

2021

Parallel Anisotropic Unstructured Grid Adaptation

Christos Tsolakis
Old Dominion University

Nikos Chrisochoides
Old Dominion University

Michael A. Park

Adrien Loseille

Todd Michal

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_fac_pubs



Part of the [Aeronautical Vehicles Commons](#), and the [Computer Sciences Commons](#)


Original Publication Citation

Tsolakis, C., Chrisochoides, N., Park, M. A., Loseille, A., & Michal, T. (2021). Parallel anisotropic unstructured grid adaptation. *AIAA Journal*, 59(11), 4764-4776. <https://doi.org/10.2514/1.J060270>

This Article is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.



Parallel Anisotropic Unstructured Grid Adaptation

Christos Tsolakis* and Nikos Chrisochoides[†]
Old Dominion University, Norfolk, Virginia 23529

Michael A. Park[‡]

NASA Langley Research Center, Hampton, Virginia 23681

Adrien Loseille[§]

National Institute for Research in Computer Science and Automation–Paris Saclay University,
91120 Palaiseau, France

and

Todd Michal[¶]

The Boeing Company, St. Louis, Missouri 63166

<https://doi.org/10.2514/1.J060270>

Computational fluid dynamics (CFD) has become critical to the design and analysis of aerospace vehicles. Parallel grid adaptation that resolves multiple scales with anisotropy is identified as one of the challenges in the CFD Vision 2030 Study to increase the capacity and capability of CFD simulation. The study also cautions that computer architectures are undergoing a radical change, and dramatic increases in algorithm concurrency will be required to exploit full performance. This paper reviews four different methods to parallel anisotropic grid adaptation. They cover both ends of the spectrum: 1) using existing state-of-the-art software optimized for a single core and modifying it for parallel platforms, and 2) designing and implementing scalable software with incomplete but rapidly maturing functionality. A brief overview for each grid adaptation system is presented in the context of a telescopic approach for multilevel concurrency. These methods employ different approaches to enable parallel execution, which provides a unique opportunity to illustrate the relative behavior of each approach. Qualitative and quantitative metric evaluations are used to draw lessons for future developments in this critical area for parallel CFD simulation.

Nomenclature

$C(\cdot)$	=	complexity of a metric field
$ k $	=	volume of a tetrahedron in evaluated metric M_{mean}
L_a	=	Euclidean edge length evaluated in the metric of vertex a
\mathcal{M}	=	continuous metric field
M	=	discrete metric field defined at the vertices of a grid
M_{mean}	=	metric tensor interpolated at the centroid of a tetrahedron
Q_k	=	mean ratio shape measure
t_{e2e}	=	total end-to-end time for grid adaptation

I. Introduction

PARALLEL anisotropic grid generation and adaptation methods modify an existing mesh to conform to a specified anisotropic metric field. This metric field is constructed to specify a new grid that reduces errors estimated on the current grid and solution. Robust grid adaptation mechanics that produce and modify anisotropic elements with aspect ratios on the order of tens of thousands are required for high-Reynolds-number viscous flows. Grid adaptation methods have made dramatic improvements in the last decade. Alauzet and Loseille [1] showed the evolution of solution-adaptive methods that include anisotropy to resolve simulations with shocks and boundary layers. Park et al. [2] documented the current state of solution-based

anisotropic grid adaptation and motivated further development for aerospace analysis and design in the broader context of the “CFD Vision 2030 Study” by Slotnick et al. [3]. The CFD Vision 2030 Study provides a number of case studies to illustrate the current state of computational fluid dynamics (CFD) capability and capacity as well as the potential impact of emerging high-performance computing (HPC) environments forecasted to be available by the year 2030.

Parallel adaptive and anisotropic grid generation is at early stages of research and development compared to parallel isotropic grid generation. In terms of concurrency, communication, and synchronization aspects, the codes for both types of grid generation share many common characteristics. Existing massively parallel isotropic grid generation and adaptation procedures for current and emerging HPC platforms often (over-)decompose the original grid generation problem into n smaller subproblems, which are solved (i.e., meshed) concurrently using $n \gg p$ cores [4]. The subproblems can be formulated to be either tightly coupled, partially coupled, weakly coupled, or decoupled. The coupling of the subproblems determines the intensity of the communication and the amount/type of synchronization required to maintain correctness and grid quality. For example, a tightly coupled approach requires each subproblem to constantly maintain consistency with adjacent subproblems. A decoupled approach decomposes the grid generation task in a way that eliminates the need for synchronization.

Four different parallel anisotropic grid adaptation methods are presented with different communication and synchronization requirements. The methods are evaluated with a number of qualitative and quantitative criteria introduced by the Unstructured Grid Adaptation Working Group (UGAWG) in their first benchmark [5], which focused on evaluating adaptive grid mechanics for analytic metric fields on planar and simple curved domains. The UGAWG is an informal group that has been formed to mature unstructured grid adaptation technology. The first UGAWG benchmark article contains a list of future directions; among them is parallel execution, which is the focus of this paper.

II. Parallel Strategies

The following parallel grid generation and adaptivity attributes are embodied to varying degrees by the software evaluated in this study.

Presented as Paper 2019-1995 at the AIAA Scitech 2019 Forum, San Diego, CA, January 7–11, 2019; received 19 October 2020; revision received 8 April 2021; accepted for publication 29 April 2021; published online 10 August 2021. Copyright © 2021 by Christos Tsolakis, Nikos Chrisochoides, Michael A. Park, Adrien Loseille, and Todd Michal. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the eISSN 1533-385X to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

*Research Assistant, Center for Real-Time Computing. Member AIAA.

[†]Richard T. Cheng Chair Professor of Computer Science, Center for Real-Time Computing. Member AIAA.

[‡]Research Scientist, Computational AeroSciences Branch. Associate Fellow AIAA.

[§]Researcher, GAMMA3 Team, Alan Turing Building. Member AIAA.

[¶]Technical Fellow. Senior Member AIAA.

They range from attributes that are crucial to success in parallel execution to attributes that ensure longevity to enhance the adaptability of software for emerging computer concurrency architectures.

1) Stability is the requirement that the quality of the grid generated in parallel must be comparable to that of a grid generated sequentially. The quality is defined in terms of the density and shape of the elements evaluated in the metric field, as well as the number of the elements (fewer is better for the same level of metric conformity).

2) Reproducibility is separated into two forms by Chrisochoides et al. [6]. *Strong reproducibility* requires that the grid generation code, when executed with the same input, produces *identical results* under the following modes of execution: 1) continuous without restarts, and 2) with restarts and reconstructions of the internal data structures. *Weak reproducibility* requires that the grid generation code, when executed with the same input, produces *results of the same quality* under the following modes of execution: 1) continuous without restarts, and 2) 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 ratio of the time taken by the best sequential implementation to the time taken by the parallel implementation. Amdahl's law [7] 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 a minimal effort. Code reuse is feasible only if the code satisfies the reproducibility criterion.

There are two common approaches for parallel grid generation and adaptation development, where these development approaches try to satisfy the aforementioned attributes. The first approach uses existing state-of-the-art serial software (i.e., fully functional, and thus maximum code reuse) and modifies it for parallel execution, which will be referred as the *functionality-first* approach. This paper briefly introduces and presents data from two such codes: *EPIC* and *Feflo.a*. The second approach designs and implements scalable software (by definition, no code reuse) with an initially incomplete functionality and the intention of completing functionality as it is needed,

which will be referred to as the *scalability-first* approach. This paper briefly introduces and presents data from two such codes: *refine* and *CDT3D*. In this paper, we focus only on stability and scalability, given that robustness and reproducibility are addressed elsewhere: for CDT3D in Ref. [8] and for the other codes in Refs. [9–11].

The grid adaptation tools used in this study leverage the parallelization methods of data decomposition, domain decomposition, or a combination of both. Chrisochoides [12] describes the telescopic approach, which applies a combination of decomposition techniques for current and emerging architectures with multiple memory/network hierarchies as shown in Fig. 1. The implementation of the telescopic approach is part of a long-term goal for parallel grid generation and adaptation at the Center for Real-Time Computing (CRTC) to achieve and sustain a billion-way concurrency over the next 12 years. To achieve 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 as well as (in the future) power consumption.

The implementation of the telescopic approach relies on multiple abstractions used in the parallel grid generation community over the last 25 years [4]: 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 intensity/type of communication/synchronization determines their mapping to different layers of memory/network hierarchy. For example, concurrency at the element or cavity level using edge swapping 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 only within the distributed shared memory of a few nodes; and asynchronous communication of data buffers is permitted over the distributed memory of several hundreds of nodes and/or tens of racks. Given these constraints, from the chip to the node levels, the telescopic approach deploys 1) parallel optimistic (PO) methods similar to those presented in Refs. [8,13,14], and 2) parallel data refinement (PDR) methods similar to those presented in Refs. [6,15]; whereas on supernodes and/or racks, it could use 3) parallel constrained (PC) methods similar to those presented in Ref. [16] and/or loosely coupled [17,18] methods.

