DOI: 10.1145/800195.805928

[69] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, Jan. 1998, conference Name: IEEE Computational Science and Engineering. DOI: 10.1109/99.660313

[70] C. Dapogny, C. Dobrzynski, and P. Frey, “Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems,” *Journal of Computational Physics*, vol. 262, pp. 358–378, 2014, DOI: 10.1016/j.jcp.2014.01.005

[71] F. Dassi, S. Perotto, H. Si, and T. Streckenbach, “A priori anisotropic mesh adaptation driven by a higher dimensional embedding,” *Computer-Aided Design*, vol. 85, pp. 111–122, 2016, DOI: 10.1016/j.cad.2016.07.012

[72] H. L. de Cougny and M. S. Shephard, “Parallel refinement and coarsening of tetrahedral meshes,” *International Journal for Numerical Methods in Engineering*, vol. 46, no. 7, pp. 1101–1125, Nov. 1999, DOI: 10.1002/(SICI)1097-0207(19991110)46:7¡1101::AID-NME741¿3.0.CO;2-E

[73] H. L. de Cougny and M. S. Shephard, “Parallel volume meshing using face removals and hierarchical repartitioning,” *Computer Methods in Applied Mechanics and Engineering*, vol. 174, no. 3, pp. 275–298, May 1999, DOI: 10.1016/S0045-7825(98)00300-4

[74] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, “Parallel hypergraph partitioning for scientific computing,” in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, ser. IPDPS’06. IEEE Computer Society, 2006, pp. 124–124. ISBN 978-1-4244-0054-6 Event-place: Rhodes Island, Greece. DOI: 10.1109/IPDPS.2006.1639359

[75] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, “New challenges in dynamic load balancing,” *Applied Numerical Mathematics*, vol. 52, no. 2–3, pp. 133–152, 2005, DOI: 10.1016/j.apnum.2004.08.028

[76] H. Digonnet, T. Coupez, P. Laure, and L. Silva, “Massively parallel anisotropic mesh adaptation,” *The International Journal of High Performance Computing Applications*, vol. 33, no. 1, pp. 3–24, Jan. 2019, publisher: SAGE Publications Ltd STM. DOI: 10.1177/1094342017693906

[77] E. W. Dijkstra, “On the role of scientific thought,” in *Selected writings on computing: a personal perspective.* Berlin, Heidelberg: Springer-Verlag, 1982, pp. 60–66. ISBN 978-0-387-90652-2

[78] E. W. Dijkstra, "Shmuel Safra's Termination Detection Algorithm," in *On a Method of Multiprogramming*, ser. Monographs in Computer Science, W. H. J. Feijen and A. J. M. van Gasteren, Eds. New York, NY: Springer, 1999, pp. 313–332. ISBN 978-1-4757-3126-2 DOI: 10.1007/978-1-4757-3126-2_29

[79] C. Dobrzynski and P. Frey, "Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations," in *Proceedings of the 17th International Meshing Roundtable*, R. V. Garimella, Ed. Springer Berlin Heidelberg, 2009, pp. 177–194. ISBN 978-3-540-87920-6 978-3-540-87921-3

[80] J. Dompierre, Y. Mokwinski, M.-G. Vallet, and F. Guibault, "On ellipse intersection and union with application to anisotropic mesh adaptation," *Engineering with Computers*, vol. 33, no. 4, pp. 745–766, Oct. 2017, DOI: 10.1007/s00366-017-0533-y

[81] J. J. Dongarra, "Performance of various computers using standard linear equations software," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 1, p. 17, Mar. 1990, DOI: 10.1145/379126.379129

[82] F. Drakopoulos, "Finite Element Modeling Driven by Health Care and Aerospace Applications," Ph.D. dissertation, Computer Science, Old Dominion University, Norfolk,Virginia, Jul. 2017, ISBN: 9780355362169. DOI: 10.25777/p9kt-9c56

[83] F. Drakopoulos and N. P. Chrisochoides, "Accurate and fast deformable medical image registration for brain tumor resection using image-guided neurosurgery," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, vol. 4, no. 2, pp. 112–126, Mar. 2016, DOI: 10.1080/21681163.2015.1067869

[84] F. Drakopoulos, C. Tsolakis, A. Angelopoulos, Y. Liu, C. Yao, K. R. Kavazidi, N. Foroglou, A. Fedorov, S. Frisken, R. Kikinis, A. Golby, and N. Chrisochoides, "Adaptive Physics-Based Non-Rigid Registration for Immersive Image-Guided Neuronavigation Systems," *Frontiers in Digital Health*, vol. 2, 2021, DOI: 10.3389/fdgth.2020.613608

[85] F. Drakopoulos, C. Tsolakis, and N. P. Chrisochoides, "Fine-Grained Speculative Topological Transformation Scheme for Local Reconnection Methods," *AIAA Journal*, vol. 57, no. 9, pp. 4007–4018, Jul. 2019, DOI: 10.2514/1.J057657

[86] A. Fedorov and N. Chrisochoides, "Tetrahedral Mesh Generation for Non-rigid Registration of Brain MRI: Analysis of the Requirements and Evaluation of Solutions,"

in *Proceedings of the 17th International Meshing Roundtable*, R. V. Garimella, Ed. Springer, 2008, pp. 55–72. ISBN 978-3-540-87921-3 DOI: 10.1007/978-3-540-87921-3_4

[87] D. Feng, A. N. Chernikov, and N. P. Chrisochoides, “Two-level locality-aware parallel delaunay image-to-mesh conversion,” *Parallel Computing*, vol. 59, pp. 60–70, 2016, DOI: 10.1016/j.parco.2016.01.007

[88] D. Feng, A. N. Chernikov, and N. P. Chrisochoides, “A hybrid parallel Delaunay image-to-mesh conversion algorithm scalable on distributed-memory clusters,” *Computer-Aided Design*, vol. 103, pp. 34–46, Oct. 2018, DOI: 10.1016/j.cad.2017.11.006

[89] D. Feng, C. Tsolakis, A. N. Chernikov, and N. P. Chrisochoides, “Scalable 3D Hybrid Parallel Delaunay Image-to-mesh Conversion Algorithm for Distributed Shared Memory Architectures,” *Comput. Aided Des.*, vol. 85, no. C, pp. 10–19, Apr. 2017, DOI: 10.1016/j.cad.2016.07.010

[90] K. J. Fidkowski and D. L. Darmofal, “Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics,” *AIAA Journal*, vol. 49, no. 4, pp. 673–694, 2011, DOI: 10.2514/1.J050073

[91] P. J. Fleming and J. J. Wallace, “How not to lie with statistics: the correct way to summarize benchmark results,” *Communications of the ACM*, vol. 29, no. 3, pp. 218–221, Mar. 1986, DOI: 10.1145/5666.5673

[92] P. Foteinos, “Real-Time High-Quality Image to Mesh Conversion for Finite Element Simulations,” Ph.D. dissertation, The College of William and Mary, United States, Virginia, 2013. http://search.proquest.com/docview/1532219419

[93] P. Foteinos and N. Chrisochoides, “Dynamic Parallel 3D Delaunay Triangulation,” in *Proceedings of the 20th International Meshing Roundtable*, W. R. Quadros, Ed. Springer Berlin Heidelberg, 2011, pp. 3–20. ISBN 978-3-642-24734-7 DOI: 10.1007/978-3-642-24734-7_1

[94] P. Foteinos and N. Chrisochoides, “4D space–time Delaunay meshing for medical images,” *Engineering with Computers*, vol. 31, no. 3, pp. 499–511, Oct. 2014, DOI: 10.1007/s00366-014-0380-z

[95] P. A. Foteinos and N. P. Chrisochoides, “High quality real-time Image-to-Mesh conversion for finite element simulations,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2123–2140, Feb. 2014, DOI: 10.1016/j.jpdc.2013.11.002

[96] L. Freitag, M. Jones, and P. Plassmann, “A Parallel Algorithm for Mesh Smoothing,” *SIAM Journal on Scientific Computing*, vol. 20, no. 6, pp. 2023–2040, Jan. 1999, DOI: 10.1137/S1064827597323208

[97] L. A. Freitag and C. Ollivier-Gooch, “Tetrahedral mesh improvement using swapping and smoothing,” *International Journal for Numerical Methods in Engineering*, vol. 40, no. 21, pp. 3979–4002, Nov. 1997, DOI: 10.1002/1097-0207

[98] X.-M. Fu, Y. Liu, J. Snyder, and B. Guo, “Anisotropic simplicial meshing using local convex functions,” *ACM Transactions on Graphics*, vol. 33, no. 6, pp. 182:1–182:11, Nov. 2014, DOI: 10.1145/2661229.2661235

[99] F. J. Furrer, *Future-Proof Software-Systems: A Sustainable Evolution Strategy.* Springer Vieweg, 2019. ISBN 978-3-658-19937-1 DOI: 10.1007/978-3-658-19938-8

[100] M. C. Galbraith, P. C. Caplan, H. A. Carson, M. A. Park, A. Balan, W. K. Anderson, T. Michal, J. A. Krakos, D. S. Kamenetskiy, A. Loseille, F. Alauzet, L. Frazza, and N. Barral, “Verification of Unstructured Grid Adaptation Components,” *AIAA Journal*, vol. 58, no. 9, pp. 3947–3962, 2020, DOI: 10.2514/1.J058783

[101] J. Galtier and P. L. George, “Prepartitioning as a way to mesh subdomains in parallel,” in *Proceedings of the 5th International Meshing Roundtable*, Pittsburgh, Pennsylvania, Oct. 1996, pp. 107–122.

[102] M. Gammon, “A review of common geometry issues affecting mesh generation,” in *2018 AIAA Aerospace Sciences Meeting.* American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-1402

[103] M. R. Gammon, J. H. Bucklow, R. Fairey, and S. Seebooa, “Flexible 3D medial partitioning for CFD and FEA meshing,” in *AIAA Scitech 2020 Forum.* American Institute of Aeronautics and Astronautics, 2020, DOI: 10.2514/6.2020-0901

[104] R. R. Garlapati, G. R. Joldes, A. Wittek, J. Lam, N. Weisenfeld, A. Hans, S. K. Warfield, R. Kikinis, and K. Miller, "Objective Evaluation of Accuracy of Intra-Operative Neuroimage Registration," *Computational Biomechanics for Medicine*, pp. 87–99, 2013, DOI: 10.1007/978-1-4614-6351-1_9

[105] K. Garner, "Parallelization of the Advancing Front Local Reconnection Mesh Generation Software Using a Pseudo-Constrained Parallel Data Refinement Method," Master's thesis, Computer Science, Old Dominion University, 2020, DOI: 10.25777/appr-3169

[106] K. M. Garner, P. Thomadakis, T. Kennedy, C. Tsolakis, and N. N. Chrisochoides, "On the End-User Productivity of a Pseudo-Constrained Parallel Data Refinement Method for the Advancing Front Local Reconnection Mesh Generation Software," in *AIAA Aviation 2019 Forum.* American Institute of Aeronautics and Astronautics, Jun. 2019, DOI: 10.2514/6.2019-2844

[107] G. Gavalian and N. Chrisochoides, "Next-Generation Imaging Filters and Mesh-Based Data Representation for Phase-Space Calculations in Nuclear Femtography (CNF19-04)," SURA Headquarters Washington DC, Aug. 2019. https://indico.jlab.org/event/335/contributions/5274/attachments/4377/5329/NuclerFemtography-Aug-2019.pdf (Accessed 2021-07-01).

[108] P. L. George, F. Hecht, and E. Saltel, "Automatic mesh generator with specified boundary," *Computer Methods in Applied Mechanics and Engineering*, vol. 92, no. 3, pp. 269–288, Nov. 1991, DOI: 10.1016/0045-7825(91)90017-Z

[109] P. L. George, F. Hecht, and M. G. Vallet, "Creation of internal points in Voronoi's type method. Control adaptation," *Advances in Engineering Software and Workstations*, vol. 13, no. 5, pp. 303–312, 1991, DOI: 10.1016/0961-3552(91)90034-2

[110] P. L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille, D. Marcum, and L. Maréchal, "Mesh Generation and Mesh Adaptivity: Theory and Techniques," in *Encyclopedia of Computational Mechanics Second Edition.* Wiley, 2017, pp. 1–51. ISBN 978-1-119-17681-7 DOI: 10.1002/9781119176817.ecm2012

[111] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009, DOI: 10.1002/nme.2579

[112] G. J. Gorman, J. Southern, P. E. Farrell, M. D. Piggott, G. Rokos, and P. H. J. Kelly, "Hybrid OpenMP/MPI Anisotropic Mesh Smoothing," *Procedia Computer Science*, vol. 9, pp. 1513–1522, Jan. 2012, DOI: 10.1016/j.procs.2012.04.166

[113] G. J. Gorman, "PRAgMaTIc GitHub website," https://meshadaptation.github.io, 2021, (Accessed 2021-05-21).

[114] G. J. Gorman, G. Rokos, J. Southern, and P. H. J. Kelly, "Thread-Parallel Anisotropic Mesh Adaptation," in *New Challenges in Grid Generation and Adaptivity for Scientific Computing*, ser. SEMA SIMAI Springer Series. Springer, Cham, 2015, pp. 113–137. ISBN 978-3-319-06053-8 DOI: 10.1007/978-3-319-06053-8_6

[115] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to parallel computing*, 2nd ed. Harlow, England ; New York: Addison-Wesley, 2003. ISBN 0-201-64865-2

[116] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[117] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, May 1988, DOI: 10.1145/42411.42415

[118] W. G. Habashi, J. Dompierre, Y. Bourgault, D. Ait-Ali-Yahia, M. Fortin, and M.-G. Vallet, "Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part I: general principles," *International Journal for Numerical Methods in Fluids*, vol. 32, no. 6, pp. 725–744, 2000, DOI: 10.1002/(SICI)1097-0363(20000330)32:6¡725::AID-FLD935¿3.0.CO;2-4

[119] R. Haimes and J. Dannenhoffer, "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry," in *21st AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2013, DOI: 10.2514/6.2013-3073

[120] R. Haimes and J. F. Dannenhoffer, III, "EGADSlite: A Lightweight Geometry Kernel for HPC," in *2018 AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-1401

[121] R. Haimes and M. Drela, "On the construction of aircraft conceptual geometry for high-fidelity analysis and design," in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2012, DOI: 10.2514/6.2012-683

[122] R. Hartmann, J. Held, T. Leicht, and F. Prill, "Error Estimation and Adaptive Mesh Refinement for Aerodynamic Flows," in *ADIGMA - A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, ser. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, N. Kroll, H. Bieler, H. Deconinck, V. Couaillier, H. van der Ven, and K. Sørensen, Eds. Berlin, Heidelberg: Springer, 2010, pp. 339–353. ISBN 978-3-642-03707-8 DOI: 10.1007/978-3-642-03707-8_24

[123] B. Hendrickson and R. Leland, "The chaco users guide. version 1.0," Sandia National Labs., Albuquerque, NM (United States), Tech. Rep., 1993.

[124] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, "Online Learning: A Comprehensive Survey," *arXiv:1802.02871 [cs]*, Oct. 2018, arXiv: 1802.02871.

[125] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. USA: Cambridge University Press, 2012. ISBN 0521548233

[126] G. Houzeaux, R. de la Cruz, H. Owen, and M. Vázquez, "Parallel uniform mesh multiplication applied to a Navier–Stokes solver," *Computers & Fluids*, vol. 80, pp. 142–151, Jul. 2013, DOI: 10.1016/j.compfluid.2012.04.017

[127] T. Hsieh, "An investigation of separated flow about a hemisphere-cylinder at incidence in the Mach number range from 0.6 to 1.5," in *15th Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 1977, DOI: 10.2514/6.1977-179

[128] D. Ibanez and M. Shephard, "Mesh adaptation for moving objects on shared memory hardware," in *25th International Meshing Roundtable*, 2016, Research Note.

[129] D. Ibanez, N. Barral, J. Krakos, A. Loseille, T. Michal, and M. Park, "First benchmark of the Unstructured Grid Adaptation Working Group," *Procedia Engineering*, vol. 203, pp. 154–166, Jan. 2017, DOI: 10.1016/j.proeng.2017.09.800

[130] D. A. Ibanez, "Conformal mesh adaptation on heterogeneous supercomputers," Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, New York, 2016.