A survey of experience with isotropic grid generation can be used to forecast the performance of future enhancements to the anisotropic algorithms. PO anisotropic grid generation codes like CDT3D on

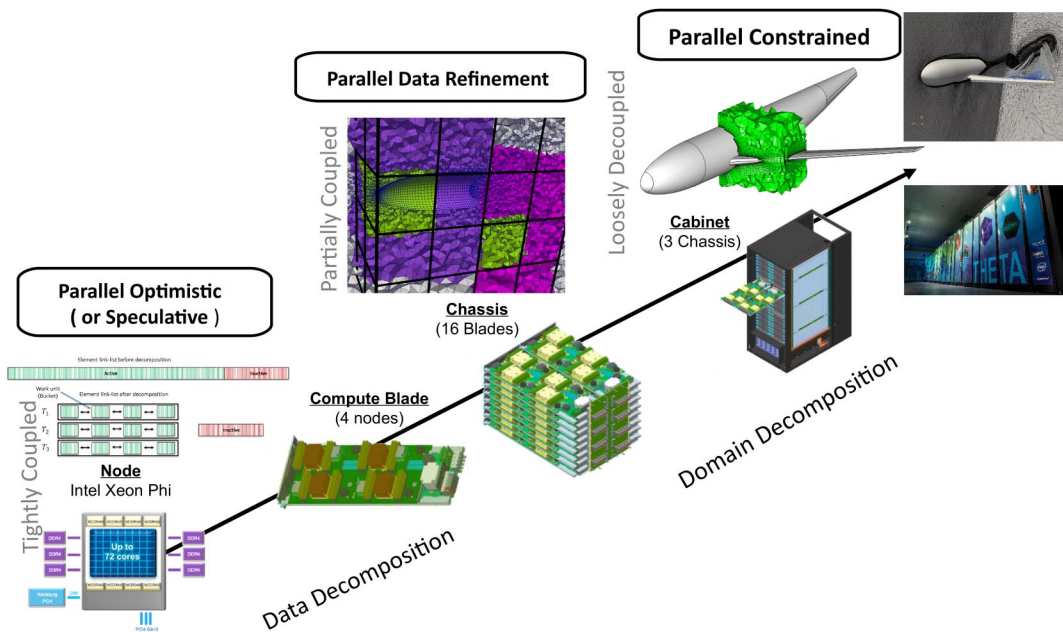


Fig. 1 The telescopic approach.

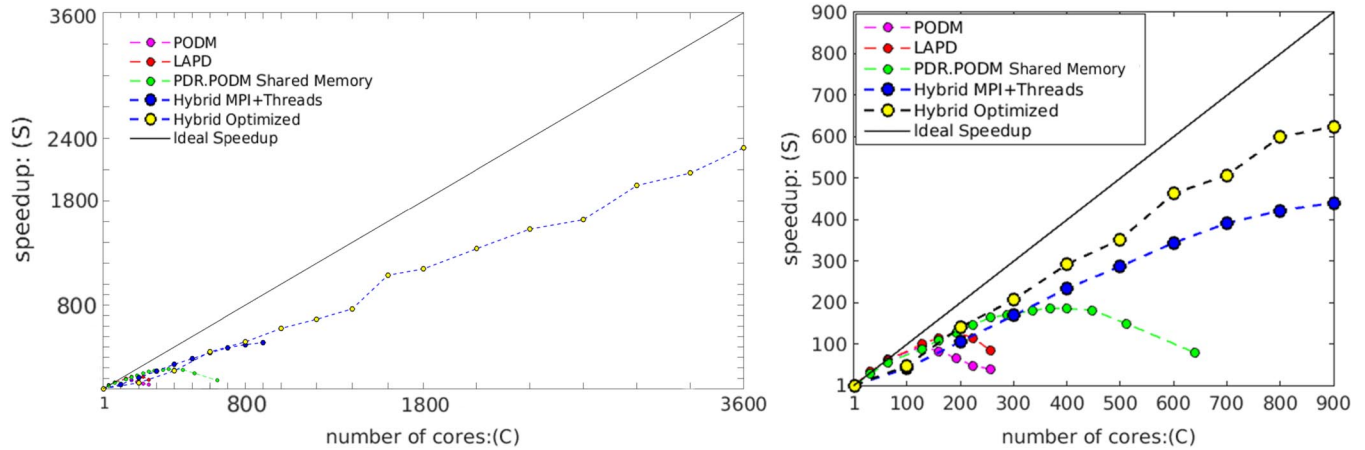


Fig. 2 Data on the first two layers of the telescopic approach applied on isotropic imaged-based grid generation.

current and emerging distributed shared-memory (DSM) machines are expected to scale up to 150 to 200 cores due to memory management issues similar to ones observed with parallel optimistic Delaunay meshing (PODM) [14]. The use of sophisticated memory pools can help to sustain scaling but do not significantly extend the practical concurrency. Locality-aware parallel implementations can help with better data affinity but have limited impact due to the dynamic memory management aspects. For example, locality aware parallel Delaunay (LAPD) [19] can improve performance for up to 200 to 250 cores; see Fig. 2 (right).

These two studies [14,19] and the data in Fig. 2 (right) suggest that one has to explore nested parallelism at both fine- and coarse-grain levels in order to improve performance for up to 900 to 1000 cores. Namely, data from isotropic grid generation implementations indicate that the application of the PDR on PODM (PDR.PODM) on both DSM [20] and distributed memory machines using hybrid Message Passing Interface (MPI + threads) programming models [21] can improve the overall performance.

However, Fig. 2 indicates that such improvements are expected to have limited impact (i.e., about 66% parallel efficiency) on higher than 3600 to 6000 cores due to local synchronization and volume of data migration. Parallel metric-based adaptive anisotropic codes are expected to have the same behavior since they are very similar to isotropic ones when it comes to concurrency, communication, and synchronization. Load balancing is a big factor in adaptive codes, but all data depicted in Fig. 2 do not use load balancing.

The CRTC team plans to address the load balancing problem using a parallel runtime software system designed and implemented for load balancing [18]. However, even with load balancing for a large number of cores ($\gg 10,000$), communication/synchronization overheads are addressed by using remaining levels of telescopic approach (i.e., PC and loosely coupled methods that rely on a lower volume of asynchronous communication). Anisotropic grid adaptation tools use one or more levels of this hierarchy as described in the following subsections.

A. CDT3D from Old Dominion University

CDT3D implements a tightly coupled approach and exploits fine-grain parallelism at the cavity level using data decomposition. Its current implementation targets shared-memory multicore nodes using multithreaded execution at the chip level. CDT3D is designed for scalability with maximum code reuse. Specifically, it is designed to simplify the implementation requirements for the communication and synchronization of both data and domain decomposition layers of the telescopic approach. This design is expected to be able to achieve high speed at the core level and tolerate costs due to gather/scatter operations in order to meet scalability requirements. The stability, robustness, and reproducibility are constantly reinforced with the parallelization of any new operation included in CDT3D.

At the chip level, CDT3D performs concurrently multiple (but same) grid operations (e.g., edge/face swapping) on different data by

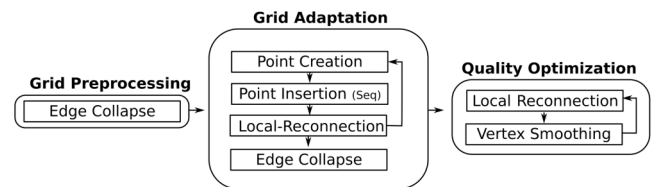


Fig. 3 The CDT3D grid generation pipeline (Seq = sequential operation).

using fast atomic lock instructions to guarantee correctness. The pipeline of CDT3D can be divided into three main steps: grid preprocessing, grid refinement, and grid quality optimization; see Fig. 3. In the first stage, edges longer than a user-defined threshold are collapsed in parallel. Grid adaptation introduces points iteratively into the grid using a centroid point creation scheme and direct insertion. After each point creation iteration, the grid is optimized in parallel using a fine-grained topological scheme for local reconnection [8], optimizing metric-based criteria. Once the refinement has been completed, a final edge-collapse pass is performed to suppress any small edges created during the refinement. In the last stage, the grid quality is further improved using a combination of local reconnection and vertex smoothing. One major improvement of CDT3D in this work is that it adapts both the boundary and the volume of the grid at the same time that, together with the introduction of the edge-collapse operation, improve the quality of the grid by three orders of magnitude in comparison to the results presented in an earlier form of this work [22]. Moreover, the advancing front *point creation* strategy that was replaced with computing the centroids of elements with at least one edge longer than the target size offers a significant improvement toward the scalability of the method.

B. EPIC from The Boeing Company

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 rebalancing 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 rebalancing 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 grid adaptation process [23] provides a modular framework for anisotropic unstructured grid adaptation that can be linked with external flow solvers. EPIC relies on repeated use of mesh operator passes to modify a grid such that element edge lengths match a given anisotropic metric tensor field. The metric field on the adapted grid is continuously interpolated from the initial metric field. Several methods are available to preprocess 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 grid and/or to the local geometry surface curvature. The surface grid is maintained on an Initial Graphics Exchange Specification (IGES) geometry definition with geometric projections and a local regriding. 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 [9,10]. Throughout this study, EPIC grids using three operations are presented: insertion, collapse and swap (EPIC-ICS).

C. refine from NASA

The refine code 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 by Park and Darmofal [24]. 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 are performed to equalize the number of nodes on each partition while penalizing elements marked for modification that span subdomains after migration. The repartitioning step provides edge weights to either ParMETIS [25] or Zoltan [26] graph-based partitioning libraries. The current load-balancing and migration approach has improved parallel scaling properties over the transcript approach described in Ref. [24].

The refine code is designed to output a unit grid [27] for a given metric field.** A combination of edge split and collapse operations proposed by Michal and Krakos [23] is used 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 [28]. Moreover, the refine code uses pliant smoothing [29], improving significantly over the results presented in Ref. [22]. Geometry is accessed through the Engineering Geometry Aerospace Design System (EGADS) [30] and EGADSlite [31] application program interface.

D. Feflo.a from National Institute for Research in Computer Science and Automation

Feflo.a employs a partially coupled coarse-grained approach that exploits parallelism at the subdomain level in a shared-memory environment. The initial grid 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 are constructed entirely of the constrained interface elements of the previous subdomains. This process recurses until all the constrained elements are adapted [32].

Feflo.a is an adaptation code developed at the National Institute for Research in Computer Science and Automation (INRIA) that can process a manifold or nonmanifold surface and/or volume grids composed of simplicial elements. It creates a unit grid [33,34] 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 grid element shape measures with node smooth-

ing as well as tetrahedra edge and face swaps. For the surface grid 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 grid and optionally on a geometry model via the EGADS application program interface.

The classical edge-based operators are implemented by a unique cavity-based operator [32,35]. This cavity-based operator simplifies code maintenance, increases the success rate of grid modifications, has a constant execution time for many different local operations, and robustly inserts boundary-layer grids [36]. When the cavity operator is combined with advancing-point techniques, it outputs metric-aligned and metric-orthogonal grids [37].

III. Experimental Evaluation

A. Method

A series of experiments are performed to evaluate the parallel strategies defined in the previous section. In each experiment, a given grid is adapted to conform to an anisotropic metric field M . Loseille and Alauzet [27] provide a thorough introduction of the definition and properties of the metric tensor field. The complexity C of a continuous metric field \mathcal{M} is defined as the integral

$$C(\mathcal{M}) = \int_{\Omega} \sqrt{\det(\mathcal{M}(x))} dx \quad (1)$$

Complexity is computed on the discrete grid by sampling \mathcal{M} at each vertex i as the discrete metric field M :

$$C(M) \equiv \sum_{i=1}^N \sqrt{\det(M_i)} V_i \quad (2)$$

where V_i is the volume of the Voronoi dual surrounding each node. The relationship between C and the number of vertices and elements in the adapted grid is shown theoretically by Ref. [27] and experimentally by Refs. [38,39]. The complexity has a linear dependency with respect to the number of vertices and tetrahedra, where the vertices are approximately $2C$ and tetrahedra are approximately $12C$.

The complexity of a metric can be scaled to create a uniformly refined (or coarsened) grid with the same relative distribution of element density and shape. The metric tensor M_{C_T} that corresponds to the target complexity C_T is evaluated by Ref. [27]:

$$M_{C_T} = \left(\frac{C_T}{C(M)} \right)^{2/3} M \quad (3)$$

where M is the metric tensor before the scaling, and $C(M)$ is the complexity of the discrete metric before scaling.

During mesh adaptation, the metric tensor values are known only on the vertices of the discrete mesh. However, the evaluation of quantities like edge length and mean ratio (see the following) require the ability to evaluate the metric at any point of the domain. This gap is filled by the use of metric interpolation. All methods presented in this work use the log-Euclidean framework for metric interpolation presented in Ref. [40]. According to the framework given a set of points $(x_i)_{i=1, \dots, n}$ and their corresponding metric values

$$M_i \equiv (M(x_i))_{i=1, \dots, n}$$

then for a point

$$x = \sum_{i=1}^n a_i \cdot x_i$$

with

$$\sum_{i=1}^n a_i = 1$$

**The refine code is available at <https://github.com/NASA/refine> under the Apache 2.0 open-source license.

the interpolated metric is defined by

$$M(x) = \exp\left(\sum_{i=1}^n a_i \ln M_i\right) \quad (4)$$

The objective of the evaluation is to support the parallel anisotropic grid adaptation method development. Two evaluation methods are used: 1) quantitative with respect to parallel performance of the codes, and 2) qualitative with respect to metric conformity of the adapted grid. The goal of metric conformity is to create a unit grid [27], where the edges are unit length and the elements are unit volume with respect to the given metric. Computing an edge length in a continuous metric field requires an integral. If assumptions are made about the interpolation of the metric between vertex a and vertex b of the grid, an analytical expression for the edge length L_e in the metric is available as [41]

$$L_e = \begin{cases} \frac{L_a - L_b}{\log(L_a/L_b)} & |L_a - L_b| > 0.001 \\ \frac{L_a + L_b}{2} & \text{otherwise} \end{cases} \quad (5)$$

$$L_a = (v_e^T M_a v_e)^{1/2}, \quad L_b = (v_e^T M_b v_e)^{1/2}$$

where v_e is the vector along edge ab . The mean ratio shape measure is also approximated in the discrete metric:

$$Q_k = \frac{36 \left(|k| \sqrt{\det(M_{\text{mean}})}\right)^{2/3}}{3^{1/3} \sum_{e \in k} v_e^T M_{\text{mean}} v_e} \quad (6)$$

where v_e is the vector along the edge e of element k , and M_{mean} is the interpolated metric tensor evaluated at the centroid of element k . The parallel performance is evaluated in terms of traditional metrics, like strong and weak speedup.

A cube with an analytically defined metric field, 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 are examined. These three simple geometries focus on the details of parallel execution without the difficulty of evaluating curved geometries. The complexities used scale up to 10,000,000, which corresponds to about 20,000,000 vertices, which is a typical size for day-to-day workflows involving analytic metrics or flows over simple straight-line geometries like the delta-wing case.^{††} Sections III.C–III.F present results on those three geometries.

B. Experimental Setup

Both the refine and CDT3D codes 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, with each one featuring two Intel® Xeon® Gold 6148 CPUs at 2.40 GHz (20 slots) and with 368 GB of memory. Feflo.a data for Secs. III.C and III.D were collected on a dual socket machine equipped with two Intel Xeon E5-2697 v2 CPUs at 2.70 GHz (12 slots); whereas for Secs. III.E and III.F on a dual socket machine with two Intel Xeon E5-2680 v2 CPUs at 2.80 GHz (20 slots). The execution times and hardware specifications are omitted for the EPIC results to protect proprietary data. This might seem contrary to open discussions in forums like this, but the authors feel that this real-life constraint does not affect the lessons learned. In fact, the value of including the normalized scaling and metric conformity evaluations of an industrial code exceeds the minor limitation of an incomplete comparison.

Before delving into the details of the experimental results, it should be noted that, currently, there is no formally proven optimal grid adaptation method. All presented methods (as most of the anisotropic

metric-based grid adaptation methods) use heuristics built around basic grid operations like point insertion, edge collapse, edge-face swaps, and vertex smoothing. A study of the effect of different scheduling policies for these grid operations on the metric conformity appears in Ref. [42] and in more detail in Ref. [43]. Each of the presented methods has different tradeoffs in speed, quality, and robustness that stem out of the different foci of each group, as has been seen in Secs. II.A–II.D.

C. Cube

The first geometry is a cube with an analytically defined metric field M referred to as polar-2 in the first UGAWG benchmark [5], where a cube–cylinder geometry was specified. Here, a unit cube is used; see Fig. 4. The metric is defined as

$$M = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_r^{-2} & 0 & 0 \\ 0 & h_t^{-2} & 0 \\ 0 & 0 & h_z^{-2} \end{bmatrix} \begin{bmatrix} \cos(t) & \sin(t) & 0 \\ -\sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

where $r = \sqrt{x^2 + y^2}$, $t = \text{atan2}(y, x)$, $h_z = 0.1$, $h_0 = 0.001$, and $h_r = h_0 + 2(0.1 - h_0)|r - 0.5|$. The subscript t is in the θ direction, and subscript r is the radial direction. The spacing in the tangential direction is defined by

$$d = 10(0.6 - r) \quad \text{and} \quad h_t = \begin{cases} 0.1 & \text{if } d < 0 \\ d/40 + 0.1(1 - d) & \text{if } d \geq 0 \end{cases} \quad (8)$$

This metric field represents a curved shear layer. This polar distribution has low gradation and is possible to satisfy with high-quality elements by resolving curvature in the tangential direction near the layer. Its complexity as defined by Eq. (2) is evaluated on the grid vertices and then scaled using formula (3) in order to match the target number.

The initial grid conforms to the polar-2 metric with a complexity of 7600. The polar-2 metric field is scaled to a 500,000 complexity for this test. Adapted grids with approximately 1,000,000 vertices and 6,000,000 tetrahedra are expected, which is a relatively small example based on the size of a typical fluid simulation. This small 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 the refine, CDT3D, and EPIC codes to adapt the initial 7600 complexity grids to conform to the 500,000 complexity as a function of the number of cores is shown in Fig. 5. 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

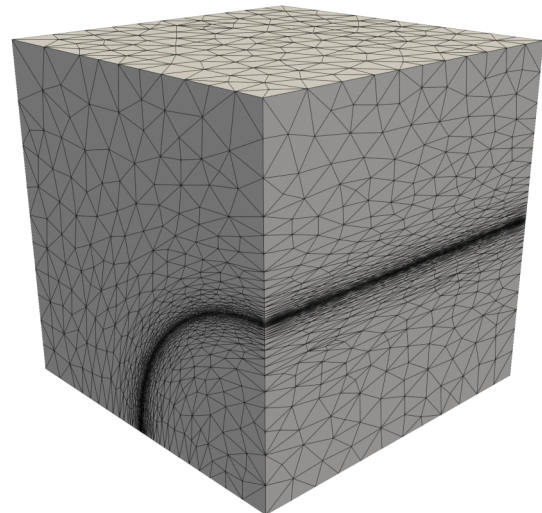


Fig. 4 Cube with polar-2 analytic metric and complexity of 7600.