[131] T. Ito, Y. Yokokawa, H. Ura, H. Kato, K. Mitsuo, and K. Yamamoto, "High-Lift Device Testing in JAXA 6.5M X 5.5M Low-Speed Wind Tunnel," in *25th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*, ser. Fluid Dynamics

and Co-located Conferences. American Institute of Aeronautics and Astronautics, Jun. 2006, DOI: 10.2514/6.2006-3643

[132] Y. Ito, A. M. Shih, A. K. Erukala, B. K. Soni, A. Chernikov, N. P. Chrisochoides, and K. Nakahashi, "Parallel unstructured mesh generation by an advancing front method," *Mathematics and Computers in Simulation*, vol. 75, no. 5-6, pp. 200–209, Sep. 2007, DOI: 10.1016/j.matcom.2006.12.008

[133] E. G. Ivanov, H. Andrä, and A. N. Kudryavtsev, "Domain Decomposition Approach for Automatic Parallel Generation of 3D Unstructured Grids," in *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics*, 2006. http://resolver.tudelft.nl/uuid:076b639d-2296-4d4e-91e7-ac7069951eed (Accessed 2021-05-21).

[134] F. J. Bossen and P. Heckbert, "A Pliant Method for Anisotropic Mesh Generation," in *Proceedings of the 5th International Meshing Roundtable*, Oct. 1998, pp. 63–74.

[135] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, Jul. 1985, DOI: 10.1145/3916.3988

[136] A. A. Johnson and S. K. Aliabadi, "Application of automatic mesh generation and mesh multiplication techniques to very large scale free-surface flow simulations," in *Proceedings of the 7th international conference on numerical grid generation in computational field simulations*, Whistler (Canada), Sep. 2000.

[137] M. T. Jones and P. E. Plassmann, "Computational results for parallel unstructured mesh computations," *Computing Systems in Engineering*, vol. 5, no. 4, pp. 297–309, Aug. 1994, DOI: 10.1016/0956-0521(94)90013-2

[138] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998, DOI: 10.1137/S1064827595287997

[139] L. G. Khachiyan, "Rounding of Polytopes in the Real Number Model of Computation," *Mathematics of Operations Research*, vol. 21, no. 2, pp. 307–320, 1996. https://www.jstor.org/stable/3690235 (Accessed 2020-01-31).

[140] B. M. Klingner, "Improving tetrahedral meshes," Ph.D. dissertation, University of California, Berkeley, California, United States, 2008.

[141] J. Kohout, I. Kolingerová, and J. Žára, "Parallel Delaunay triangulation in E2 and E3 for computers with shared memory," *Parallel Computing*, vol. 31, no. 5, pp. 491–522, May 2005, DOI: 10.1016/j.parco.2005.02.010

[142] M. Kulkarni, K. Pingali, B. Walter, G. Ramanarayanan, K. Bala, and L. P. Chew, "Optimistic parallelism requires abstractions," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY, USA: Association for Computing Machinery, Jun. 2007, pp. 211–222. ISBN 978-1-59593-633-2 DOI: 10.1145/1250734.1250759

[143] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems*, vol. 6, no. 2, pp. 213–226, Jun. 1981, DOI: 10.1145/319566.319567

[144] F. Labelle and J. R. Shewchuk, "Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation," in *in SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry.* ACM Press, 2003, pp. 191–200.

[145] B. G. Larwood, N. P. Weatherill, O. Hassan, and K. Morgan, "Domain decomposition approach for parallel unstructured mesh generation," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 177–188, Sep. 2003, DOI: 10.1002/nme.769

[146] C. L. Lawson, "Software for C1 Surface Interpolation," in *Mathematical Software*, J. R. Rice, Ed. Academic Press, Jan. 1977, pp. 161–194. ISBN 978-0-12-587260-7 DOI: 10.1016/B978-0-12-587260-7.50011-X

[147] G. Leibon and D. Letscher, "Delaunay Triangulations and Voronoi Diagrams for Riemannian Manifolds," in *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, ser. SCG '00. New York, NY, USA: ACM, 2000, pp. 341–349. ISBN 1-58113-224-7 DOI: 10.1145/336154.336221

[148] T. Leicht and R. Hartmann, "Error estimation and anisotropic mesh refinement for 3D laminar aerodynamic flow simulations," *Journal of Computational Physics*, vol. 229, no. 19, pp. 7344–7360, Sep. 2010, DOI: 10.1016/j.jcp.2010.06.019

[149] B. Lévy and N. Bonneel, "Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration," in *Proceedings of the 21st International Meshing Roundtable*,

X. Jiao and J.-C. Weill, Eds. Springer Berlin Heidelberg, 2013, pp. 349–366. ISBN 978-3-642-33573-0 DOI: 10.1007/978-3-642-33573-0_21

[150] K. Li, "Performance Analysis of Power-Aware Task Scheduling Algorithms on Multiprocessor Computers with Dynamic Voltage and Speed," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1484–1497, Nov. 2008, DOI: 10.1109/TPDS.2008.122

[151] L. Linardakis and N. Chrisochoides, "Delaunay Decoupling Method for Parallel Guaranteed Quality Planar Mesh Refinement," *SIAM J. Sci. Comput.*, vol. 27, no. 4, pp. 1394–1423, Jan. 2006, DOI: 10.1137/030602812

[152] L. Linardakis and N. Chrisochoides, "Algorithm 870: A Static Geometric Medial Axis Domain Decomposition in 2d Euclidean Space," *ACM Transactions on Mathematical Software*, vol. 34, no. 1, pp. 4:1–4:28, Jan. 2008, DOI: 10.1145/1322436.1322440

[153] K. Lipnikov and Y. Vassilevski, "An adaptive algorithm for quasioptimal mesh generation," *Computational Mathematics and Mathematical Physics*, vol. 39, no. 9, pp. 1468–1486, 1999.

[154] A. Liu and B. Joe, "Relationship between tetrahedron shape measures," *BIT Numerical Mathematics*, vol. 34, no. 2, pp. 268–287, Jun. 1994, DOI: 10.1007/BF01955874

[155] S. H. Lo, "3D Delaunay triangulation of 1 billion points on a PC," *Finite Elements in Analysis and Design*, vol. 102–103, pp. 65–73, Oct. 2015, DOI: 10.1016/j.finel.2015.05.003

[156] R. Löhner, "A parallel advancing front grid generation scheme," *International Journal for Numerical Methods in Engineering*, vol. 51, no. 6, pp. 663–678, Jun. 2001, DOI: 10.1002/nme.175

[157] R. Löhner, "Recent Advances in Parallel Advancing Front Grid Generation," *Archives of Computational Methods in Engineering*, vol. 21, no. 2, pp. 127–140, Jun. 2014, DOI: 10.1007/s11831-014-9098-8

[158] A. Loseille, "Unstructured mesh generation and adaptation," in *Handbook of Numerical Methods for Hyperbolic Problems: Applied and Modern Issues*, ser. Handbook of Numerical Analysis, R. Abgrall and C.-W. Shu, Eds. Elsevier, 2017, vol. 18, pp. 263–302, DOI: 10.1016/bs.hna.2016.10.004

[159] A. Loseille, F. Alauzet, and V. Menier, “Unique cavity-based operator and hierarchical domain partitioning for fast parallel generation of anisotropic meshes,” *Computer-Aided Design*, vol. 85, pp. 53–67, Apr. 2017, DOI: 10.1016/j.cad.2016.09.008

[160] A. Loseille, “Metric-orthogonal Anisotropic Mesh Generation,” *Procedia Engineering*, vol. 82, no. Supplement C, pp. 403–415, Jan. 2014, DOI: 10.1016/j.proeng.2014.10.400

[161] A. Loseille, A. Dervieux, P. Frey, and F. Alauzet, “Achievement of Global Second Order Mesh Convergence for Discontinuous Flows with Adapted Unstructured Meshes,” in *18th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, Jun. 2007, DOI: 10.2514/6.2007-4186

[162] A. Loseille and R. Lohner, “Anisotropic Adaptive Simulations in Aerodynamics,” in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, 2010, DOI: 10.2514/6.2010-169

[163] A. Loseille and R. Löhner, “Robust Boundary Layer Mesh Generation,” in *Proceedings of the 21st International Meshing Roundtable*, X. Jiao and J.-C. Weill, Eds. Springer Berlin Heidelberg, 2013, pp. 493–511. ISBN 978-3-642-33573-0 DOI: 10.1007/978-3-642-33573-0_29

[164] A. Loseille and V. Menier, “Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive,” in *Proceedings of the 22nd International Meshing Roundtable*, J. Sarrate and M. Staten, Eds. Springer International Publishing, 2014, pp. 541–558. ISBN 978-3-319-02335-9 DOI: 10.1007/978-3-319-02335-9_30

[165] A. Loseille, V. Menier, and F. Alauzet, “Parallel generation of large-size adapted meshes,” in *Procedia Engineering*, ser. 24th International Meshing Roundtable, vol. 124. Sandia National Laboratories, 2015, pp. 57–69. ISSN 1877-7058 DOI: 10.1016/j.proeng.2015.10.122

[166] A. Loseille and F. Alauzet, “Continuous Mesh Framework Part I: Well-Posed Continuous Interpolation Error,” *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, pp. 38–60, 2011, DOI: 10.1137/090754078

[167] A. Loseille and F. Alauzet, “Continuous Mesh Framework Part II: Validations and Applications,” *SIAM Journal on Numerical Analysis*, vol. 49, no. 1, pp. 61–86, Jan. 2011, DOI: 10.1137/10078654X

[168] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986, DOI: 10.1137/0215074

[169] B. Lévy, "Robustness and efficiency of geometric programs: The Predicate Construction Kit (PCK)," *Computer-Aided Design*, vol. 72, pp. 3–12, 2016, DOI: 10.1016/j.cad.2015.10.004

[170] D. Marcum and F. Alauzet, "Aligned Metric-based Anisotropic Solution Adaptive Mesh Generation," *Procedia Engineering*, vol. 82, pp. 428–444, 2014, DOI: 10.1016/j.proeng.2014.10.402

[171] D. L. Marcum and F. Alauzet, "Unstructured Mesh Generation Using Advancing Layers and Metric-Based Transition for Viscous Flowfields," in *21st AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2013, DOI: 10.2514/6.2013-2710

[172] D. L. Marcum and F. Alauzet, "Unstructured Mesh Generation Using Advancing Layers and Metric-Based Transition for Viscous Flowfields," in *21st AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2013, DOI: 10.2514/6.2013-2710

[173] D. L. Marcum and N. P. Weatherill, "Unstructured grid generation using iterative point insertion and local reconnection," *AIAA Journal*, vol. 33, no. 9, pp. 1619–1625, 1995, DOI: 10.2514/3.12701

[174] L. Maréchal, "An easy way to access files in Gamma Mesh Format, the libMeshb library," https://github.com/LoicMarechal/libMeshb, 2021, (Accessed 2021-05-21).

[175] C. Marot, "Parallel tetrahedral mesh generation," Ph.D. dissertation, UCL - Université Catholique de Louvain, Belgium, 2020. http://hdl.handle.net/2078.1/240626

[176] C. Marot, J. Pellerin, and J.-F. Remacle, "One machine, one minute, three billion tetrahedra," *International Journal for Numerical Methods in Engineering*, vol. 117, no. 9, pp. 967–990, Mar. 2019, DOI: 10.1002/nme.5987

[177] D. J. Mavriplis, “Adaptive mesh generation for viscous flows using triangulation,” *Journal of Computational Physics*, vol. 90, no. 2, pp. 271–291, Oct. 1990, DOI: 10.1016/0021-9991(90)90167-Y

[178] H. W. Meuer, “The TOP500 Project: Looking Back Over 15 Years of Supercomputing Experience,” *Informatik-Spektrum*, vol. 31, no. 3, pp. 203–222, Jun. 2008, DOI: 10.1007/s00287-008-0240-6

[179] J. C. Meyer, “Implementation of an Energy-Aware OmpSs Task Scheduling Policy,” Zenodo, Tech. Rep., Jul. 2013, DOI: 10.5281/zenodo.832011

[180] T. Michal, D. Babcock, D. Kamenetskiy, J. Krakos, M. Mani, R. Glasby, T. Erwin, and D. L. Stefanski, “Comparison of Fixed and Adaptive Unstructured Grid Results for Drag Prediction Workshop 6,” *Journal of Aircraft*, pp. 1–13, Dec. 2017, DOI: 10.2514/1.C034491

[181] T. Michal and J. Krakos, “Anisotropic Mesh Adaptation Through Edge Primitive Operations,” in *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, 2012, DOI: 10.2514/6.2012-159

[182] T. R. Michal, D. S. Kamenetskiy, and J. Krakos, “Anisotropic Adaptive Mesh Results for the Third High Lift Prediction Workshop (HiLiftPW-3),” in *2018 AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-1257

[183] Mmg Developers, “Mmg Github,” https://github.com/MmgTools/Mmg, Mar. 2021, (Accessed 2021-05-19).

[184] Mmg Developers, “ParMmg Github,” https://github.com/MmgTools/ParMmg, Mar. 2021, (Accessed 2021-05-19).

[185] L. Mukhanov, D. S. Nikolopoulos, and B. R. De Supinski, “ALEA: Fine-Grain Energy Profiling with Basic Block Sampling,” in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, Oct. 2015, pp. 87–98, iSSN: 1089-795X. DOI: 10.1109/PACT.2015.16

[186] J. Nash, “C1 Isometric Imbeddings,” *Annals of Mathematics*, vol. 60, no. 3, pp. 383–396, 1954, DOI: 10.2307/1969840

[187] D. Nave, N. Chrisochoides, and L. P. Chew, "Guaranteed: Quality Parallel Delaunay Refinement for Restricted Polyhedral Domains," in *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, ser. SCG '02. New York, NY, USA: ACM, 2002, pp. 135–144. ISBN 1-58113-504-1 DOI: 10.1145/513400.513418

[188] D. Nave, N. Chrisochoides, and L. P. Chew, "Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains," *Computational Geometry*, vol. 28, no. 2–3, pp. 191–215, Jun. 2004, DOI: 10.1016/j.comgeo.2004.03.009

[189] Y.-W. Ning, P. Suprobo, G. D. Jeong, and E. C. Ting, "A regional mixed refinement procedure for finite element mesh design," *Finite Elements in Analysis and Design*, vol. 13, no. 4, pp. 299–318, 1993, DOI: 10.1016/0168-874X(93)90046-S

[190] H. Nishikawa and B. Diskin, "Customized Grid Generation Codes for Benchmark Three-Dimensional Flows," in *2018 AIAA Aerospace Sciences Meeting*, ser. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2018, DOI: 10.2514/6.2018-1101

[191] M. O'Connell, C. Druyor, K. B. Thompson, K. Jacobson, W. K. Anderson, E. J. Nielsen, J.-R. Carlson, M. A. Park, W. T. Jones, R. Biedron, E. M. Lee-Rausch, and B. Kleb, "Application of the Dependency Inversion Principle to Multidisciplinary Software Development," in *2018 Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, Jun. 2018, DOI: 10.2514/6.2018-3856

[192] T. Okusanya and J. Peraire, "3D Parallel Unstructured Mesh Generation," in *Trends in Unstructured Mesh Generation*, ser. AMD (Series). Evaston, Illinois: American Society of Mechanical Engineers, 1997, vol. 220, pp. 109–115.

[193] L. Oliker, R. Biswas, and H. N. Gabow, "Parallel tetrahedral mesh adaptation with dynamic load balancing," *Parallel Computing*, vol. 26, no. 12, pp. 1583–1608, Nov. 2000, DOI: 10.1016/S0167-8191(00)00047-8

[194] C. F. Ollivier Gooch, "Generation of Exascale Meshes by Subdivision of Coarse Meshes," in *AIAA Scitech 2020 Forum*, ser. AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan. 2020, DOI: 10.2514/6.2020-1404

[195] S. J. Owen, M. L. Staten, and M. C. Sorensen, "Parallel Hex Meshing from Volume Fractions," in *Proceedings of the 20th International Meshing Roundtable*, W. R.

Quadros, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 161–178. ISBN 978-3-642-24734-7 DOI: 10.1007/978-3-642-24734-7_9

[196] D. Panozzo, E. Puppo, M. Tarini, and O. Sorkine-Hornung, "Frame fields: anisotropic and non-orthogonal cross fields," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 134:1–134:11, Jul. 2014, DOI: 10.1145/2601097.2601179

[197] M. Park and D. Darmofal, "Parallel Anisotropic Tetrahedral Adaptation," in *46th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, 2008, DOI: 10.2514/6.2008-917

[198] M. A. Park, "refine : grid adaptation framework," https://github.com/nasa/refine, (Accessed 2021-01-07).