^{††}Materials for these three cases are available at <https://github.com/UGAWG> [retrieved 1 September 2020].

^{‡‡}<https://wiki.hpc.odu.edu/en/Cluster/Wahab> [retrieved 5 July 2021].

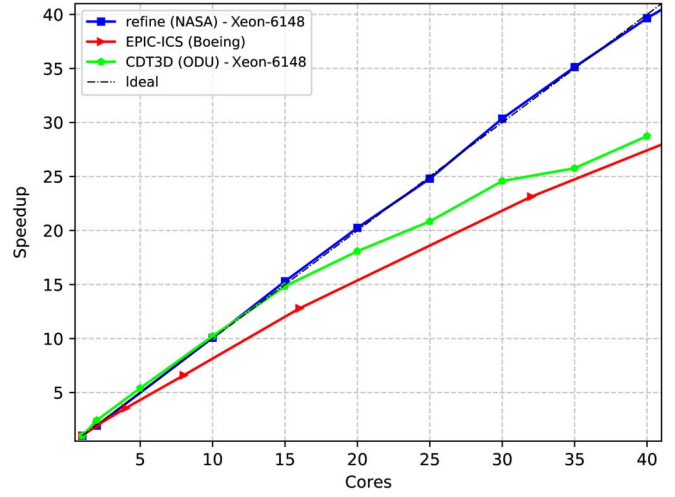
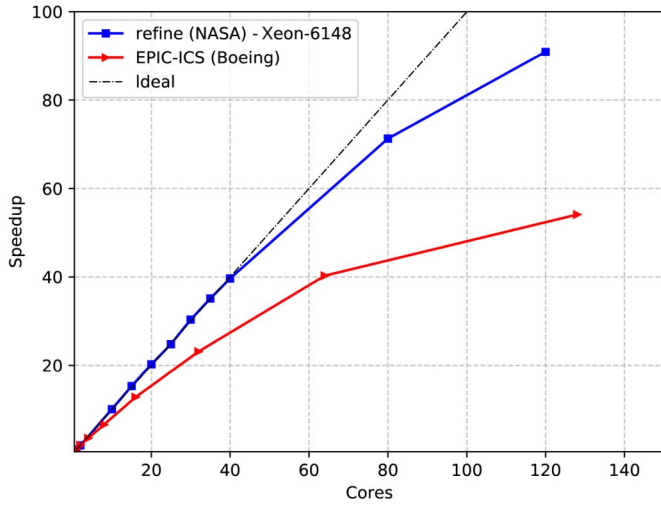


Fig. 5 Speedup for the cube case adapted from 7600 to 500,000 complexity (left), and zoom-in view of the data for up to 40 cores (right). (Base case is the sequential time of each software.) ODU denotes Old Dominion University.

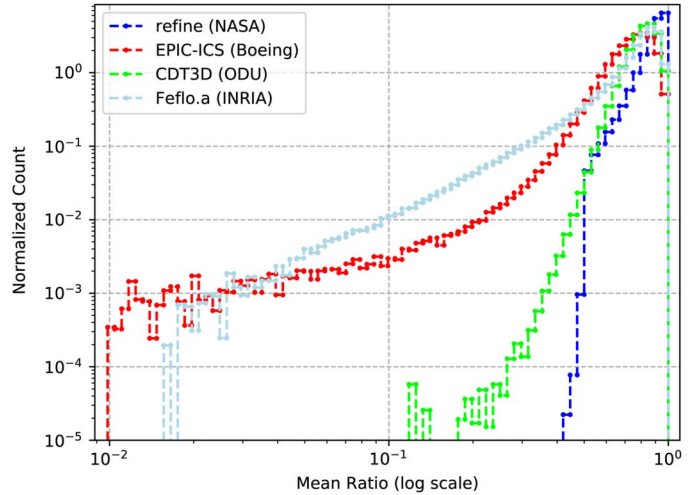
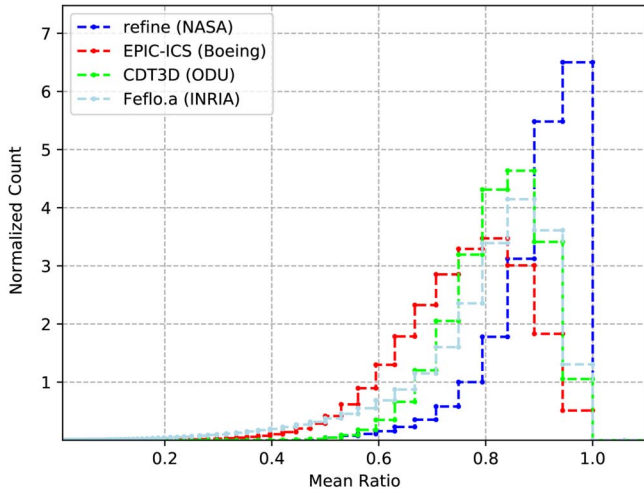


Fig. 6 Comparison of the mean ratio of the generated grids for the cube case in linear and logarithmic scales.

appear in Sec. III.F. All three methods exhibit linear scaling in a low number of cores. At higher core numbers, the speedup becomes constant for both EPIC and refine codes.

Metric conformity, characterized by element shape measure and edge length histograms of the generated grids, is shown in Figs. 6

and 7. 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 the refine code generating the highest number of elements in the range [0.8, 1.0]. The log scale makes the differences

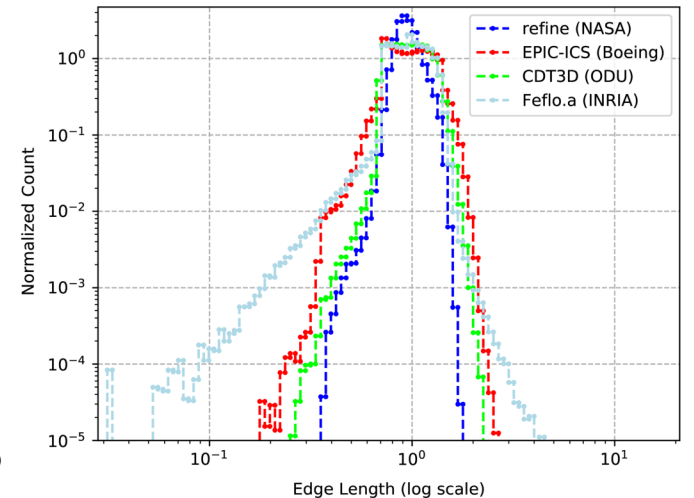
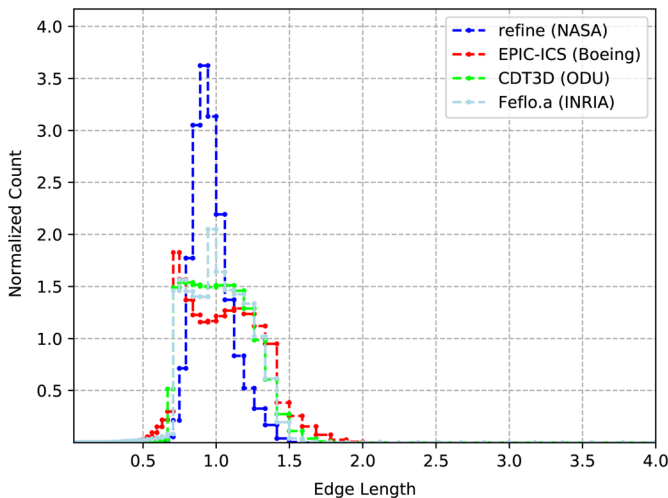


Fig. 7 Comparison of the edge lengths of the generated grids for the cube case in linear and logarithmic scales.

more prevalent. The refine code produces elements with the highest minimum mean ratio of 0.4, CDT3D has the second-best quality result, whereas the lowest mean ratio is around 0.01 for EPIC and Feflo.a. The ideal edge length distribution is clustered tightly around unity. Figure 7 (left) reveals that the refine, CDT3D, and EPIC codes generated edges with less variance, whereas Feflo.a produced both the shortest edges and the largest edges.

D. Delta Wing

The second geometry (Fig. 8) is a delta wing constructed of planar facets. A multiscale metric [44] is constructed based on the Mach field of this subsonic laminar flow. The initial grid is adapted to a specified complexity of 50,000, and details of the verification of the delta wing/grid adaptation process are provided by Park et al. [45]. The multiscale metric is scaled to have a complexity of 500,000 for the input to the adaptation evaluation. Adapted grids with approximately 1,000,000 vertices and 6,000,000 tetrahedra are expected. In the second set of data, the complexity is scaled to 10,000,000, which produces approximately 20,000,000 vertices; these grid sizes are close to the maximum of the verification study performed in Ref. [45]. Exploring more complicated flows over complex geometries may require higher complexities, and thus larger grids, but this is outside the scope of this paper.