[199] M. A. Park, A. Balan, W. K. Anderson, M. C. Galbraith, P. C. Caplan, H. A. Carson, T. Michal, J. A. Krakos, D. S. Kamenetskiy, A. Loseille, F. Alauzet, L. Frazza, and N. Barral, "Verification of unstructured grid adaptation components," in *AIAA Scitech 2019 Forum*, 2019, DOI: 10.2514/6.2019-1723

[200] M. A. Park, A. Balan, F. Clerici, F. Alauzet, A. Loseille, D. S. Kamenetskiy, J. A. Krakos, T. R. Michal, and M. C. Galbraith, "Verification of Viscous Goal-Based Anisotropic Mesh Adaptation," in *AIAA Scitech 2021 Forum*. American Institute of Aeronautics and Astronautics, 2021, DOI: 10.2514/6.2021-1362

[201] M. A. Park, N. Barral, D. Ibanez, D. S. Kamenetskiy, J. A. Krakos, T. R. Michal, and A. Loseille, "Unstructured Grid Adaptation and Solver Technology for Turbulent Flows," in *2018 AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-1103

[202] M. A. Park, W. L. Kleb, W. T. Jones, J. A. Krakos, T. R. Michal, A. Loseille, R. Haimes, and J. Dannenhoffer, "Geometry Modeling for Unstructured Mesh Adaptation," in *AIAA Aviation 2019 Forum*, ser. AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, Jun. 2019, DOI: 10.2514/6.2019-2946

[203] M. A. Park, J. A. Krakos, T. Michal, A. Loseille, and J. J. Alonso, "Unstructured grid adaptation: Status, potential impacts, and recommended investments toward CFD vision 2030," in *46th AIAA Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2016, DOI: 10.2514/6.2016-3323

[204] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz, "Adaptive remeshing for compressible flow computations," *Journal of Computational Physics*, vol. 72, no. 2, pp. 449–466, Oct. 1987, DOI: 10.1016/0021-9991(87)90093-3

[205] M. Perdigão do Carmo, *Differential Geometry of Curves and Surfaces.* Prentice-Hall, Inc., 1976. ISBN 0132125897

[206] J. R. Pilkington and S. B. Baden, "Dynamic Partitioning of Non-Uniform Structured Workloads with Spacefilling Curves," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 3, pp. 288–300, Mar. 1996, DOI: 10.1109/71.491582

[207] S. Pirzadeh, "Recent progress in unstructured grid generation," in *30th Aerospace Sciences Meeting and Exhibit.* Reno, NV, U.S.A.: American Institute of Aeronautics and Astronautics, Jan. 1992, DOI: 10.2514/6.1992-445

[208] S. Pirzadeh, "Three-dimensional unstructured viscous grids by the advancing-layers method," *AIAA Journal*, vol. 34, no. 1, pp. 43–49, 1996, DOI: 10.2514/3.13019

[209] A. Raman, H. Kim, T. R. Mason, T. B. Jablin, and D. I. August, "Speculative parallelization using software multi-threaded transactions," in *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*, ser. ASPLOS XV. New York, NY, USA: Association for Computing Machinery, Mar. 2010, pp. 65–76. ISBN 978-1-60558-839-1 DOI: 10.1145/1736020.1736030

[210] L. Rauchwerger and D. Padua, "The LRPD test: speculative run-time parallelization of loops with privatization and reduction parallelization," *ACM SIGPLAN Notices*, vol. 30, no. 6, pp. 218–232, Jun. 1995, DOI: 10.1145/223428.207148

[211] J.-F. c. Remacle, "A two-level multithreaded Delaunay kernel," *Computer-Aided Design*, vol. 85, pp. 2–9, Apr. 2017, DOI: 10.1016/j.cad.2016.07.018

[212] J.-F. c. Remacle, V. Bertrand, and C. Geuzaine, "A Two-Level Multithreaded Delaunay Kernel," *Procedia Engineering*, vol. 124, pp. 6–17, Jan. 2015, DOI: 10.1016/j.proeng.2015.10.118

[213] D. Q. Ren, E. Bracken, S. Polstyanko, N. Lambert, R. Suda, and D. D. Giannacopulos, "Power Aware Parallel 3-D Finite Element Mesh Refinement Performance Modeling and Analysis With CUDA/MPI on GPU and Multi-Core Architecture," *IEEE*

Transactions on Magnetics, vol. 48, no. 2, pp. 335–338, Feb. 2012, DOI: 10.1109/TMAG.2011.2177814

[214] M.-C. Rivara, "Lepp-bisection algorithms, applications and mathematical properties," *Applied Numerical Mathematics*, vol. 59, no. 9, pp. 2218–2235, Sep. 2009, DOI: 10.1016/j.apnum.2008.12.011

[215] M.-C. Rivara, C. Calderon, A. Fedorov, and N. Chrisochoides, "Parallel decoupled terminal-edge bisection method for 3D mesh generation," *Engineering with Computers*, vol. 22, no. 2, pp. 111–119, May 2006, DOI: 10.1007/s00366-006-0013-2

[216] P. J. Roache, "Perspective: A Method for Uniform Reporting of Grid Refinement Studies," *Journal of Fluids Engineering*, vol. 116, no. 3, pp. 405–413, Sep. 1994, DOI: 10.1115/1.2910291

[217] P. A. Rodriguez and M.-C. Rivara, "Multithread Lepp-Bisection Algorithm for Tetrahedral Meshes," in *Proceedings of the 22nd International Meshing Roundtable*, J. Sarrate and M. Staten, Eds. Springer International Publishing, 2014, pp. 525–540. ISBN 978-3-319-02335-9 DOI: 10.1007/978-3-319-02335-9_29

[218] C. L. Rumsey, J. P. Slotnick, and A. J. Sclafani, "Overview and Summary of the Third AIAA High Lift Prediction Workshop," *Journal of Aircraft*, vol. 56, no. 2, pp. 621–644, Dec. 2018, DOI: 10.2514/1.C034940

[219] O. Sahni, A. Ovcharenko, K. C. Chitale, K. E. Jansen, and M. S. Shephard, "Parallel anisotropic mesh adaptation with boundary layers for automated viscous flow simulations," *Engineering with Computers*, vol. 33, no. 4, pp. 767–795, Oct. 2017, DOI: 10.1007/s00366-016-0437-2

[220] R. Said, N. P. Weatherill, K. Morgan, and N. A. Verhoeven, "Distributed parallel Delaunay mesh generation," *Computer Methods in Applied Mechanics and Engineering*, vol. 177, no. 1-2, pp. 109–125, Jul. 1999, DOI: 10.1016/S0045-7825(98)00374-0

[221] J. Saltz, R. Mirchandaney, and K. Crowley, "Run-time parallelization and scheduling of loops," *IEEE Transactions on Computers*, vol. 40, no. 5, pp. 603–612, May 1991, conference Name: IEEE Transactions on Computers. DOI: 10.1109/12.88484

[222] S. P. Sastry and S. M. Shontz, "A parallel log-barrier method for mesh quality improvement and untangling," *Engineering with Computers*, vol. 30, no. 4, pp. 503–515, Oct. 2014, DOI: 10.1007/s00366-014-0362-1

[223] K. Schloegel, G. Karypis, and V. Kumar, "Parallel static and dynamic multi-constraint graph partitioning," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 3, pp. 219–240, 2002.

[224] V. Schmitt and F. Charpin, "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers," in *Experimental Data Base for Computer Program Assessment: Report of the Fluid Dynamics Panel Working Group 04.* NATO Research and Technology Organisation AGARD, May 1979, no. AR-138, pp. B1:1–B1:44.

[225] W. Schroeder, K. Martin, B. Lorensen, and I. Kitware, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics.* Kitware, 2006. ISBN 978-1-930934-19-1. https://books.google.com/books?id=rx4vPwAACAAJ

[226] J. Schöberl, "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules," *Computing and Visualization in Science*, vol. 1, no. 1, pp. 41–52, Jul. 1997, DOI: 10.1007/s007910050004

[227] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. Carns, A. Castelló, D. Genet, T. Herault, S. Iwasaki, P. Jindal, L. V. Kalé, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, and P. Beckman, "Argobots: A Lightweight Low-Level Threading and Tasking Framework," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 512–526, Mar. 2018, DOI: 10.1109/TPDS.2017.2766062

[228] E. S. Seol and M. S. Shephard, "Efficient distributed mesh data structure for parallel automated adaptive analysis," *Engineering with Computers*, vol. 22, no. 3-4, pp. 197–213, Dec. 2006, DOI: 10.1007/s00366-006-0048-4

[229] M. Shang, C. Zhu, J. Chen, Z. Xiao, and Y. Zheng, "A Parallel Local Reconnection Approach for Tetrahedral Mesh Improvement," *Procedia Engineering*, vol. 163, pp. 289–301, Jan. 2016, DOI: 10.1016/j.proeng.2016.11.062

[230] S. Sharma, C.-H. Hsu, and W.-c. Feng, "Making a case for a Green500 list," in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, Apr. 2006, pp. 8 pp.–, iSSN: 1530-2075. DOI: 10.1109/IPDPS.2006.1639600

[231] J. R. Shewchuk, “Triangle: Engineering a 2d quality mesh generator and delaunay triangulator,” in *Applied Computational Geometry Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 203–222. ISBN 978-3-540-70680-9 DOI: 10.1007/BFb0014497

[232] J. R. Shewchuk, “Delaunay Refinement Mesh Generation,” Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997, available as Technical Report CMU-CS-97-137.

[233] J. R. Shewchuk, “Tetrahedral Mesh Generation by Delaunay Refinement,” in *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, ser. SCG ’98. New York, NY, USA: ACM, 1998, pp. 86–95. ISBN 0-89791-973-4 DOI: 10.1145/276884.276894

[234] J. R. Shewchuk and H. Si, “Higher-Quality Tetrahedral Mesh Generation for Domains with Small Angles by Constrained Delaunay Refinement,” in *Proceedings of the Thirtieth Annual Symposium on Computational Geometry.* ACM, 2014, pp. 290:290–290:299. ISBN 978-1-4503-2594-3 DOI: 10.1145/2582112.2582138

[235] H. Si, “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator,” *ACM Transactions on Mathematical Software*, vol. 41, no. 2, pp. 11:1–11:36, 2015, DOI: 10.1145/2629697

[236] H. Si and K. Gärtner, “3D boundary recovery by constrained Delaunay tetrahedralization,” *International Journal for Numerical Methods in Engineering*, vol. 85, no. 11, pp. 1341–1364, Mar. 2011, DOI: 10.1002/nme.3016

[237] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis, “CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences,” Langley Research Center, Tech. Rep. CR-2014-218178, Mar. 2014, DOI: 2060/20140003093

[238] S. Soner and C. Ozturan, “Generating Multibillion Element Unstructured Meshes on Distributed Memory Parallel Machines,” *Scientific Programming*, vol. 2015, p. e437480, May 2015, DOI: 10.1155/2015/437480

[239] D. A. Spielman, S.-H. Teng, A. Üngör, I. Shang-hua, and T. Alper, “Parallel Delaunay Refinement: Algorithms and Analyses,” in *Proceedings, 11th International Meshing Roundtable*, 2002, pp. 205–217.

[240] G. L. Steele, “Making asynchronous parallelism safe for the world,” in *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL ’90. New York, NY, USA: Association for Computing Machinery, Dec. 1989, pp. 218–231. ISBN 978-0-89791-343-0 DOI: 10.1145/96709.96731

[241] G. Strang, *Linear algebra and its applications*, 2nd ed. New York: Academic Press, 1980. ISBN 012673660X

[242] H. Sukas and M. Sahin, “HEMLAB Algorithm Applied to the High-Lift JAXA Standard Model,” in *AIAA Scitech 2021 Forum.* American Institute of Aeronautics and Astronautics, 2021, DOI: 10.2514/6.2021-1994

[243] N. J. Taylor and R. Haimes, “Geometry Modelling: Underlying Concepts and Requirements for Computational Simulation (Invited),” in *2018 Fluid Dynamics Conference.* American Institute of Aeronautics and Astronautics, 2018, DOI: 10.2514/6.2018-3402

[244] P. Thomadakis, C. Tsolakis, and N. Chrisochoides, “Multithreaded runtime framework for parallel and adaptive applications,” *IEEE Transactions on Parallel and Distributed Systems.*, 2021, (under review).

[245] P. Thoman, K. Dichev, T. Heller, R. Iakymchuk, X. Aguilar, K. Hasanov, P. Gschwandtner, P. Lemarinier, S. Markidis, H. Jordan, T. Fahringer, K. Katrinis, E. Laure, and D. S. Nikolopoulos, “A taxonomy of task-based parallel programming technologies for high-performance computing,” *The Journal of Supercomputing*, vol. 74, no. 4, pp. 1422–1434, Apr. 2018, DOI: 10.1007/s11227-018-2238-4

[246] J. F. Thompson, “A survey of dynamically-adaptive grids in the numerical solution of partial differential equations,” *Applied Numerical Mathematics*, vol. 1, no. 1, pp. 3–27, Jan. 1985, DOI: 10.1016/0168-9274(85)90026-1

[247] J. F. Thompson, B. K. Soni, and N. P. Weatherill, *Handbook of grid generation*, N. Weatherill, B. Soni, and J. Thompson, Eds. CRC press, 1998. ISBN 978-0-8493-2687-5 DOI: 10.1201/9781420050349

[248] M. J. Todd, *Minimum-Volume Ellipsoids*, ser. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Jul. 2016. ISBN 978-1-61197-437-9 DOI: 10.1137/1.9781611974386

[249] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25–33, Jan. 1967, DOI: 10.1147/rd.111.0025

[250] C. Tsolakis, N. Chrisochoides, M. A. Park, A. Loseille, and T. R. Michal, "Parallel Anisotropic Unstructured Grid Adaptation," in *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics, 2019, DOI: 10.2514/6.2019-1995

[251] C. Tsolakis, N. Chrisochoides, M. A. Park, A. Loseille, and T. R. Michal, "Parallel Anisotropic Unstructured Grid Adaptation," *AIAA Journal*, Jan. 2021, accepted.

[252] C. Tsolakis, F. Drakopoulos, and N. Chrisochoides, "Sequential metric-based adaptive mesh generation," in *Modeling, Simulation, and Visualization Student Capstone Conference*, Suffolk, VA, Apr. 2018, pp. 25–35.

[253] C. Tsolakis, P. Thomadakis, and N. Chrisochoides, "Exascale-Era Parallel Adaptive Mesh Generation and Runtime Software System Activities at the Center for Real-Time Computing," Oct. 2020, (presentation). (Accessed 2021-03-08).

[254] A. Üngör, "Parallel Delaunay refinement and space-time meshing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, United States – Illinois, 2002. ISBN 9780493902661. https://search.proquest.com/docview/305620884

[255] X.-q. Wang, X.-l. Jin, D.-z. Kou, and J.-h. Chen, "A Parallel Approach for the Generation of Unstructured Meshes with Billions of Elements on Distributed-Memory Supercomputers," *International Journal of Parallel Programming*, pp. 1–31, Sep. 2016, DOI: 10.1007/s10766-016-0452-3

[256] Z. J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. T. Huynh, N. Kroll, G. May, P.-O. Persson, B. v. Leer, and M. Visbal, "High-order CFD methods: current status and perspective," *International Journal for Numerical Methods in Fluids*, vol. 72, no. 8, pp. 811–845, 2013, DOI: 10.1002/fld.3767

[257] D. F. Watson, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes," *The Computer Journal*, vol. 24, no. 2, pp. 167–172, Jan. 1981, DOI: 10.1093/comjnl/24.2.167

[258] T. Willhalm and N. Popovici, “Putting Intel® threading building blocks to work,” in *Proceedings of the 1st international workshop on Multicore software engineering*, ser. IWMSE ’08. New York, NY, USA: Association for Computing Machinery, May 2008, pp. 3–4. ISBN 978-1-60558-031-9 DOI: 10.1145/1370082.1370085

[259] M. Yano and D. L. Darmofal, “An optimization-based framework for anisotropic simplex mesh adaptation,” *Journal of Computational Physics*, vol. 231, no. 22, pp. 7626–7649, Sep. 2012, DOI: 10.1016/j.jcp.2012.06.040

[260] M. A. Yerry and M. S. Shephard, “Automatic three-dimensional mesh generation by the modified-octree technique,” *International Journal for Numerical Methods in Engineering*, vol. 20, no. 11, pp. 1965–1990, 1984, DOI: 10.1002/nme.1620201103

[261] Y. Yokokawa, M. Murayama, M. Kanazaki, K. Murota, T. Ito, and K. Yamamoto, “Investigation and Improvement of High-Lift Aerodynamic Performances in Lowspeed Wind Tunnel Testing,” in *46th AIAA Aerospace Sciences Meeting and Exhibit*, ser. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 2008, DOI: 10.2514/6.2008-350

[262] Y. Yokokawa, M. Murayama, H. Uchida, K. Tanaka, T. Ito, K. Yamamoto, and K. Yamamoto, “Aerodynamic Influence of a Half-Span Model Installation for High-Lift Configuration Experiment,” in *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, ser. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 2010, DOI: 10.2514/6.2010-684

[263] G. Zagaris, S. Pirzadeh, and N. Chrisochoides, “A Framework for Parallel Unstructured Grid Generation for Practical Aerodynamic Simulations,” in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, 2009, DOI: 10.2514/6.2009-980

[264] R. Zangeneh and C. F. Ollivier-Gooch, “Thread-parallel mesh improvement using face and edge swapping and vertex insertion,” *Computational Geometry*, vol. 70-71, pp. 31–48, Feb. 2018, DOI: 10.1016/j.comgeo.2018.01.006

[265] Z. Zhong, X. Guo, W. Wang, B. Lévy, F. Sun, Y. Liu, and W. Mao, “Particle-based anisotropic surface meshing.” *ACM Transactions on Graphics*, no. 4, 2013, DOI: 10.1145/2461912.2461946

[266] Z. Zhong, W. Wang, B. Lévy, J. Hua, and X. Guo, "Computing a high-dimensional euclidean embedding from an arbitrary smooth riemannian metric," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 62:1–62:16, 2018, DOI: 10.1145/3197517.3201369

[267] B. Y. Zhou, N. R. Gauger, C. Tsolakis, J. K. Pardue, A. Chernikov, F. Drakopoulos, N. Chrisochoides, and B. Diskin, "Hybrid RANS/LES Simulation of Vortex Breakdown Over a Delta Wing," in *AIAA Aviation 2019 Forum.* American Institute of Aeronautics and Astronautics, Jun. 2019, DOI: 10.2514/6.2019-3524

[268] O. Zienkiewicz, R. Taylor, and J. Zhu, *The Finite Element Method: its Basis and Fundamentals.* Elsevier, 2013. ISBN 978-1-85617-633-0 DOI: 10.1016/C2009-0-24909-9

APPENDIX A

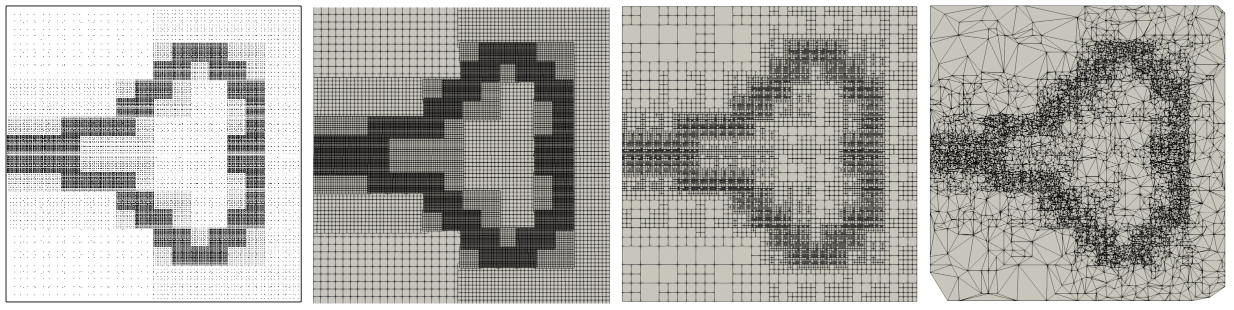
MESH ADAPTATION FOR OTHER DISCIPLINES

A.1 FEMTOGRAPHY DATA

In this work, meshes are utilized as an enabling technology that allows the creation of look-up tables that enhance Monte-Carlo simulations used in Nuclear Femtography. The problem at hand can be described as follows: Given a two- or three-dimensional probability matrix of some non-spatial variables, we attempt to create an unstructured mesh that tessellates its data while capturing all of its features.

A.1.1 TYPES OF TESSELLATIONS

The plethora of applications that utilize meshes have different requirements and restrictions, thus creating a wide variety of meshing methods [247]. Early approaches used structured tessellations that arise naturally from the matrix formulation of many problems [247]. Most problems have rarely uniform characteristics throughout the computational space and the ability to concentrate more points, and thus reduce the discretization error at regions of interest becomes a necessity. Figure 78 presents some of the most common methods of discretization.



(a) Input dataset. (b) Multiblock. (c) Quad-tree. (d) Unstructured mesh.

Fig. 78: Different types of tessellations applied on a dataset.

Elliptic and hyperbolic methods [44] provide a solution to this problem by tweaking the point density at regions of interest. Still, they are bound to the initial point distribution and they are limited by the complexity of the domain. Multi-block methods (see Figure 78b) [15] provide another way to capture features of the problem by overlaying uniform tessellations of different density. Octree methods [260] utilize quad-trees (2D) and oct-trees (3D) structures (see Figure 78c) that can capture the features of the input using a lower count of elements while at the same time providing a data structure that favor queries. Yet another method is the use of unstructured simplices (see Figure 78d) that not only further reduce the number of points and elements, but are able to scale into higher dimensions at low cost. Figure 78a depicts an example point dataset. The point density corresponds to the features we need to capture. Figures 78b-78d present different methods to create a mesh that captures the features of the input. Table 37 presents the required number of vertices for each approach.

	Uniform	Mutliblock	Quad-tree	Unstructured
# vertices	62,000	59,000	14,500	7,500

TABLE 37: Number of vertices for each of the representations of Figure 78.

Clearly, unstructured meshes are the most efficient approach in terms of number of vertices to capture point-sets that exhibit non-uniform density. Moreover, Table 38 summarizes the number of lower dimensional elements contained in each element at different dimensions. Notice that the number of vertices scales linearly for simplices while for box-based elements the growth rate is exponential. This characteristic is crucial for the approach of this work since it enables the generalization of the method to areas of interest to the Nuclear Femtography such as Deeply Virtual Compton Scattering calculations and Deep Virtual phi-meson Production calculations which utilize 5 and up to 7 dimensions respectively.

	2-cube	3-cube	4-cube	5-cube	2-simplex	3-simplex	4-simplex	5-simplex
0-face (vertex)	4	8	16	32	3	4	5	6
1-face (edge)	4	12	32	80	3	6	10	15
2-face	1	6	24	80	1	4	10	20
3-face		1	8	40		1	5	15
4-face			1	10			1	16
5-face				1				1

TABLE 38: Number of lower dimensional faces of n-dimensional cube and simplex elements.

A.1.2 METHOD

In this section we describe a pipeline designed at the CRTC lab for generating adaptive meshes for Monte-Carlo simulations. The pipeline is depicted in Figure 79.

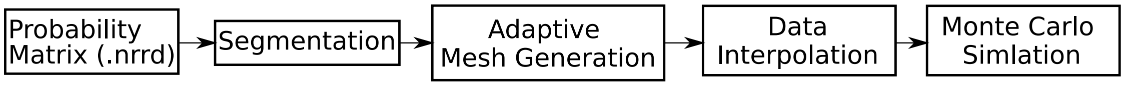


Fig. 79: Pipeline of our approach.

The input to the procedure is a probability matrix created with PARtonic Tomography Of Nucleon Software (PATRONS) [26]. For the purposes of our approach, the two- and three-dimensional probability matrices of PATRONS are treated as a 2D (3D resp.) image. The Nearly Raw Raster Data `.nrrd` format[44] was selected due to its simplicity. The segmentation filter is then applied on the image. Segmentation is the process of generating a labeled image (often called mask) that assigns to all pixels of the input a label based on a discrete number of values. These values are then used to identify regions of interest within the image. It is a requirement for the mesh generation method we use for 3D images PODM [95]. Since in

[44]`http://teem.sourceforge.net/nrrd/format.html` (Accessed 2021-07-02)

this case we are interested for the entire image the segmentation results in a trivial mask that has the same value everywhere. The adaptive mesh generation process will then create a mesh based on the segmented image. The spacing will be guided by the values of the probability matrix. Details on the adaptation process are described below. Once the mesh is generated, the original values of the probability matrix are interpolated onto the new mesh. Finally, the mesh with its interpolated values which we call “weights” can be used for the Monte-Carlo procedure.

A.1.3 ADAPTATION APPROACH

Central requirement for the meshes generated by this effort is the adaptation of the mesh to the features of the input. More specifically, in order to allow speedup for the subsequent step of Monte-Carlo simulations, the size of the elements of the generated mesh needs to be based on the rate of change of the values of the probability matrix. For our application this requirement translates into adding rules for splitting elements. The current method operates based on two user-defined criteria. First, a relative limit between the difference of the weights for a sample of points within the element. This is designed to give some control over the discretization error with respect to the input data. The second is a limit on the size of the smallest element which is used to control the size of the mesh and to guarantee termination in cases where the weight limit cannot be achieved. This sizing function has been implemented both in two and three dimensions. In 2D we use `Triangle` [231] while for 3D inputs we take advantage of PODM [95]. Figure 80 depicts two examples of the adaptation approach applied on synthetic 2D data representing a binomial Gaussian distribution and 3D data produced by PATRONS[45]. For the 2D case, the 1,000,000 values of the image are decimated to $3,805$ vertices and $7,509$ triangles. The 3D input has 1,000,000 values while the generated mesh $42,217$ vertices and $257,041$ tetrahedra. Figure 81 presents a different view of the 2D problem. The 3D histogram that corresponds to the values of a binomial Gaussian distribution (left) is reconstructed by linearly extruding the mesh triangles to prisms based on the interpolated weights.

The adaptation approach allows to decimate probability matrices given as 2D/3D array while capturing the features of the distribution efficiently. More importantly, the resulting mesh can aid towards speeding up significantly event generation for Physics calculations. Quoting from [107]: *“Generating grids using PARTONS took 4 days (1M computations, per*

[45]We would like to thank Gagik Gavalian and Pawel Sznajder for proving the input probability distributions.

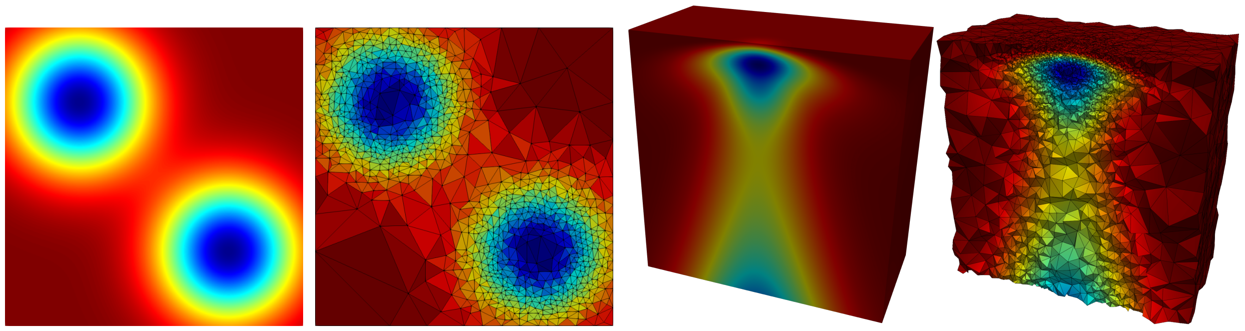


Fig. 80: Two- and three-dimensional demonstration of the approach.

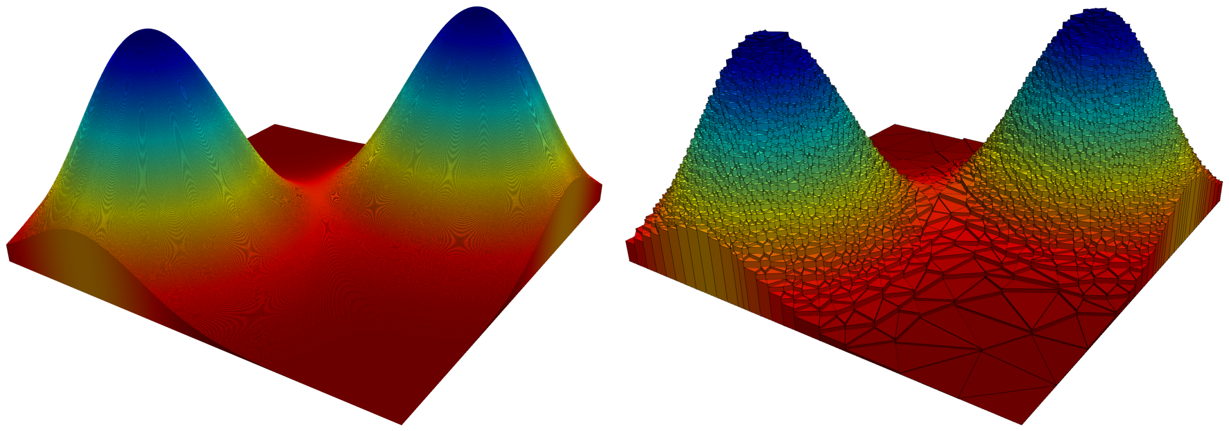


Fig. 81: Decimation of a 2d histogram that corresponds to a binomial Gaussian distribution.

grid point), [However] Event generation (10M events) takes $\tilde{2}0$ minutes".

A.1.4 ADAPTIVE PIPELINE

The ability to generate adaptive meshes based on the probability distributions of PARTONS, enables also to build an adaptive pipeline similar to the one presented in Figure 47 in page 98. In particular, since the PARTONS evaluations are computationally expensive, one could start the procedure with only a few events and a coarse mesh. The generated mesh can then be utilized to speedup the process. Instead of generating events (Monte Carlo trials) within the whole subdomain, one can guide PARTONS to generate events only within few elements of interest based on some user-defined criterion. If this criterion is derived from an error-based estimator one can build an error-based adaptive pipeline, see for example Figure 82.

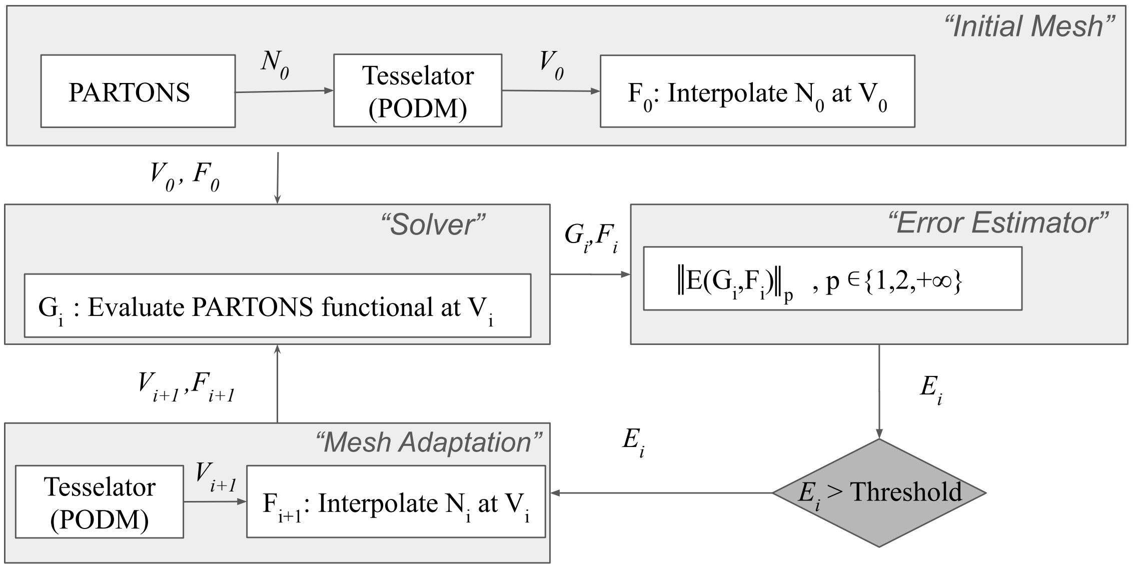


Fig. 82: An error-based adaptive pipeline based for Nuclear Femptography data. N_i: image values, V_i: mesh , F_i: interpolated values on mesh V_i, G_i: "ground-truth" values.

The initial mesh can be constructed based on a (coarse) probability matrix of PARTONS similar to the approach presented so far. PARTONS can then be used as a solver estimating its functional on the vertices of the mesh. Based on a norm defined on the ground-truth and the interpolated values one could extract an error-indicator that drives the interpolation procedure that will generate a new mesh for the next iteration of the pipeline. Notice that this pipeline can also be implemented based on metric-based adaptation approaches. If E is the interpolation error, one could construct a multi-scale metric and adapt the mesh in a similar manner with our approach in Section 4.2.

A.2 ADAPTIVE MESH GENERATION FOR MEDICAL DATA

In this section, we attempt to apply mesh adaptation techniques in the context of Medical Imaging. In particular, we focus on generating adapted meshes for non-rigid registration (NRR) of pre- and intra-operative images. NRR is used in image-guided neurosurgery and aids towards "matching" sections of pre-operative medical data (e.g., Magnetic resonance imaging (MRI), ultrasound, etc.) to intra-operative. This step is crucial to neurosurgical operations since it allows for the identification of tissues in the intra-operative image that need to be removed (e.g., tumor volumes). The registration is called non-rigid because the underlying model treats the brain as a non-rigid object susceptible to deformation and shift during the surgical operation. We evaluate our approach within the Adaptive Physics-Based Non-Rigid Registration (A-PBNRR) method [83]. Figure 83 depicts a high-level view of the A-PBNRR pipeline. For a complete description see [83].