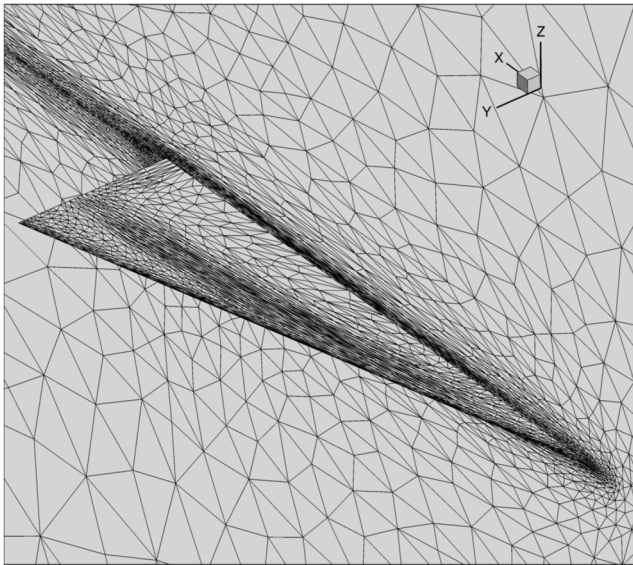
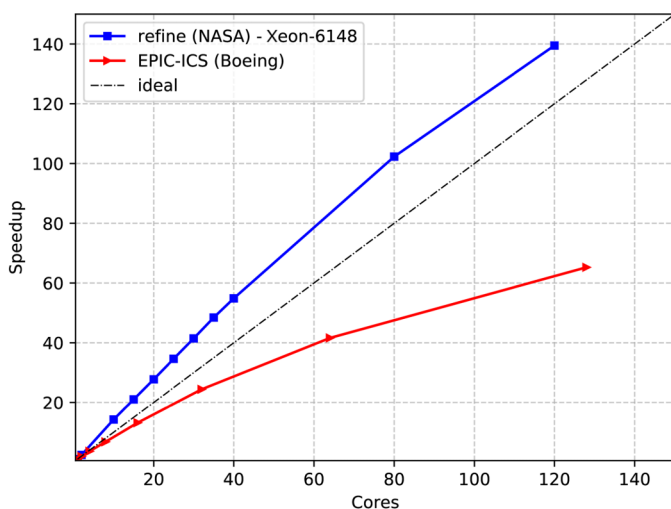


Fig. 8 Delta wing with multiscale metric in laminar flow: 50,000 complexity.



1. Strong Scaling and Quality Data

The initial grid conforms to the metric with a complexity of 50,000. The speedup of the refine, CDT3D, and EPIC codes when adapting the initial 50,000 complexity grid to conform to a 500,000 complexity metric field as a function of the number of cores is shown in Fig. 9. Similar to the previous case, Feflo.a results are omitted for the lower complexity case. At high core numbers, both EPIC and refine codes exhibit improved scaling over the performance of the cube case due to the larger size of the initial grid for the delta wing. At lower core counts, the refine code exhibits the best scaling, whereas CDT3D falls between the EPIC and refine codes. The superlinear scaling of the refine code is a result of the fact that the refine code has optimizations like reordering of the nodes for cache efficiency within each partition, which have a computational complexity higher than $\mathcal{O}(n)$, 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 the refine code 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, the refine code 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 the refine code for one core is 12,604 s and for 120 cores is 90 s, whereas on a node of the same cluster, CDT3D requires 794 s for one core and 29 s for 40.

When the complexity of the target grid is scaled to 10,000,000, EPIC retains the same scalability with the previous case as shown in Fig. 10. CDT3D exhibits minor superlinear speedup up until 30 cores and linear between 30 and 40. The origin of the superlinear speedup could be attributed to the increased throughput achieved by using the cache memory shared among the hardware threads. Feflo.a’s scaling becomes constant at eight cores, which is a result of the high startup cost of the decomposition method. More details for this cost appear in Sec. III.F. The refine code’s results are omitted from the graphs because they exhibit the same issue as before, with the sequential performance skewing the results to highly superlinear trends.

Returning to the 500,000 complexity target metric, metric conformity (characterized by element shape measure and edge length histograms of the generated grids) is shown in Figs. 11 and 12, respectively. On a linear scale, all methods appear to exhibit good overall quality. The log scale makes the differences more prevalent. The refine code’s grid quality exhibits the best lower bound in the mean ratio measure and the distribution with the smallest deviation in the edge length measure.

2. Stability

The concepts of stability and reproducibility were introduced in Sec. II. Adherence to these attributes is measured by evaluating the

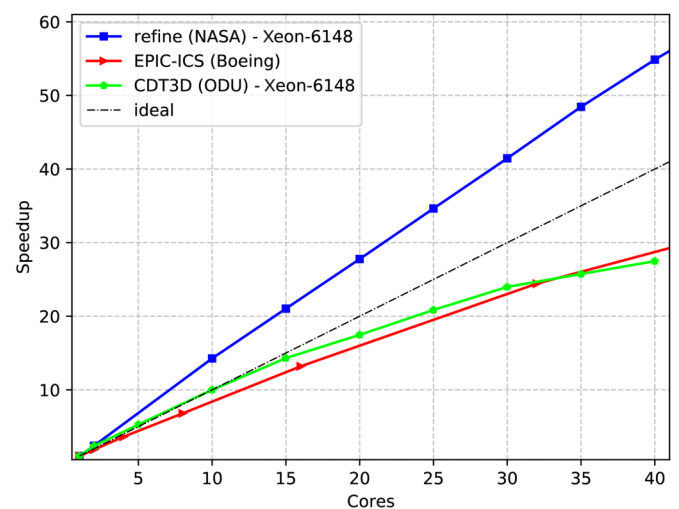


Fig. 9 Speedup data for delta wing adapted from 50,000 to 500,000 complexity (left) and zoom-in view of the data for up to 40 cores (right). (Base case is the sequential time of each software.)

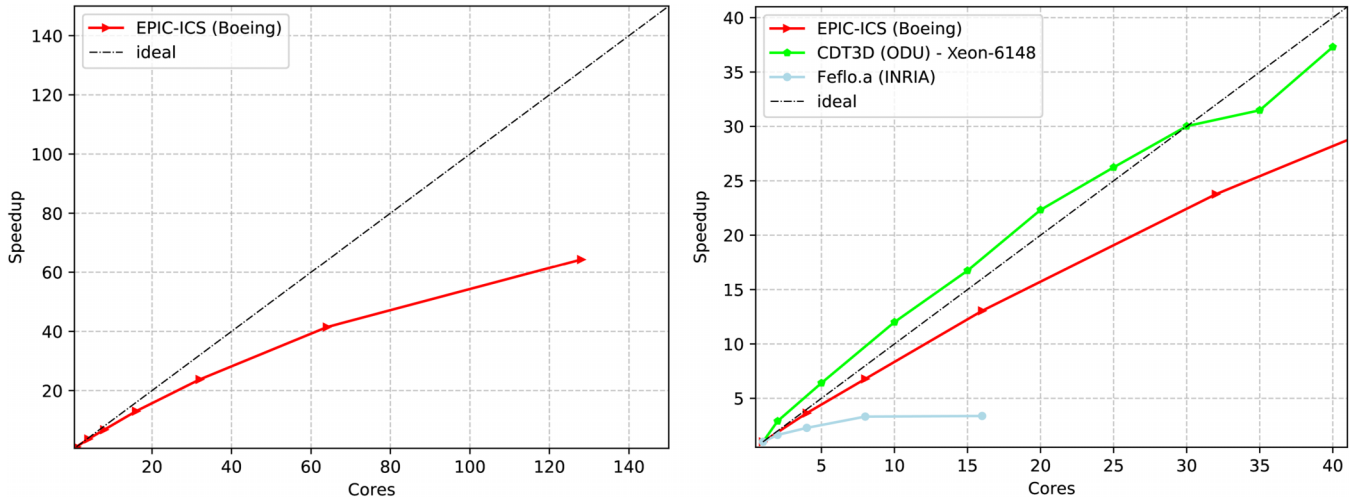


Fig. 10 Speedup data for the delta wing adapted from 50,000 to 10,000,000 complexity (left) and zoom-in view of the data for up to 40 cores (right).