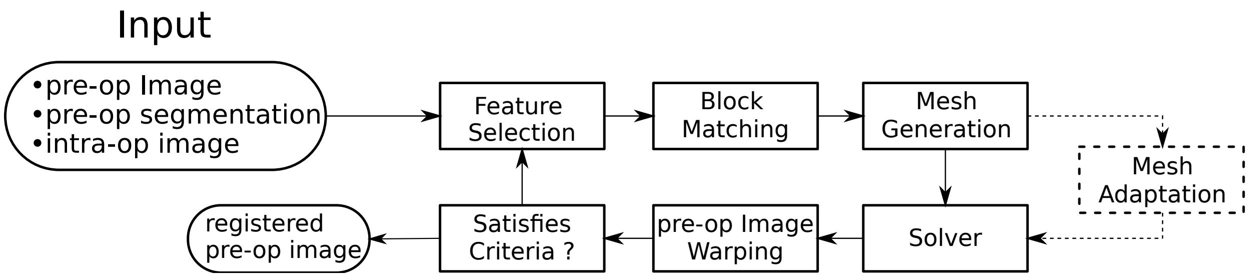


Fig. 83: High-level A-PBNRR pipeline. Solids boxes correspond to the original approach. The Mesh Adaptation module represents our attempt in the context of the A-PBNRR pipeline.

The method accepts as input the pre- and intra-operate images along with a segmentation of the pre-operative image. The pre-operative segmentation is a labeled copy of the pre-operative image where each pixel is labeled based on the tissues of interest. In the first step, the method identifies features of the pre-operative image based on computer vision techniques that identify regions of high intensity variance. The Block Matching step will then search for pairs of blocks in the pre- and intra-operative image that maximize a similarity measure. Mesh generation creates a mesh based on the segmented image. The outputs of the three steps are then combined in the solver stage that evaluates displacements at each vertex of the mesh based on information of the other two steps. The displacements are then applied to the pre-operative data by warping the image. Based on user-defined criteria, the method may apply more iterations by using as input a warped pre-operative image and segmentation.

In the current evolution of A-PBNRR, the mesh generation steps use the Parallel Optimistic Delaunay Meshing (*PODM*) method [95] that can generate a mesh that faithfully captures (with geometric guarantees) the surface of the input image and the interface between different tissues. However, it does not consider any information about the registration points (landmarks) recovered by the Block Matching step. In [86] the authors incorporate the distribution of landmarks over the mesh into the mesh generation step using custom sizing functions for two different mesh generation methods (Delaunay refinement and Advancing Front). The goal of these modifications is to equi-distribute the landmarks among the mesh elements which is expected to improve the registration error. The evaluation presented in [86] was based on synthetic deformation fields and showed that indeed these modifications reduce the registration error. Our approach focuses on applying the same sizing function in order to validate the effectiveness of the method. Moreover, preliminary results on applying mesh adaptation methods that originate from the Computational Fluid Dynamics field are presented. For completeness, a summary of the method employed in [86] is presented along with the modifications that can turn it into an anisotropic metric-based method.

The equi-distribution of the registration points can be formulated as assigning approximately the same number of registration points at each mesh vertex cell complex, where a mesh vertex cell complex is defined as the set of all the elements attached to a vertex. See, for example, Figure 84. On the left, the vertex cells of p_1, p_2, p_3 have $3, 7$ and 5 landmarks, respectively. While on the right, by collapsing edge p_2p_1 one can equi-distribute the landmarks. Both the vertex cells of p_1 and p_2 have now 7 landmarks.

The crux of the method is to set the local spacing at each vertex equal to the distance to the k-th closest registration point. Assuming an ideal spacing, the mesh vertex cell complex of each vertex will contain k registration points. An illustration for $k = 5$ is given in Figure 85 left. Notice that another way to interpret the sizing constrain at each vertex is using a sphere centered at each mesh vertex with a radius equal to the distance to the k-th registration point. This method creates adaptive meshes but, it does not capture the local

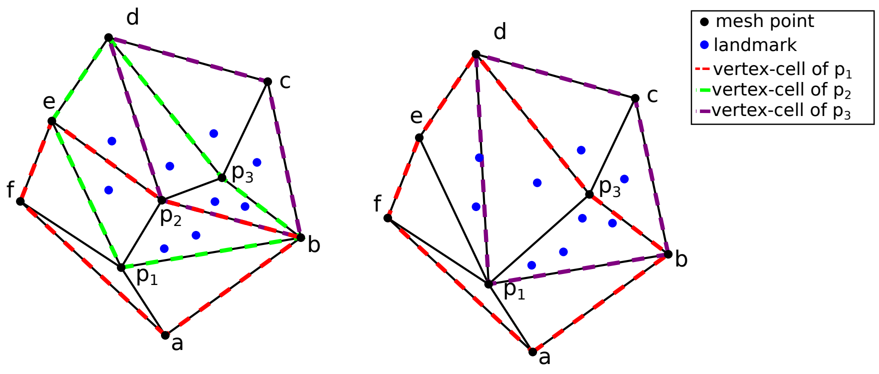


Fig. 84: Optimizing landmark distribution.

density of the landmarks efficiently due to the fact that only the k-th point is used and the relative locations of the rest $k-1$ landmarks are ignored. Building upon this observation one can replace the spheres at each vertex, with the smallest bounding ellipsoid that contains the k closest registration points and is centered at the given vertex. Describing the local spacing as an ellipsoid gives the ability to capture the local distribution of the landmarks better due to the increased degrees of freedom of an ellipsoid in comparison to a sphere (see for example Figure 85 right).

Creating the minimum volume ellipsoid that encloses a given point-set is a well-studied problem in the optimization literature [248]. The constructed ellipsoid has a natural mapping to a 3x3 positive definite matrix [80] that can be used as a metric that guides the anisotropic mesh adaptation procedure. In order to give to the mesh adaptation procedure more flexibility, an additional "inflation" parameter a is introduced that is common for all the points and allows to enlarge all ellipsoids by a constant factor. The goal of this parameter is to allow the mesh generation procedure to perform operations that may not conform to the strict size but improve the overall result (see for example Figure 85 right).

Generating an adapted mesh begins by initially creating a uniform isotropic mesh $PODM$. The generated mesh along with the landmarks identified by the Block-Matching step are used to build a metric field. The metric field is constructed by iterating, in parallel, the mesh vertices and evaluating the k-closest registration points using a k-nn search from the Visualization Toolkit (VTK) library [225]. The minimum volume bounding ellipsoid is constructed using the Khachiyan algorithm [139] which we implemented in the Eigen

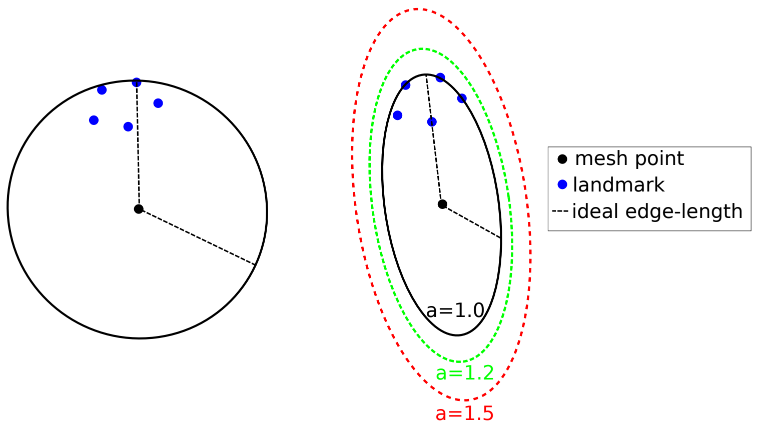


Fig. 85: Visualization of the metric construction for mesh adaptation. Left: Isotropic metric that set the spacing equal to the distance of the 5th closest registration point. Right: Anisotropic metric based on the five registration points for different values of the inflation parameter a.

library [116] based on publicly available implementations of the algorithm[46]. Directly using the landmarks (Figure 86 (b)) will not yield an ellipse centered at a mesh point. Including the mesh point into the input of the minimum ellipsoid algorithm does not fix the issue either (see Figure 86 (c)). Instead, we generate reflections of the k-closest landmarks by the mesh point and include them in the input of the minimum ellipsoid algorithm. Due to symmetry, the mesh point will always be in the center of the constructed ellipsoid. Finally, the mesh is adapted using MMG3D [70, 183].

The results of augmenting mesh adaptation to the A-PBNRR method for two of the cases described in [84] are presented in Table 39. The accuracy of the registration is evaluated using two quantitative metrics. The first described in [104] and expressed in the first column, uses the Hausdorff distance [65] between point-sets extracted from the transformed pre-operative image and the intra-operative image. The second method uses six anatomical landmarks selected by a neurosurgeon and located into both the pre- and intra-operative image. Statistics (min, max and mean) related to the distance between the transformed pre-operative points as evaluated by A-PBNRR and the neurosurgeon's suggested points are expressed in the corresponding columns.

[46]Nima Moshtagh (2021). Minimum Volume Enclosing Ellipsoid (`https://www.mathworks.com/matlabcentral/fileexchange/9542-minimum-volume-enclosing-ellipsoid`) MATLAB Central File Exchange, (Accessed 2021-06-28).

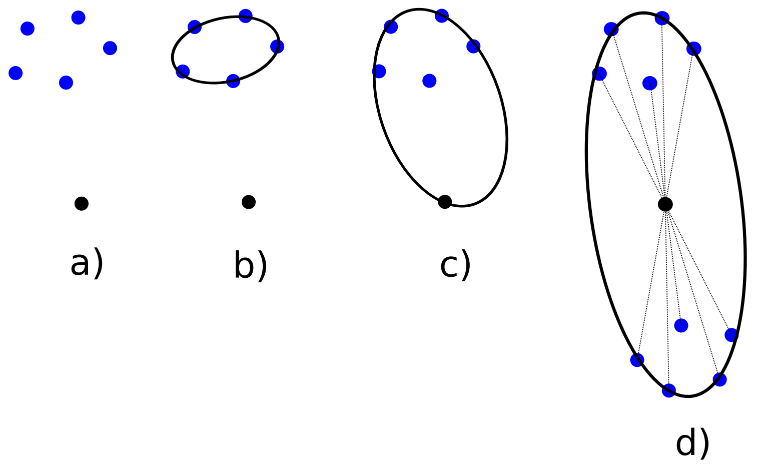


Fig. 86: Different approaches to constructing a metric utilizing the minimum ellipsoid method.

Method	HD (mm)	Min error (mm)	Max error (mm)	Mean error (mm)	# vertices	# elements
			case 9			
Baseline	2.24	1.07	5.90	3.51	3,264	13,210
Isotropic	1.95	1.22	7.53	3.71	4,177	19,893
Anisotropic (a = 1.0)	2.22	0.55	7.85	3.99	4,520	22,383
Anisotropic (a = 1.2)	2.00	1.01	7.10	3.70	3,629	17,593
Anisotropic (a = 1.5)	2.64	0.93	6.15	3.25	2,838	13,291
			case 18			
Baseline	4.06	2.06	5.37	3.65	2,833	11,040
Isotropic	3.42	2.29	5.76	3.92	4,008	19,466
Anisotropic (a = 1.0)	3.71	2.12	5.50	3.96	4,460	22,342
Anisotropic (a = 1.2)	4.05	2.06	5.05	3.61	3,766	18,077
Anisotropic (a = 1.5)	4.05	1.92	5.17	3.65	2,983	13,812

TABLE 39: Results applying mesh adaptation to the A-PBNRR pipeline.

The number of registration points per mesh cell vertex is set to $k = 500$. This value was selected since it produces meshes with a vertex count close to the baseline meshes. The first row of each dataset corresponds to the base case of using A-PBNRR with no adaptation. The isotropic rows indicate the application of the method described in [86]. In practice, we build an isotropic metric composed out of spheres with radii corresponding to the distance of the k-th closest landmark at each mesh vertex. Isotropic adaptation reduces the Hausdorff distance by almost 13% for case 9 and almost 16% for case 18. However, it comes with the price of generating a mesh twice as big for both cases. The rest of the rows correspond to applying anisotropic mesh adaptation. Anisotropic mesh adaptation produces a higher Hausdorff distance error. However, it decreases the landmark-based error when compared to the isotropic method and in some cases, performs better than the baseline. Moreover, by using an inflation parameter of 1.5 we were able to produce meshes with a size comparable to the baseline.

Although these results are preliminary, they indicate that the problem of generating an "ideal" mesh for image registration purposes includes competing evaluation criteria like the minimum mesh size, Hausdorff distance, and the landmark-based error above. The Hausdorff distance could be improved by tweaking the `-hausd` parameter of the MMG3D code that controls the distance between the surface of the input and the output meshes. Moreover, one could attempt to intersect the generated metric field with one constructed based on the features of the input mesh which has fidelity and quality guarantees on the surface since it was produced by *PODM*. Such a mesh could be created using the implied metric of the surface mesh or a feature-based metric similar to the approach we used fpr curved domains in Section 3.3. Introducing mesh adaptation to A-PBNRR has the potential to improve its effectiveness, but further investigation is needed to optimize its parameters.

APPENDIX B

PARALLEL MESH GENERATION CHALLENGES

B.1 POWER CONSUMPTION ASPECTS OF MESH GENERATION

The Top500[47] is an online list maintained by the supercomputing community that first published in 1993 [178] and reports the top 500 supercomputers over the world based on their speed in terms of floating-point operations per second (Flops) for the LINPACK benchmark [81]. Since its conception, another important characteristic of the performance of a supercomputer has emerged and that is its energy consumption. In [230] the authors presented data on the high amount of power required to operate and regulate the temperature of supercomputers in the Top500 list and advocated for the creation of the Green500[48] list that compares supercomputers based on their energy efficiency in terms of Flops/Watt. Since then, the energy efficiency of the top supercomputers in the list has improved by three orders of magnitude from 0.147 GFlops/Watt for Blue-Gene\L in 2007 to 26.195 GFlops/Watt for NVIDIA DGX SuperPOD in 2020[49]. Still, the there are very few efforts optimizing a parallel mesh generation application for energy consumption [213].

In this section, we discuss our attempt to optimize the energy aspects of mesh generation of *PODM* [95] and present some preliminary results. At the time of contacting this analysis the available profilers were not ready yet for fine-grained power measurements of highly complex applications such as mesh generation codes. So, as a first step we attempted to optimize the performance of the code and as a consequence its power consumption. For this study we used the Abstraction-Level Energy Accounting (ALEA) [185] energy profiler[50]. ALEA allows to measure energy consumption at a fine-grain level that goes lower than the sampling rate by combining physical power measurements with a probabilistic model that distributes energy among code blocks. Based on instruction-level information of the profiler, we identified the most energy-consuming functions of the code which corresponded to the rollback treatment code of the Load balancing sections of Figure 63a and the most expensive floating-point operations which were the circumcenter evaluation and the predicate-based orientation tests. Then, we applied the following optimizations to the code:

[47]`https://www.top500.org/lists/top500/` (Accessed in 2021-06-27).

[48]`https://www.top500.org/lists/green500/` (Accessed in 2021-06-27).

[49]`https://www.top500.org/lists/green500/2020/11` (Accessed in 2021-06-27).

[50]We would like to thank Lev Mukhanov for sharing the ALEA code and assisting towards performing the analysis of this section.

- Enforcing "-ffast-math" and "-O3" to the above functions by using gcc 's attributes (gcc version version 4.9 was used) `__attribute__((optimize ("fast-math"))` and `__attribute__((optimize(3)))`
- Enabling Advanced Vector extensions (AVX) during compilation (`-mavx` for gcc)
- Using of `static inline` instead of `inline` on the most expensive floating point functions.

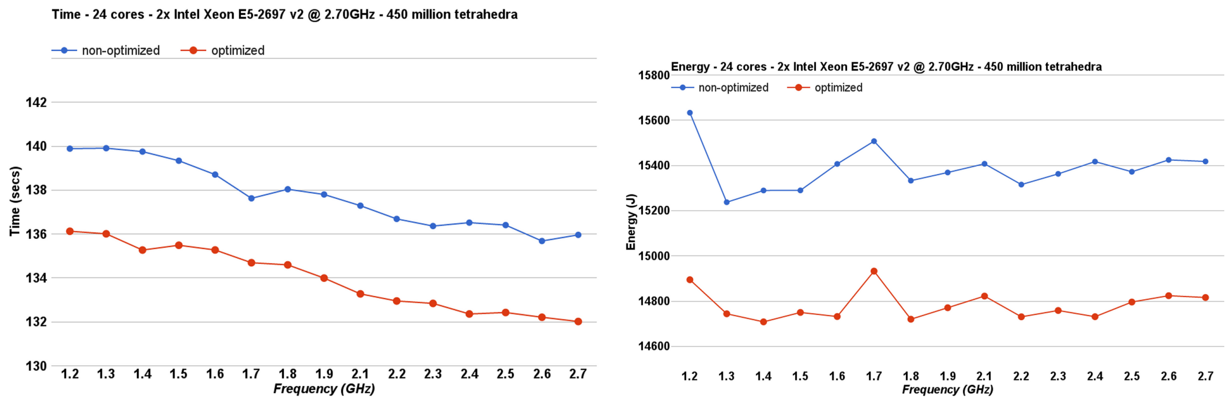
To study the effect of these optimizations we employed Dynamic Voltage and Frequency Scaling (DVFS) by modifying the frequency of the CPU's cores before running each experiment. To alter the frequency we utilized `cpufreq`[51]. Figure 87 presents our results on two different nodes: A two-socket Intel®Xeon E5-2697 v2 @ 2.70GHz with 24 cores in total, and a 4-socket Intel®Xeon E5-4610 v2 @ 2.30GHz with 32 cores in total. The original and the optimized code were executed 5 times and the median values are reported in the figures.

The results indicate that the mesh generation code behaves differently when running on these machines and the optimization affects it in different ways. On the older machine, (2nd generation Xeon) in Figure 87a, scaling the frequency affects the running time by a small amount. The optimizations yielded an 4% improvement on the energy when using all the cores. The performance on the newer 4th generation Xeon in Figure 87b increases linearly with the frequency. Moreover, by comparing the energy consumption between the two models, one can see that the newer generation processor uses less than half the energy although it runs for a longer time. On the other hand, it is affected less by the optimizations. The energy gain is only 0.3% on average.

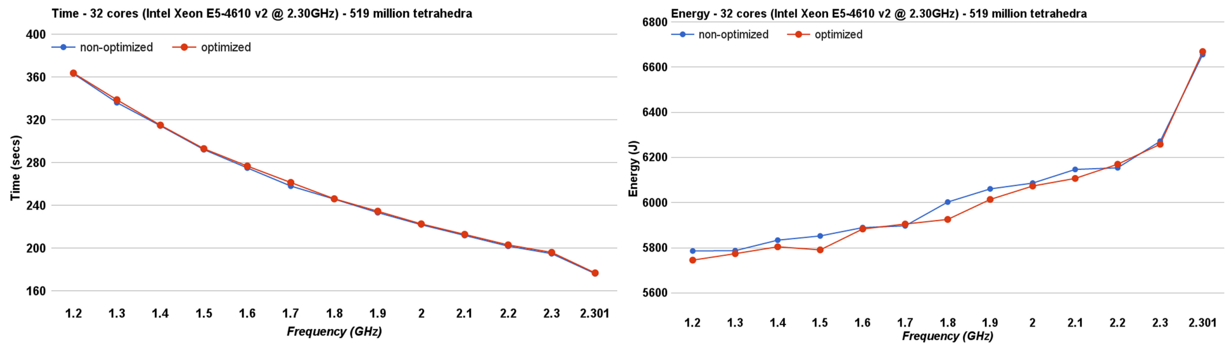
In conclusion, these preliminary data indicate that currently, the power gains for a complex mesh generation application are limited and require low-level optimizations. Also, any optimizations may diminish once the application is executed on newer hardware.

An alternative approach to the one presented could be to lower power-aware optimizations into the tasking framework. Models [150] and methods [179] for optimizing tasking schedulers for power consumption already exist and they could be combined with our tasking approach presented in Chapter 5.

[51]`https://www.kernel.org/doc/Documentation/cpu-freq/index.txt` (Accessed 2021-06-27).



(a) Time and energy data for 2nd Generation Intel®processor.



(b) Time and energy data for 4th Generation Intel®processor.

Fig. 87: Data of power-aware analysis.

B.2 PARALLEL CONSTRAINED MESH REFINEMENT IN THREE DIMENSIONS

One of the layers of the *Telescopic Approach* (see Figure 1 on page 7) is the Parallel Constrained layer. This section summarizes our effort and findings towards creating a three-dimensional parallel Constrained Delaunay mesh refinement method.

To the best of our knowledge, the first two-dimensional Parallel Constrained Delaunay algorithm was presented in [55]. Although the scalability was limited, it introduced the idea of Parallel Constrained Delaunay Meshing: Each subdomain will be treated as an independent mesh which will be refined in parallel. Conformity along the shared edges is accomplished by sending split messages to the neighbor subdomain. Later in [52], the authors presented a new implementation, called Parallel Constrained Delaunay Mesh (PCDM), with many improvements that enable the method to scale up to 100 processors efficiently. PCDM uses the Medial Axis Domain Decomposition software (MADD) presented in [152] to over-decompose the input geometry into several subdomains. MADD, in contrast to generic graph-based decomposition methods, produces separators that do not introduce low quality features in the subdomains. Subdomains are then mapped to the processors using METIS [138] and distributed using MPI similar to the previous method. The high quality of separators allowed to apply a custom mesh generation implementation employing the Constrained Delaunay Algorithm in each subdomain. Shared boundary conformity is achieved by sending split messages which encode the fractions that the split should create with respect to the initial segment avoid thus any round-off errors.

In this section, based on ideas and methods presented in the two-dimensional implementation, we present preliminary results related to a token-based communication scheme for parallel mesh generation. Using this scheme, we implement and evaluate a three-dimensional parallel constrained Delaunay mesh generator that introduces minor overheads but depends on a good initial domain decomposition. Table 40 presents the classification of the method based on the criteria of Section 2.1.

Coupling	Sync.	Gran.	Method	P. Model	Decomp.	Prog.
Partially	Asynchronous	Coarse	Constrained Delaunay	MPI	D. Domain	No

TABLE 40: Classification of the method of this section.

Figure 88 depicts the steps of the method when updating a constrained face between two subdomains. Initially, the interface between the two subdomains is conforming and points can be inserted independently as long as they do not encroach upon any shared face (see Figure 88a). Encroachment for a point p with respect to a face is defined using the circumscribed sphere of the triangular face. If p is inside this sphere, it is rejected and its projection on the face q is inserted instead (see Figure 88b). The cavity of q is then evaluated (highlighted green elements in Figure 88c) and the elements are removed (see Figure 88d). The cavity will then be triangulated and the point will be sent to the neighboring subdomain (see Figure 88e). Finally, the subdomain on the receiving side will repeat the steps resulting in a conforming interface (see Figure 88f).

B.2.1 IMPLEMENTATION

In this section, we present an implementation of the proposed algorithm using TetGen [235] as the underlying mesher. Our method decomposes the initial mesh and then refines it in parallel. Every subdomain is treated as a separate mesh and conformity across the shared boundaries of the subdomains is achieved by exchanging information for every point introduced on the shared boundary. The procedure is composed of three steps: Mesh Decomposition, Subdomain Distribution, and Parallel Refinement.

B.2.1.1 Mesh Decomposition

The lack of a proper domain decomposition method for three-dimensional geometries restricted the options to a general-purpose mesh decomposition scheme like the PxQxR method [58] and the Graph-based METIS library [138]. The input can be either a surface mesh or a coarse tetrahedral or hexahedral volume mesh that conforms to the boundary of the object of interest. In the case of a surface mesh, TetGen is used sequentially in a prepossessing step and produces a coarse Constrained Delaunay volume mesh. Using either of the decomposition methods, a map between all the elements and n subdomains is constructed sequentially, where $n > P$ and P is the available number of processors. Once the map is evaluated, the boundary of each subdomain is extracted along with the adjacency information between them.

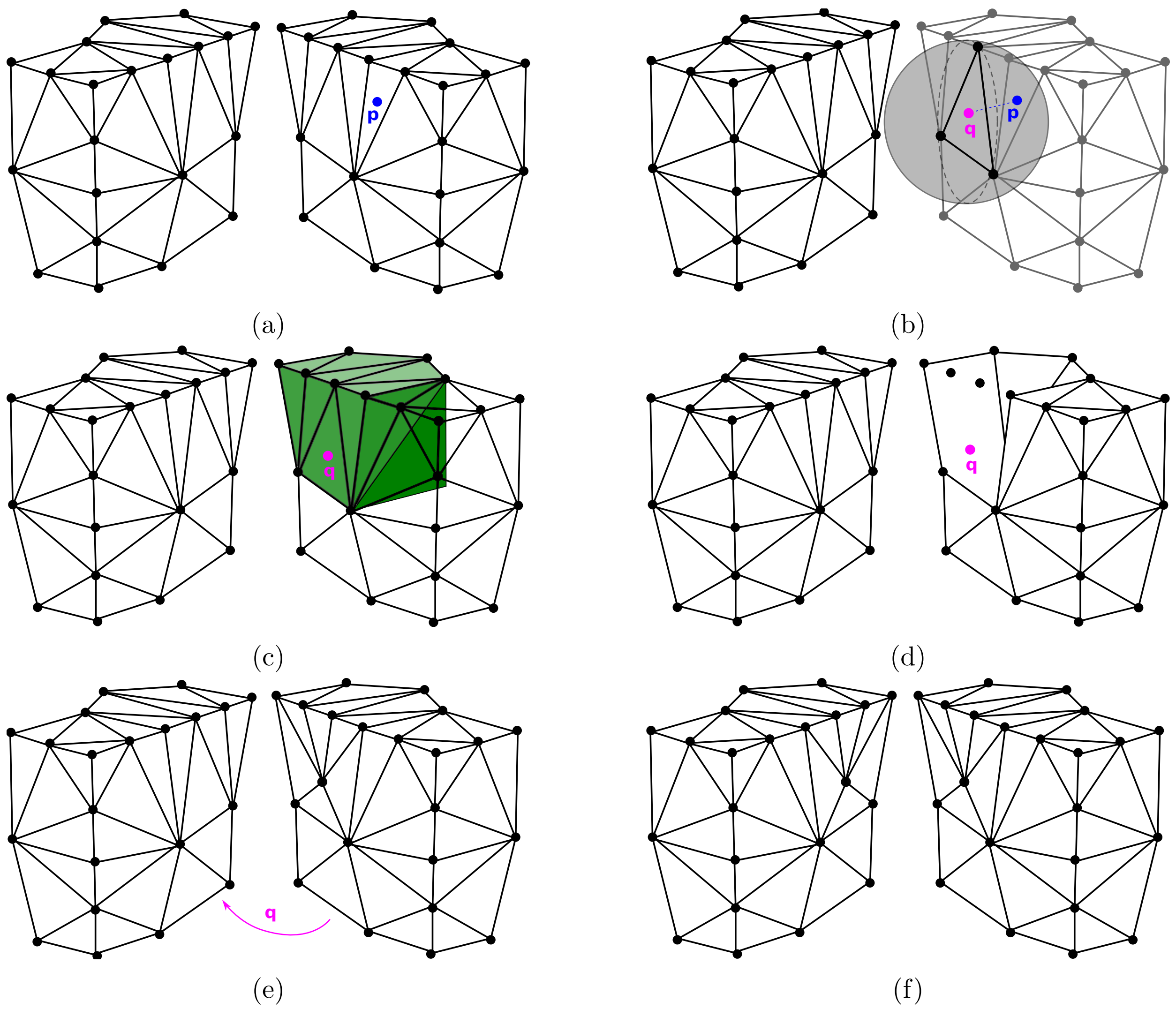


Fig. 88: Steps of the PCDM method in three dimensions.

B.2.1.2 Subdomain Distribution

Creating more subdomains than processes gives the method a better chance to equi-distribute the load. The proposed method uses information acquired from the previous step and distributes the subdomains across the processes in a configuration that reduces the communication among them. Similarly to [52] we utilize the dual graph of the subdomains. In particular, we create a graph G with its nodes representing the subdomains and its edges the interfaces between them. The weight of a vertex corresponds to the volume of the subdomain. The weight of an edge corresponds to the size of the interface. We define the size to be the sum of the areas of all constrained triangles and the lengths of all the constrained edges on the interface. METIS is then used to partition G into n parts by minimizing the cost of cutting graph-edges and equi-distributing the graph vertices among the partitions taking into account their weights. The resulting mapping between the subdomains and the processes is then used to distribute the subdomains from the root process.

B.2.1.3 Parallel Refinement

In the parallel refinement stage, each subdomain is unpacked, and an instance of TetGen is initialized, having as input the boundary of the subdomain. If the boundary is not composed solely out of triangles, it is triangulated, and then using the Boundary Recovery algorithm of TetGen, an initial volume mesh satisfying the constrained Delaunay property is created. We have chosen this approach instead of using the built-in capabilities for reconstructing the mesh out of the tetrahedra of the subdomain because we have found that the reconstruction function is prone to round-off errors and very often cannot detect constrained edges. This limitation created the need for the *Reproducibility* criterion, which we describe in Section 1. We have presented a more detailed example of this issue in [62]. Since the initial coarse mesh of the subdomain was derived out of a constrained Delaunay mesh, it is a constrained Delaunay mesh itself. Our expectation is that TetGen can recover its boundary without adding additional points on the surface. No extra points on the boundary, imply that the shared interfaces remain untouched and, thus no communication is needed at this stage. TetGen has met our expectation in the cases we studied, however more research and experimentation is required in order to have a guarantee that this will always be the case for TetGen.

The algorithm proceeds by refining the subdomains in parallel in each subdomain while sending any modifications performed on shared boundaries following a predefined communication scheme. More details about the scheme are presented in the next section.

The version of TetGen we utilized (1.5.0) performs the mesh refinement in three stages as follows: Let a be the user-defined upper volume bound for the tetrahedra in the mesh. In the first stage, TetGen splits all the encroached segments as well as all the edges that have a length larger than $\sqrt[3]{a}$. In the second stage, all the faces that are encroached or have an area larger than $\sqrt[3]{a^2}$ are split. Finally, all the tetrahedra are split until the requested quality and volume constrains are met for all the elements except those near small input angles.

Following the same approach within our method would create a problem since all the shared boundaries are split up-front, thus accumulating all the communication in the first two stages. Moreover, this approach causes almost every point insertion to trigger a message which creates network congestion. Since the Constrained Delaunay Mesh refinement rules [234] do not require splitting the edges or the faces according to some quality metric, we restructured the refinement operation by removing the first two stages. This modification increased the running time of TetGen up to 5% in some cases, but it spread the communication across the entire run of the algorithm, thus distributing the load on the network more equally throughout the execution of the algorithm. It should be noted, that removing these steps did not affect the robustness of the algorithm, and the difference in quality measured both by dihedral angles, and the radius-to-shortest-edge ratio was negligible.

The final modification performed on the TetGen code was to replace the predicates with the ones provided by the Predicate Construction Kit (PCK) [169]. The reason for this switch was that in order to guarantee conformity in the presence of co-circular point we need a robust predicate for the 3D in-circle test with support of symbolic perturbation.

B.2.2 CHALLENGES OF CREATING A RELIABLE COMMUNICATION SCHEME

Since the subdomains are refined in parallel, there is a need of synchronizing the changes on the shared boundary while keeping the overhead low and maintaining the constrained Delaunay property within each subdomain. The communication scheme needs to be flexible enough to cover the needs of Constrained Delaunay refinement in three dimensions. As it is shown in [234], the constrained Delaunay refinement of domains with small angles requires more advanced rules than the traditional ones [232]. In particular, the split of a single tetrahedron may require the insertion of multiple points to restore the constrained Delaunay property of the domain. For the communication scheme, this means that some point insertions should be bundled together, and the receiver should perform the same steps and in the same order.

A further requirement of the communication scheme is to avoid the rounding errors that will unavoidably appear between different subdomains since, in general, the order of the floating-point operations performed on the interface between two subdomains is not the same. Also, in contrast to the two-dimensional case, in our case, we may have to establish communication between more than two subdomains since a segment may belong to the shared interface of more than two subdomains.

The proposed communication scheme could resolve the first issue by packing together all the points required to split an edge/face and restore the constrained Delaunay property. Although this approach should work in theory, in practice, we discovered that quite often, the receiving subdomain would declare a cavity as invalid and reject the incoming point. This case seems to appear when the shared segment is an edge of a sliver as in Figure 89. Since the decomposition method we use has no control over the quality of the interfaces in terms of angles, we created synthetic data for our experiments. In particular, we used as an initial mesh the structured grid[52] composed of cubes shown in Figure 90. Any decomposition for this input would result in right angles which do not create the aforementioned problems.

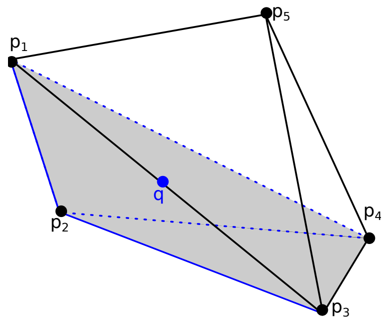


Fig. 89: A configuration where a remote point was rejected by TetGen. A sliver $p_1p_2p_3p_4$ shares the face $p_1p_3p_4$ with the tetrahedron $p_1p_3p_4p_5$. In this case q was rejected by TetGen because its cavity was classified as invalid.