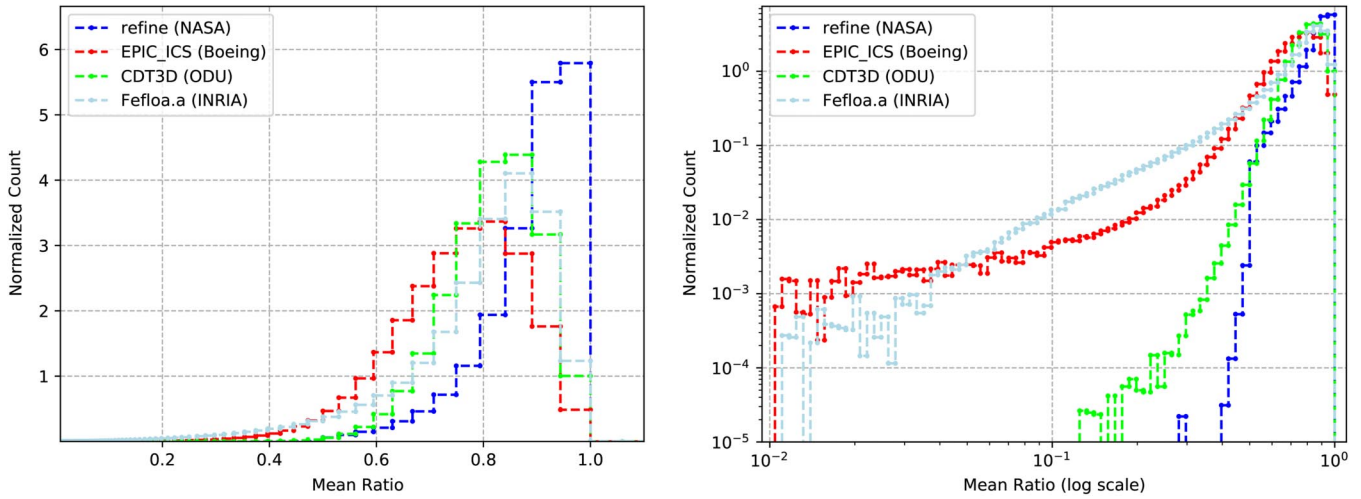


Fig. 11 Comparison of the mean ratio of the generated grids for the delta wing 500,000 complexity case in linear and logarithmic scales.

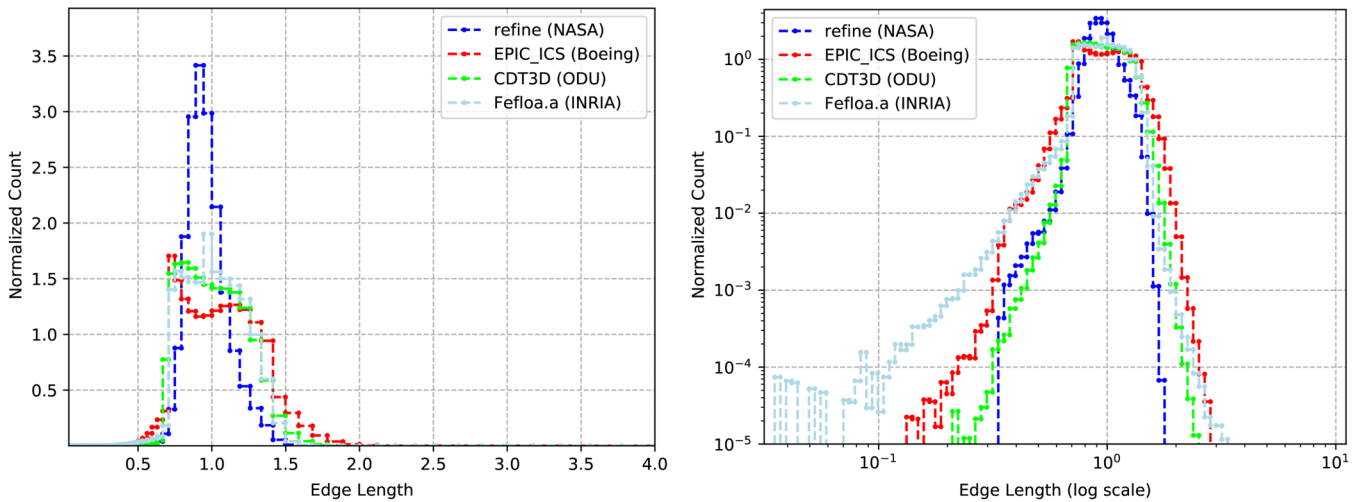


Fig. 12 Comparison of the edge lengths of the generated grids for the delta wing 500,000 complexity case in linear and logarithmic scales.

metric conformity of the same case with different numbers of cores. Histograms of the edge length in the metric are evaluated for three codes for execution with different numbers of cores in Fig. 13. The refine, CDT3D, and EPIC codes show an almost perfect overlap of the histograms, but they do not produce the same grid (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.

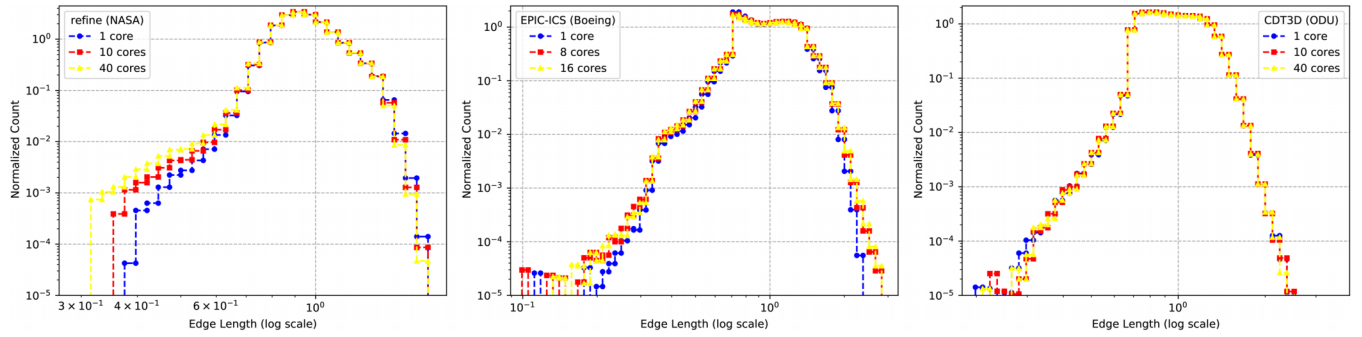


Fig. 13 Stability data for the delta wing 50,000 to 500,000 complexity case using refine, EPIC, and CDT3D codes.

E. Spherical Blast

To complement the previous two cases where grid 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 [46]. The target metric complexity is 49,013, which corresponds to about 98,000 vertices in the final grid. As initial input, a uniform tetrahedral grid of 1,900,000 vertices is provided. The adapted grid is shown in Fig. 14.

For the refine code, 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 skewing, and thus the results. CDT3D was configured with a higher collapse limit for the grid preprocessing step (see Fig. 3). This configuration was selected because it gives more flexibility in

the subsequent refining step and yields better quality in the final grid. A similar approach is used in Ref. [37] for generating an *almost empty*, grid and subsequently a metric-orthogonal grid.

Figure 15 depicts the strong scaling performance of Feflo.a, CDT3D, and refine codes. The refine code exhibits superlinear scalability for a low number of cores (less than 80) and is 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 1 presents the total time for this experiment in a shared-memory setting (40 cores); for the refine code, we include distributed memory results (up to 400 cores). A direct comparison of the times is not possible because, as is mentioned in Sec. III.B, the results of refine and

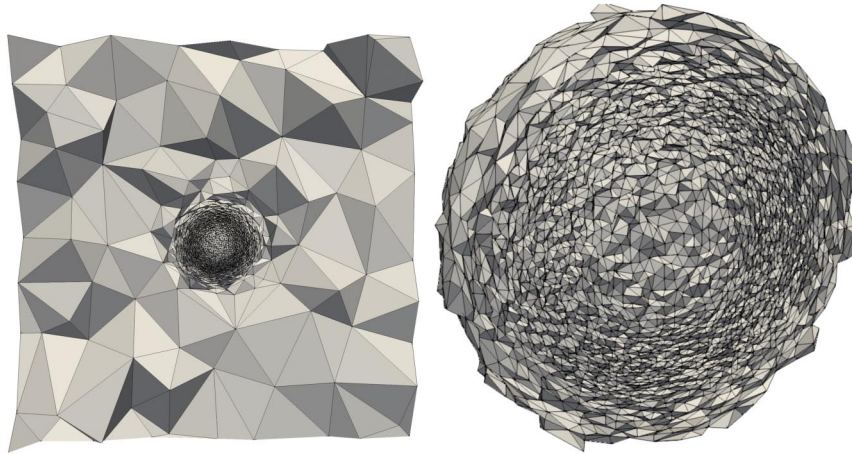


Fig. 14 Adapted grid of the spherical blast case: crosscut of the domain (left) and zoom-in of the extracted part of the core (right).