Floating-point issues are handled by including the points of the to-be-split edge/face in the message. Conformity along the three-dimensional interfaces cannot be achieved just by utilizing the asynchronous point insertions described in the two-dimensional implementation of this approach [52] since a point can now be inserted anywhere on the interface and not just at the midpoint of a shared edge. Also, interfaces can be shared by more than two subdomains, while in two dimensions the subdomains around a shared interface are

[52]The grid is constructed out of a 3D image by converting each voxel (3D pixel) to a cube. We would like to thank Fotis Drakopoulos for providing the software that performs the conversion.

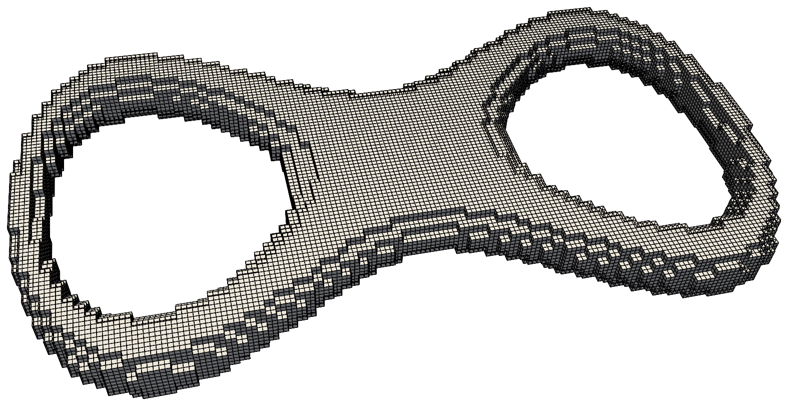


Fig. 90: Double torus, the initial mesh of our experiments 67,000 hexahedra, 80,600 points.

always two. In our approach, conformity enforced by serializing the modifications on the interface utilizing distributed locks in the form of tokens. The next section presents two communication models created to handle the updates on the interfaces.

B.2.3 COMMUNICATION MODELS

This section presents two communication models that can be used with the meshing approach presented so far. Their main difference is in the way interfaces are defined. The first approach generates interfaces that cannot intersect, and thus a unique token for each interface is enough to handle the communication. For example, in Figure 91 the first model would define five interfaces; four on the outer section of the torus and one in the center. One the other hand, the second model introduces two types of interfaces: *face interfaces* that are defined between a pair of two subdomains and *segment interface* that are defined at the intersection of face interfaces. Based on the second model, Figure 91 contains 15 face interfaces and 6 segment interfaces.

Both methods utilize asynchronous messages and thus termination cannot be trivially detected. Even if all subdomains have been refined, messages may still be in transit. To resolve this issue, we use a circular token transmission scheme described in [78] and used in [52].

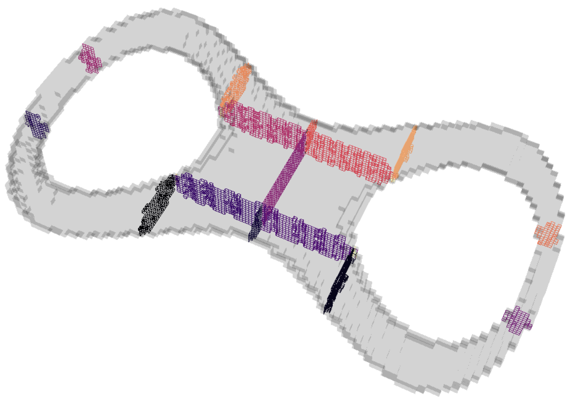


Fig. 91: Double torus with interfaces highlighted.

B.2.3.1 First approach: Unique token per interface

The union of segments and faces shared between two subdomains, $\mathcal{S}_1, \mathcal{S}_2$ is called **Interface** and will be denoted by $\mathcal{I}(\mathcal{S}_1, \mathcal{S}_2)$ or just $\mathcal{I}$ whenever the subdomains can be inferred form the context. Interfaces should not intersect. To achieve this requirement, we utilize a region-grow approach. In particular, starting from interface triangles, we create interfaces by appending neighboring interface triangles and edges. If two interfaces meet, we merge them into one. The algorithm operates under the following rules:

1. Each interface $\mathcal{I}$ is equipped with a (unique) token $t(\mathcal{I})$ and a global interface Id (iid).

2. For each group of subdomains that share an interface a cyclic order (ring) $R(\mathcal{I})$ is defined resulting in a unique id for each subdomain relative to the group that belongs.

3. Interface $\mathcal{I}$ can be modified by $\mathcal{S}_i$ only if $\mathcal{S}_i$ holds $t(\mathcal{I})$.

4. If $t(\mathcal{I})$ is required but missing from $\mathcal{S}_i$ it can be requested from the owner of the token.

5. The token has two states: empty or it holds a pair of (point, the to-be-split segment/face) and the id of the subdomain that initiated the transmission.

6. When a non-empty token $t(\mathcal{I})$ is received, if the target segment/face belongs to the receiver it is split with the provided point. If the next subdomain $\mathcal{S}_j$ on $R(\mathcal{I})$ is not

the one that initiated the token transmission then $t(\mathcal{I})$ is sent to $\mathcal{S}_j$.

Additionally, each subdomain is equipped with four maps indexed by the global interface Id:

1. `tokenIsHere[iid]` indicates whether the token for interface `iid` is present.
2. `tokenOwner[iid]` holds the current owner (or the next owner in case there is an on-going transmission[53]) of the token of the interface `idd`.
3. `requestSent[iid]` indicates whether a token request has been sent since the last synchronization. This map prevents the algorithm from sending an excessive number of messages.
4. `requestReceived[iid]` indicates whether any request has been received since the last synchronization for interface `iid`.

Initially the tokens are distributed arbitrarily among the neighbors of the interfaces. The locations of the tokens are broadcasted among the subdomains.

Main refinement procedure: Each subdomain $\mathcal{S}_i$ is being refined by processing elements from a queue Q. When the steiner point p of an element belongs to an interface a check for token ownership is performed.

- If the token is present, a token transmission is initiated: p is inserted in the mesh and copied into the token together with the segment/face that was split and the id of the subdomain. The subdomain then sends the token to the next subdomain in $R(\mathcal{I})$ updates the value of the owner of the $t(\mathcal{I})$ to the previous subdomain in $R(\mathcal{I})$ and sets `requestSent` for $t(\mathcal{I})$ to `false`.
- If the token is not present, then the element that caused the creation of p is pushed into a pending queue PQ, a token request is sent to the owner of the token as indicated by `tokenOwner` and `requestSent` is set to `true`. The reason of the last step is to avoid sending unnecessary many requests.

[53]The next owner is the last in $R(\mathcal{I})$ with respect to the Subdomain that initiated the token transit.

Incoming Messages: During the course of the above steps a message may be received. This can be a token, a token request or an empty token.

- If it is a token request, and $\mathcal{S}_i$ holds currently the token. It sends the token to the sender of the request. If $\mathcal{S}_i$ does not hold the token it sets `requestReceived` to `(true,source)`. The value of `requestReceived` will be used as hint: when $\mathcal{S}_i$ no longer uses the token it will send it to $\mathcal{S}_{source}$.

- If the token contains a point p, then the to-be-split segment/face g is checked: If g belongs to the boundary of the subdomain, p is inserted and the token is sent as-is to the next on $R(\mathcal{I})$. On the other hand, if g does not belong to the subdomain boundary, the token is just forwarded to the next subdomain on $R(\mathcal{I})$.

- In the case that the token is empty, it means that it is a response to a token request so the corresponding pending queue is checked for elements that still need refinement. If this check results in a point of the interface, it is being treated similarly with the main procedure above: insertion, token initialization, and token transmission.

Figure 92 depicts the method using three separate "threads". In practice the code is sequential but the use of threads simplifies the diagram.

Evaluation: To evaluate the first communication model we conducted a series of experiments on the `turing` High Performance Computing cluster at Old Dominion University[54]. In particular, we employed two nodes equipped with 2xIntel®Xeon E5-2698 v3 2.30GHz and 125Gb of memory each, for a total of 64 cores and 250Gb of memory.

We performed a weak scalability [117] study by increasing the number of elements linearly with respect to the number of processes while keeping an average load of 10 million elements per process. To avoid creating interfaces between multiple subdomains, we use a P decomposition and sorted the elements along the x-axis. Figure 93 presents the speedup and efficiency of the approach. For this dataset the number of subdomains matches the number of processes. The results indicate that this approach cannot maintain efficiency for more than a few cores.

Table 41 provides more insight into the data. The total cost of the communication for our method is quite low, less than 4% of the total running time. The time spent waiting for termination is due to the load imbalance between the processes. The imbalance does

[54]`https://wiki.hpc.odu.edu/en/Cluster/Turing` (Accessed 2021-07-01).

```
def next(i):
    return (i+1)mod number_of_subdomains_sharing_interface
def prev(i):
    return (i-1) mod number_of_subdomains_sharing_interface
struct token{
   initID /* id of subdomain that initialized the token transmission*/
   point
};
```

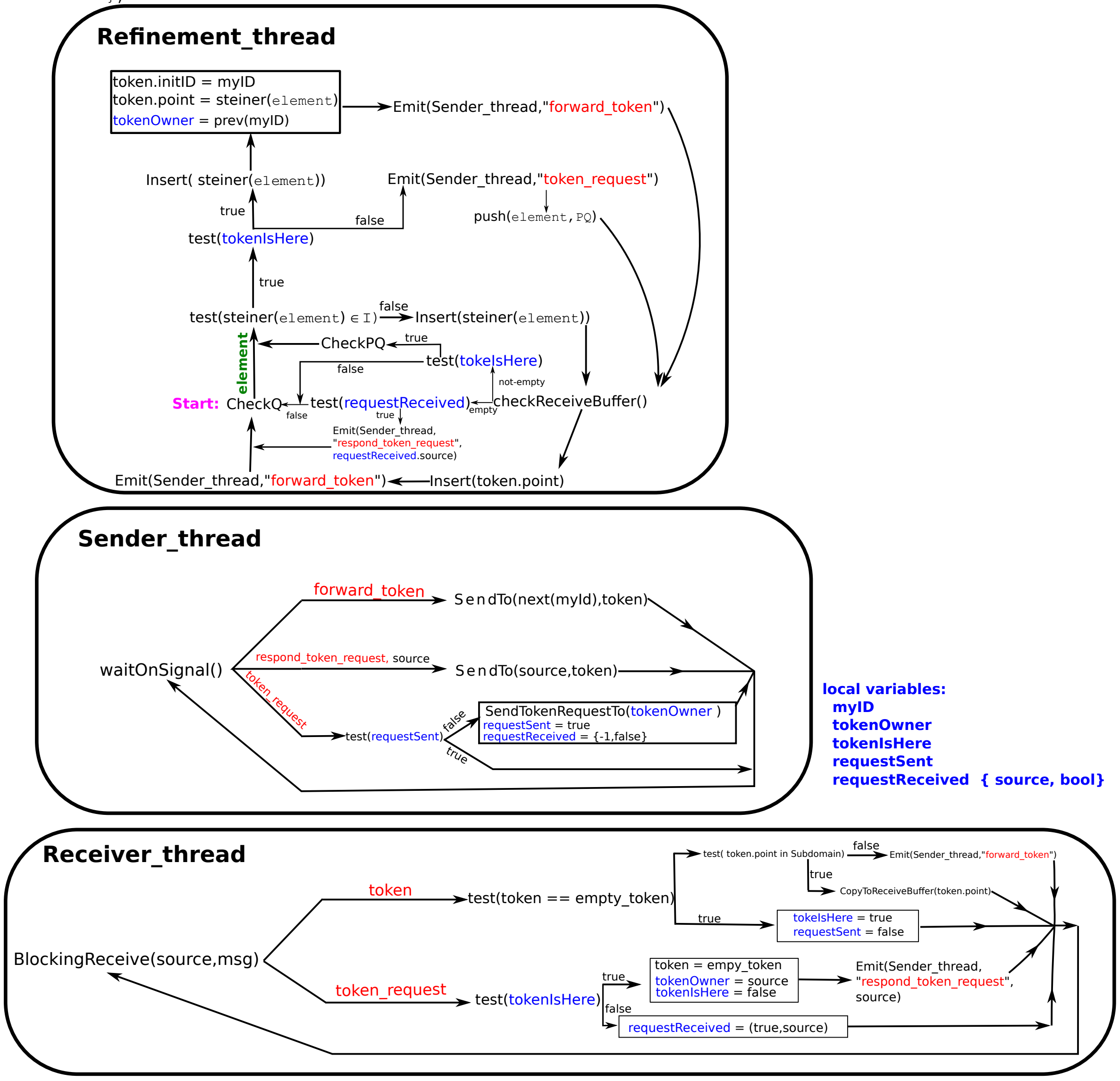
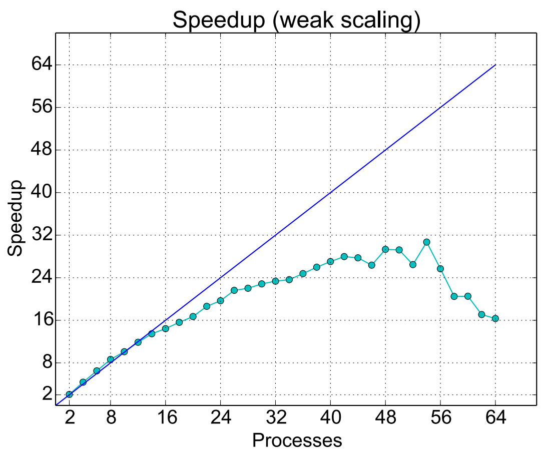


Fig. 92: Flowchart of the method. The algorithm can be visualized as three threads sharing data using the token structure, the variable `tokenOwner`, the boolean flags `tokenIsHere`, `requestSent`, `requestReceived` and the ReceiveBuffer. **Q** is the queue of bad elements while **PQ** contains (possibly deleted) elements that could not be processed due to the absence of the token.



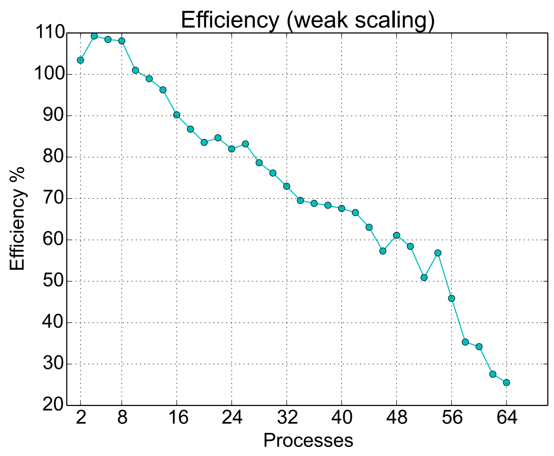


Fig. 93: Scalability results running on the double torus and generating 10 million elements per process.

Processes	**8**	**16**	**32**	**40**	**48**	**56**	**64**
Communication Overhead	2.44%	2.62%	2.94%	3.20%	3.06%	2.76%	1.61%
Handle Message	1.78%	2.72%	3.37%	3.87%	3.70%	3.24%	1.84%
Waiting to terminate	2.37%	7.55%	14.47%	12.94%	24.10%	36.23%	63.82%
In comparison to no_comm	+9.89%	+21.61%	+34.15%	+41.51%	+50.82%	+98.74%	+250.38%

TABLE 41: Overhead introduced by our method as percentage of the total running time.

not come from differences in the volume of the subdomains. The number of elements in each subdomain differs by less than 1%. The source of imbalance is that, as the number of processors grows, the ratio between volume and surface of a subdomain becomes smaller introducing thus more communication and deteriorating the performance. A solution would be to use a different decomposition such as PQ or PQR. Such a decomposition however creates interfaces between more than two subdomains and which requires more communication. For reference, using PQ across the X and Y axis instead of P decomposition increased the ruining time by more than 6 times due to the increased time spend for the tokens shared by more than two subdomains. The last row of Table 41 compares our method with a version of the code that uses no communication among the subdomains. Of course, the interfaces are not conforming anymore but, on the other hand, the lack of communication reveals the total overhead of the method. Notice that even at a low number of processes

Processes	**8**	**16**	**24**	**32**	**40**	**48**	**56**	**64**
Point Messages	28.5%	26.6%	25.5%	24.7%	24.0%	23.5%	23.3%	23.3%
Token Requests	38.5%	40.8%	42.2%	43.2%	44.0%	44.6%	45.2%	46.2%
Empty Tokens	33.0%	32.6%	32.3%	32.1%	32.0%	31.9%	31.5%	30.5%

TABLE 42: Percentage of different types of messages.

(< 16) where the efficiency of the method is high, the overhead is not negligible. Still, the method performs well due to the overlap between communication and computation that the asynchronous approach offers. Table 42 offers a breakdown of the types of messages used during the execution. Only $23 - 28\%$ of the messages contain points and the rest is the bookkeeping that the communication model performs, thus indicating the need to improve it. The negative effect of a single token among many multiple subdomains of our first approach pushed our efforts towards redefining the notion of interface and resulted in the second approach described in the next section.

B.2.3.2 Second approach: Compound Tokens

Based on the observation that the main source of inefficiency of our initial approach is the interfaces that are shared among more than two subdomains, we revised our model introducing compound tokens. This model is an extension of the previous but we repeat the common rules and definitions for completeness.
Interface is a set of (connected) faces and segments. There are two type of interfaces:

- **Segment Interfaces** Contain only segments shared by more than two subdomains.
- **Face Interfaces** Contain segments and faces that are shared by exactly two subdomains.

The rules of this model are the following:

(D_1) Only interfaces of different types can intersect.

(D_2) For each group of subdomains sharing $\mathcal{I}$, a cyclic order (ring) is defined $R(\mathcal{I})$.

(D_3) For each segment interface $\mathcal{I}$, the set of ids of the intersecting face interfaces is denoted by $\mathcal{D}(\mathcal{I})$.

(D_4) Each interface $\mathcal{I}$ is equipped with a (unique) token $t(\mathcal{I})$ and a global interface Id (*iid*).

(D_5) For each segment interface $\mathcal{I}_s$, we define the **compound** token to be the union between the segment token and the tokens of all intersecting faces i.e.,

$$\text{compound\_token}(\mathcal{I}_s) = \bigcup_{j \in \mathcal{D}(\mathcal{I}_s)} t(\mathcal{I}_j) \cup t(\mathcal{I}_s)$$

To distinguish compound from non compound tokens we refer to the latter as **proper tokens**.

(D_6) A proper token has four states:

(a) empty and unlocked

(b) empty and locked

(c) in transit together with a point (*token.point*) and the id of the subdomain that initiated the transmission (*token.initID*)

(d) in transit and empty

(D_7) Each subdomain S is equipped with five arrays:

(a) $tokenIsHere[iid]$ indicates whether the proper token for interface with global id *iid* is present.

(b) $tokenOwner[iid]$ holds the current owner (or the future owner if there is an ongoing transmission) of the proper token of the interface with global id *iid*.

(c) $tokenIsLocked[iid]$ indicates whether a proper token is locked and therefore cannot be sent as a response to a request or used for splitting a segment/face.

(d) $requestForToken[iid] = priority\_queue\{source\}$ collects the token requests received while the token was missing. Token requests that arise from compound token assemblies, have higher priority than the rest. Ties on priorities are resolved based on process ids.

(e) $requestSent[iid]$ a boolean array, indicates whether a request has been sent for the interface with id *iid*. The purpose of this array is to avoid excessive number of messages.

In the following rules the compound token is excluded unless it is explicitly stated:

(R_1) A face interface $\mathcal{I}$ with id *iid* can be modified only if $t(\mathcal{I})$ is present and not locked i.e.,

$$tokenIsHere[iid] \wedge \neg\, tokenIsLocked[iid]$$

(R_2) To construct the compound token of $\mathcal{I}$, a subdomain must first acquire and lock the segment token $t(\mathcal{I})$, and then acquire and lock one at a time and in ascending order the tokens in $\mathcal{D}(\mathcal{I})$.

(R_3) A segment interface $\mathcal{I}$ can be modified only if the *compound_token*$(\mathcal{I})$ is present.

(R_4) If a proper token of an interface with id *iid* is missing and has not been requested yet (i.e. $requestSent[iid] = false$) it will be requested from the owner as indicated by the *tokenOwner*$[iid]$ value. As soon as a request is made, *requestSent*$[iid]$ is set to true.

(R_5) After inserting a point p on a shared interface $\mathcal{I}$, the token $t(\mathcal{I})$ is sent together with p to the next subdomain on $R(\mathcal{I})$ if the next is not *token.initId*.

(R_6) When a compound token *compound_token*$(\mathcal{I})$ completes a circle around $R(\mathcal{I})$ each of its components in $\mathcal{D}(\mathcal{I})$ are sent to the subdomain with the lowest id among the neighbors of $\mathcal{I}$. At this point *compound_token*$(\mathcal{I})$ is destroyed.

(R_7) If a token of an interface with id *iid* is empty and unlocked it will be sent to the first subdomain in $requestForToken[iid]$. If the entry is empty, the token will remain idle.

The requests for assembling the segment interface could cause a deadlock whenever two segment interfaces share more than one face interface. To avoid it, we always request the tokens in a predefined order and one at a time (R_2). See for example Figure 94. If subdomains D and F begin acquiring tokens to assemble I_8 and I_9 respectively at the same time, they will be competing for I_4 and I_5. If D acquires $\{I_8, I_1, I_2, I_3, I_4\}$ and waits for I_5 and E acquires $\{I_9, I_{10}, I_7, I_6, I_5\}$ and waits for I_4 a deadlock will occur. Livelock can also occur if both D and E release and re-acquire the tokens. On the other hand, if we collect them one by one and in ascending order the first one who receives I_4 will continue and the second will wait until the missing token is available. Notice also that by (D_7) pending requests tokens have higher priority.

Finally, Figures 95-100 present a pseudocode of the proposed approach. Although, it satisfies all our requirements and it is expected to perform better than the initial approach, implementing this method may be unnecessary complex. Moreover, looking through the algorithm we see a mix of mesh-, communication- and correctness-related steps. A direct

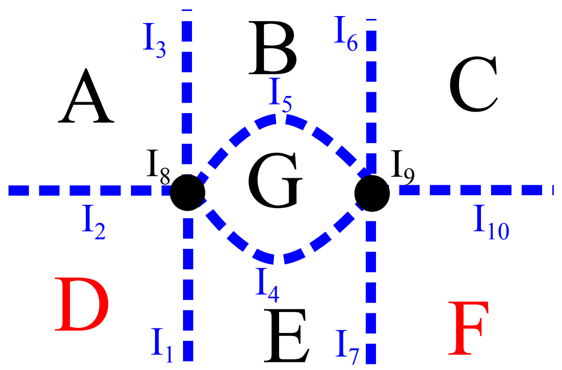


Fig. 94: A configuration of subdomains and interfaces that could lead to a deadlock.

implementation of this approach would be against the notions of separation of concerns we mentioned in Chapter 5, and it would limit significantly the code-reuse aspects of the project.

This approach could be implemented and greatly simplified if the communication scheme is approached having the capabilities of the PREMA runtime system [244] in mind. Subdomains are the obvious candidate for mobile objects [56] and together with the Implicit Load Balancing (ILB) layer [19] of PREMA they aid towards writing reusable code for packing/unpacking and load balancing the workload related to refining the subdomains. We have already demonstrated this capability in [244].

Tokens can be yet another mobile object. Since PREMA keeps track of the location of mobile objects, token requests and request forwarding can be taken care of by the runtime system, thus offloading a great complexity out of this method. Dependencies between a compound token and its constituent tokens can be achieved through Mobile object dependencies which we demonstrated in [106]. Such an approach is expected not only to speed up the implementation but will also offer better performance since PREMA offers dynamic load balancing, which is absent in this approach.

```
Function RefineSubdomain(S)
/* This is the usual Delaunay Refinement loop with the exception that
   a segment or a face may not be split due to a missing token.  In
   this case the tetrahedron that caused the need for a segment/face
   split is pushed back into the refinement queue and the face or
   segment to the appropriate pending queue in order to be processed
   as soon as the missing token arrives                              */
fi_pendingQ ← ∅ /* Queue of faces/segments on face interfaces that need
   to be split                                                       */
si_pendingQ ← ∅ /* Queue of segments on segment interfaces that need to
   be split                                                          */
counter ← 0
// Let Q be the list of tetrahedra that need refinement
while $Q \neq \emptyset$
do
    t ← Q.pop_front()
    p ← steiner (t)
    if ( p encroaches upon a segment e ) then
        split ← splitSegment(e)
        if ( split != true ) then
            Q.push_back(t)
        endif
    elif ( p encroaches upon a face f ) then
        split ← splitFace(e)
        if ( split != true ) then
            Q.push_back(t)
        endif
    else
        insert(p)
    endif
    counter++
    // control how often we check for new messages
    if ( counter > pollFrequency ) then
        if ( checkForMessage(msg) ) then
            handleMessage(msg)
        endif
        counter ← 0
    endif
end while
```

Fig. 95: Refinement procedure.

```
Function splitSegment(e)
  /* Split an interface segment only if the subdomain holds the token and the token
     is not locked.  Otherwise, a request is sent and the segment is pushed either
     into the fi_pendingQ or si_pendingQ if it lies on a face or segment interface
     respectively.                                                              */
  input: segment e
  return  true if e was split
          false otherwise
  if ( e is on face interface ) then
      iid ← getInterfaceId(e)
      // rule (R1)
      if ( tokenIsHere[iid] and not tokenIsLocked[iid] ) then
          q ← steiner(e)
          insert(q)
          sendToken(iid,e,q) // rule (R5)
          return  true
      else
          // rule (R4)
          if (  requestSent[iid] == false ) then
              sendTokenRequest(tokenOwner[iid],iid)
              requestSent[iid] ← true
          endif
          fi_pendingQ.push_back(e)
          return false
      endif
  elif ( e is on segment interface ) then
      iid ← getInterfaceId(e)
      // rule (R1)
      if (  tokenIsHere[iid] and not tokenIsLocked[iid] ) then
          /* Holding the token is not enough see rule (R3) assemble the compound token
             as rule (R4) suggests                                              */
          tokenIsLocked[iid] ← true
          all_acquired ← acquireAdjacent(iid)
          if ( all_acquired == true ) then
              /* if it happens that all the required tokens are present we can split the
                 segment                                                        */
              q ← steiner(e)
              insert(q)
              sendToken(iid,e,q) // rule (R5)
              return  true
          endif
      else
          // rule (R4)
          if (  requestSent[iid] == false ) then
              sendTokenRequest(tokenOwner[iid],iid)
              requestSent[iid] ← true
          endif
      endif
      si_pendingQ.push_back(e)
      return false
  else
      q ← steiner(e)
      insert(q)
      return  true
  endif
```

Fig. 96: Splitting a segment.

```
Function splitFace(f)
  /* Split an interface face only if the subdomain holds the token and
     the token is not locked.  Otherwise, a request is sent and the face
     is pushed into the fi_pendingQ.                                     */
  input: face f
  return  true if f was split
          false otherwise

  q ← steiner(f)
  if ( q is encroached by a segment e ) then
  |  return splitSegment(e)
  elif ( f is on a face interface ) then
  |  iid ← getInterfaceId(f)
  |  // rule (R1)
  |  if ( tokenIsHere[iid] and not tokenIsLocked[iid] ) then
  |  |  q ← steiner(f)
  |  |  insert(q)
  |  |  sendToken(iid,f,q) // rule (R5)
  |  |  return true
  |  else
  |  |  // rule (R4)
  |  |  if ( requestSent[iid] == false ) then
  |  |  |  sendTokenRequest(tokenOwner[iid],iid)
  |  |  |  requestSent[iid] ← true
  |  |  endif
  |  |  fi_pendingQ.push_back(f)
  |  |  return false
  |  endif
  else
  |  q ← steiner(f)
  |  insert(q)
  |  return  true
  endif
```

Fig. 97: Splitting a face.

```
Function handleMessage(msg)

switch msg.tag do
    case TOKEN_REQUEST do
        // a subdomain requested a token
        iid ← msg.interfaceId
        if ( tokenIsHere[iid] and not tokenIsLocked[iid] ) then
            sendEmptyToken(msg.source,iid)
        else
            // rule D7(e)
            requestForToken[iid].push(msg.source)
        endif
        break
    end case
    case EMPTY_TOKEN do
        /* this is a response to a request that we sent.  Use the token to process
           faces that were previously pushed due to missing token.  In case of a
           segment interface token we cannot use it yet                           */
        iid ← msg.interfaceId
        tokenIsHere[iid] ← true
        if ( iid is segment interface OR a compound token which requires iid is under assembly
         ) then
            tokenIsHere[iid] ← true
            tokenIsLocked[iid] ← true
            all_acquired ← acquireAdjacent(iid)
            if ( all_acquired ) then
                split the segment(s) belonging to the interface that initiated the assembly
                compose a message msg containing the new point(s), the tokens and the tag
                  POINT_ON_SEGMENT_INTERFACE
                send msg to the next on the ring of the segment token
            endif
        else //iid is a face interface
            processQ(fi_pendingQ)
            /* we can now forward the token to the subdomains that requested it
               while it was missing (rule (R7))                                   */
            if ( requestForToken[iid] != ∅ ) then
                send token to requestForToken[iid].top()
                requestForToken[iid].pop();
            endif
        endif
        break
    end case
    case POINT_ON_FACE_INTERFACE do
        /* a subdomain sent a point for insertion                                 */
        iid ← msg.interfaceId
        tokenIsHere[iid] ← true
        insertRemotePoint(msg.data)
        /* we can now forward the token to the subdomains that requested it while
           it was missing (rule (R7))                                             */
        if ( requestForToken[iid] != ∅ ) then
            send token to requestForToken[iid].top()
            requestForToken[iid].pop();
        endif
        break
    end case
    case POINT_ON_SEGMENT_INTERFACE do
        /* similar with previous case but now the token has to be advanced to the
           next subdomain on the ring                                             */
        siid ← msg.interfaceId
        tokenIsHere[siid] ← true
        if ( msg.initId == myId ) then
            /* the segment token completed a circle, so we can release the
               associated face interface tokens                                   */
            destroy the compound token and sent its components to the lowest id subdomain of
              each interface
        else
            insertRemotePoint(msg.data)
            send the token to the next on the ring
        endif
        break
    end case
end switch
```

Fig. 98: Handling an incoming message.

```
Function processQ(pendingQ)
  /* iterate pendingQ ( which can be either si_pendingQ or fi_pendingQ )
     and split all the segments & faces that can be split              */
  input: Queue pendingQ
  foreach g in pendingQ do
    if ( g does not exist in the mesh ) then
      pendingQ.remove(g)
    else
      giid ← getInterfaceId(g)
      // rule (R1)
      if ( tokenIsHere[giid] and not tokenIsLocked[giid] ) then
        if ( g is a segment ) then
          splitSegment(g)
        else
          splitFace(g)
        endif
        pendingQ.remove(g)
      endif
    endif
  end foreach
```

Fig. 99: Processing pending points.

```
Function acquireAdjacent(siid)
  /* this function implements rule (R2): assembles a compound token of
     all the face interface tokens adjacent to the segment interface
     siid. Each time it is called it requests the next face interface
     token adjacent to siid                                              */
  input: Segment Interface Id : siid
  return true if all required tokens are here
         false otherwise
  sent ← $false$
  foreach iid in sorted( intersecting face interfaces of siid) do
    if ( tokenIsHere[iid] == false ) then
      send an TOKEN_REQUEST for iid to tokenOwner[iid]
      sent ← true
      break
    else
      /* in case the token was already here we need to lock it to
         prevent both using and sending it                              */
      tokenIsLocked[iid] = true
    endif
  end foreach
  if ( sent == false ) then
    /* no messages sent, all required tokens are present                */
    unlock all required tokens
    return true
  endif
  return false
```

Fig. 100: Assembling a compound token.

VITA

Christos Tsolakis
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

e-mail: `ctsolakis@cs.odu.edu`

Christos Tsolakis received his Bachelor's degree in Mathematics from the Aristotle University of Thessaloniki, Greece in 2014. In Spring 2015, Christos joined the Computer Science Department of Old Dominion University to pursue a Ph.D. degree. He is currently working with Dr. Nikos Chrisochoides in the Center of Real-Time Computing as a research assistant. His research focuses on developing parallel mesh adaptation methods for a variety of applications including CFD (Computational Fluid Dynamics), Medical Imaging, and Nuclear Femtography data. Most of his research work involved collaborating with several research groups including NASA, NIA (National Institute of Aerospace), and the Jefferson Lab. He has (co)-authored 14 publications. His presentation at the 2019 Modeling Simulation and Visualization Student Capstone Conference won the best presentation award in the General Engineering track and was afterward invited to the 2019 MODSIM conference. He received the Modeling and Simulation Research Fellowship from the Virginia Modeling, Analysis & Simulation Center (VMASC) of ODU to support his research for 3 years (2016-2019) (maximum allowance). In 2017 he was selected to be part of the Argonne Training Program on Extreme-Scale Computing (70 participants world-wide). In 2018 he received an Outstanding Graduate Researcher award by the Computer Science Department as well as an NSF Travel award for the 27th International Meshing Roundtable and User Forum. During 2020-2021, his research was funded by the Dominion Scholar Fellowship.

Typeset using LaTeX.