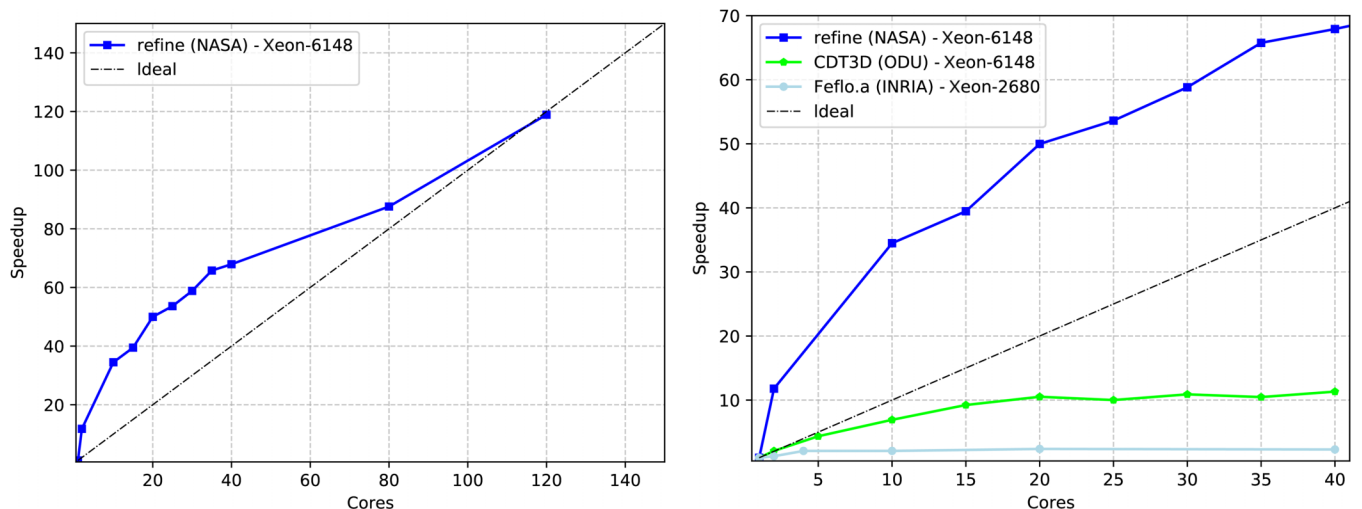


Fig. 15 Speedup data for the blast case (left) and zoom-in view of the data for up to 40 cores (right). (Base case is the sequential time of each software.)

Table 1 Total running times of Feflo.a, CDT3D, and refine codes for the blast case

No. of cores	Feflo.a, s	CDT3D, s	refine, s
1	62.82	152.41	62,574.51
2	50.57	73.76	5,311.00
10	30.41	22.02	1,814.63
20	26.45	14.49	1,252.03
40	27.42	13.46	921.39
200	---	---	332.17
400	---	---	190.36

Table 2 Breakdown of total running time for Feflo.a

No. of cores	Total time	Subdomain creation	Grid 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

CDT3D codes were collected on the same machine; whereas for Feflo.a, a different machine was used. Still, the table reveals that Feflo.a is faster than CDT3D and refine codes using one core. However, using more than 10 cores, CDT3D is 50% faster; and on 40 cores, it is more than twice faster Feflo.a. On the other hand, the

refine code achieves a speedup of 67 on 40 cores and 328 on 400 cores; the superlinear speedup occurs due to the reasons discussed in Sec. III.D.1. The breakdown of the running time of Feflo.a in Table 2 reveals that the main inefficiency is the subdomain creation step, which takes a constant amount of time for all five runs. Moreover, Feflo.a uses a *cavity-based* collapse operation [35], which always results in an edge collapse; whereas the standard collapse algorithm used by CDT3D and refine codes rejects a fair amount of configurations, which revisits in a subsequent step.

Figures 16 and 17 depict the quality of the grids generated by the refine, CDT3D and Feflo.a codes. The quality of the generated grids is in accordance with the results of the cases discussed earlier. The refine code 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 two and four, as well as a small number of edges below 0.1. For the mean ratio, the refine code delivers a grid with a minimum mean ratio quality of 0.3; for Feflo.a, the minimum is 0.2; whereas for CDT3D, it is 0.1.

F. 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 using weak scaling, also known as *scaled speedup*, for evaluating the performance of a parallel grid generation code by increasing the size

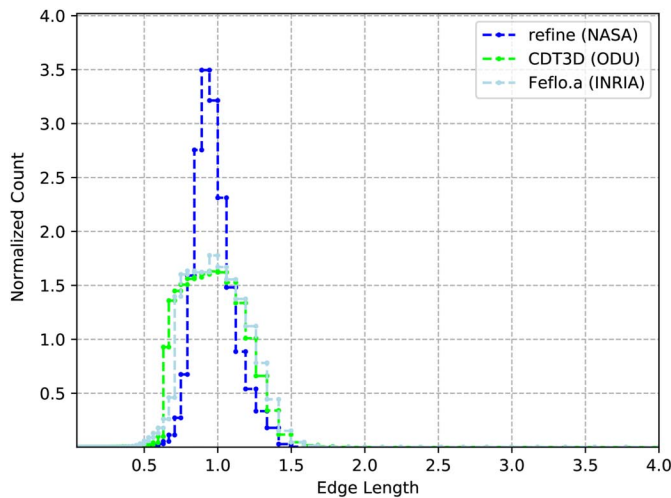


Fig. 16 Comparison of the edge lengths of the generated grids for the spherical blast case in linear and logarithmic scales.

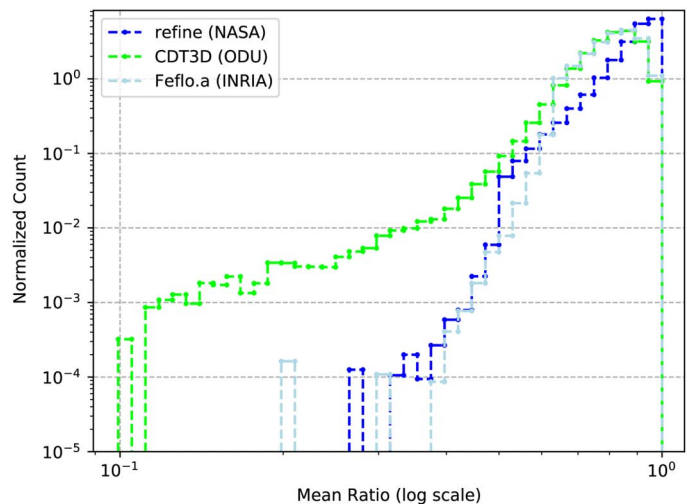
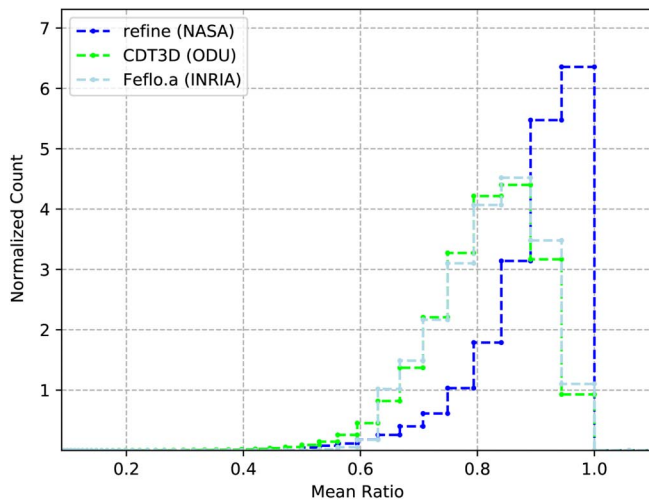


Fig. 17 Comparison of the mean ratio of the generated grids for the spherical blast case in linear and logarithmic scales.

Table 3 Weak scaling performance of refine, CDT3D, and Feflo.a codes for complexities between 50,000 and 20,000,000

Cores	Complexity	refine		CDT3D		Feflo.a	
		No. of vertices	t_{e2e}	No. of vertices	t_{e2e}	No. of 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

“k” denotes thousands and “m” denotes million.

of the grid linearly to the number of cores; see, for example, Ref. [16]. However, this approach does not reflect the workflow of a simulation using 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 grid. 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” [47].

In this work, we define the problem size to be the complexity of the target metric rather than the number of elements in the grid. Moreover, we do not use a constant step for increasing the complexity since it is common for metric adaptive simulations to use a bigger step for the first iteration [45].

Performing a fully adaptive simulation is out of the scope of this work. Instead, we simulate each solver → error-estimation → metric-construction step by artificially scaling up the complexity of the input metric. In particular, the starting grid and metric are the same as in Sec. III.D, and they are the input to the first “iteration” using one core. The input of the second iteration is created by increasing the complexity of the output grid of the previous step by a constant amount using formula (3) at every vertex of the grid. The same procedure was applied for the rest of the steps. Table 3 presents the results.

The refine and CDT3D codes retain almost constant times of approximately 10,000 and 1000 s, respectively, as the problem size increases, which indicates a good weak scaling speedup. Feflo.a is the fastest among the methods, even when considering the difference between the machines that were tested. On the other hand, it does not scale linearly as the size of the problem increases. Similar to the previous case, the overhead of domain decomposition and distribution is a considerable amount for Feflo.a, scaling from 6 s at 2 cores to 108 s at 40 cores, which corresponds to 30% of the total running time.

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

IV. Future Work

Designing and implementing scalable software from the ground up leads to short-term incomplete but rapidly maturing functionality. Evidence from this group’s experience suggests that scalability-first methods like CDT3D with proper design decisions can accelerate efforts to extend functionality [48] and improve element quality. Work remains for both approaches, but sharing experiences from very targeted efforts like this paper will aid all parties. For example, scalability-first methods like CDT3D can improve conformity of the metric by targeting and prioritizing areas of interest suggested by functionality-first software like EPIC, which has been optimized to meet industrial needs. Functionality-first methods like EPIC could benefit by using a tightly coupled and telescopic approach adopted by CDT3D to improve scalability on current and emerging hardware.

The pluralism in the different methods and their implementation (even when they belong to the classification presented in Ref. [4]) is a mutual benefit to this community. The main contribution of the lessons learned in this paper is to identify very specific improvements

for both functionality-first and scalability-first methods in a labor-efficient way. Given that grid adaptation (and, specifically, parallel grid adaptation) is a labor-intensive task, our hope is that this study (and follow-up studies) will provide insight to meet the challenges stated in the CFD Vision 2030 Study.

In the near future, a follow-up study could extend the presented results by evaluating the four codes on a series of cases that include complex geometrical models at high complexity numbers that will justify larger grid sizes. This type of case will contribute significantly toward studying the robustness of each approach in real-world configurations.

Given the trends of the upcoming exascale machines that include a massive number of GPU cores^{§§} and others focused on power efficiency and different processor architectures,^{¶¶} more research is needed toward this direction. Although Graphics Processing Unit (GPU)-capable mesh adaptation efforts like Ref. [49] exist, these methods were built from the ground up, having in mind the efficiency on GPU architectures that favors structured data structures and regular memory accesses. Moreover, one of the main difficulties is to address dynamic data structure modifications on GPUs. Dynamic data structure modifications are required by operations like edge flips that modify the grid topology. On the other hand, smoothing GPU kernels are easier to implement since they only modify vertex coordinates. For methods like CDT3D that use heavily pointer-based data structures and exhibit highly irregular memory access, a direct transition to GPU would be more challenging. However, a combination with Old Dominion University’s runtime system [18] may enable the use of GPU kernels as “plug-ins” for specific parts of the code that exhibit high floating-point operation intensity. Preliminary results of CDT3D using the runtime system appear^{***}. On the other hand, Feflo.a and refine codes that use mainly array-based structures may be more suitable for this kind of transition. In either case, more research is needed toward identifying an efficient way (both in terms of parallel efficiency and code reuse) for enabling GPU usage for the presented codes.

The unique mix of the high number of floating-point operations with frequent and irregular memory access could be profiled for power consumption. Based on our previous experience with Delaunay-based methods [50], the gains of optimizing for power diminish once we move to a newer CPU model. However, the pluralism of the different approaches here may exhibit different behavior, especially on the latest generations of processors.

The effort started by the UGAWG has already returned value to the participants and wider grid adaptation community. The general consensus of the UGAWG is that parallel anisotropic grid adaptation codes could improve their scalability by exploring concurrency at several nested levels of abstractions like the telescopic approach depicted in Fig. 1 for isotropic methods.

^{§§}Data available online at <https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/> [retrieved 8 March 2021].

^{¶¶}Data available online at <https://www.top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer> [retrieved 8 March 2021].

^{***}Tsolakis, C., Thomadakis, P., and Chrisochoides, N., “Exascale-Era Parallel Adaptive Mesh Generation and Runtime Software System Activities at the Center for Real-Time Computing,” (presentation), Oct. 2020, <https://epcced.github.io/ELEMENT/workshops.html> [retrieved 8 March 2021].

V. Conclusions

This paper presents four parallel anisotropic grid generation and adaptation methods from both ends of the spectrum for parallel grid adaptation: functionality first (i.e., EPIC and Feflo.a) and scalability first (i.e., refine and CDT3D). In the rest of this section, the lessons learned with respect to the five criteria defined in Sec. II are summarized. The experimental data from EPIC, refine, Feflo.a, and CDT3D codes suggest the following:

A. Stability

For the target geometries, all four codes exhibit stability as depicted throughout the quality plots and in Fig. 13. These codes are tested in a large set of geometries independently and experience the same behavior in terms of their stability.

B. Reproducibility

There is high cost for delivering strong reproducibility, but weak reproducibility can be attained at a lower cost. Weak reproducibility is sufficient for most flow solvers and adaptive grid processes.

C. Robustness

No special effort is made to test robustness. However, independent of this study, there is evidence [8–11] that these codes are robust, which is not a trivial task, especially for the methods that rely on discrete domain decomposition. Unexpected artifacts on the surfaces of discrete domain decomposition can disrupt boundary recovery.

D. Scalability

The scalability results on shared-memory nodes with a lower number of cores are encouraging. Strong speedup data from the EPIC and refine codes suggest high-end user productivity. Weak scaling speedup data for a low number of cores (Table 3) suggest similar end-user productivity and promising scalability at a higher number of cores.

E. Code Reuse

By design, all four codes leverage code reuse at different levels. For example, EPIC and Feflo.a rely on existing sequential fine-tuned highly optimized, fully functional code. The current version of the refine code is structured to reuse low-level data structures based on experience and code from an earlier version with lower scalability potential. CDT3D is designed from the ground up to meet all the requirements for each of the layers of the telescopic approach and is expected to accomplish this with more than 95% code reuse, which is a lower bound from CRTC's experience with TetGen [6] and PODM [20].

Acknowledgments

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, National Science Foundation (NSF) grant no. CCF-1439079, the Richard T. Cheng Endowment, the Modeling and Simulation fellowship of Old Dominion University, and the Dominion Scholar fellowship of Old Dominion University. Experiments were supported, in part, by the Research Computing clusters at Old Dominion University. Cameron Druyor and Kyle Anderson provided feedback that improved this paper. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Government.

References

- [1] Alauzet, F., and Loseille, A., "A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics," *Computer-Aided Design*, Vol. 72, March 2016, pp. 13–39. <https://doi.org/10.1016/j.cad.2015.09.005>
- [2] Park, M. A., Krakos, J. A., Michal, T., Loseille, A., and Alonso, J. J., "Unstructured Grid Adaptation: Status, Potential Impacts, and Recommended Investments Toward CFD Vision 2030," AIAA Paper 2016-

- 3323, 2016. <https://doi.org/10.2514/6.2016-3323>
- [3] Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerodynamics," NASA CR-2014-218178, March 2014. <https://doi.org/2060/20140003093>
- [4] Chrisochoides, N., "Parallel Mesh Generation," *Numerical Solution of Partial Differential Equations on Parallel Computers*, Vol. 51, Lecture Notes in Computational Science and Engineering, Springer-Verlag, New York, 2006, pp. 237–264. https://doi.org/10.1007/3-540-31619-1_7
- [5] Ibanez, D., Barral, N., Krakos, J., Loseille, A., Michal, T., and Park, M., "First Benchmark of the Unstructured Grid Adaptation Working Group," *Procedia Engineering*, Vol. 203, Jan. 2017, pp. 154–166. <https://doi.org/10.1016/j.proeng.2017.09.800>
- [6] Chrisochoides, N., Chernikov, A., Kennedy, T., Tsolakis, C., and Garner, K., "Parallel Data Refinement Layer of a Telescopic Approach for Extreme-Scale Parallel Mesh Generation for CFD Applications," AIAA Paper 2018-2887, 2018.
- [7] Amdahl, G. M., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference*, ACM, New York, 1967, pp. 483–485. <https://doi.org/10.1145/1465482.1465560>
- [8] Drakopoulos, F., Tsolakis, C., and Chrisochoides, N. P., "Fine-Grained Speculative Topological Transformation Scheme for Local Reconnection Methods," *AIAA Journal*, Vol. 57, No. 9, 2019, pp. 4007–4018. <https://doi.org/10.2514/1.J057657>
- [9] Michal, T., Kamenetskiy, D., and Krakos, J., "Anisotropic Adaptive Mesh Results for the Third High Lift Prediction Workshop (HiLiftPW-3)," AIAA Paper 2018-1257, 2018. <https://doi.org/10.2514/6.2018-1257>
- [10] Michal, T. R., Kamenetskiy, D., Krakos, J., Mani, M., Glasby, R., Erwin, T., and Stefanski, D. L., "Comparison of Fixed and Adaptive Unstructured Grid Results for Drag Prediction Workshop 6," *Journal of Aircraft*, Vol. 55, No. 4, 2018, pp. 1420–1432. <https://doi.org/10.2514/1.C034491>
- [11] Park, M. A., Barral, N., Ibanez, D., Kamenetskiy, D. S., Krakos, J., Michal, T., and Loseille, A., "Unstructured Grid Adaptation and Solver Technology for Turbulent Flows," AIAA Paper 2018-1103, 2018. <https://doi.org/10.2514/6.2018-1103>
- [12] Chrisochoides, N., "Telescopic Approach for Extreme-Scale Parallel Mesh Generation for CFD Applications," AIAA Paper 2016-3181, 2016. <https://doi.org/10.2514/6.2016-3181>
- [13] Chrisochoides, N., and Nave, D., "Parallel Delaunay Mesh Generation Kernel," *International Journal for Numerical Methods in Engineering*, Vol. 58, No. 2, 2003, pp. 161–176. <https://doi.org/10.1002/nme.765>
- [14] Foteinos, P. A., and Chrisochoides, N. P., "High Quality Real-Time Image-to-Mesh Conversion for Finite Element Simulations," *Journal on Parallel and Distributed Computing*, Vol. 74, No. 2, 2014, pp. 2123–2140. <https://doi.org/10.1016/j.jpdc.2013.11.002>
- [15] Chernikov, A. N., and Chrisochoides, N. P., "Parallel Guaranteed Quality Delaunay Uniform Mesh Refinement," *SIAM Journal on Scientific Computing*, Vol. 28, No. 5, 2006, pp. 1907–1926. <https://doi.org/10.1137/050625886>
- [16] Chernikov, A. N., and Chrisochoides, N. P., "Algorithm 872: Parallel 2D Constrained Delaunay Mesh Generation," *ACM Transactions on Mathematical Software*, Vol. 34, No. 1, 2008, pp. 1–20. <https://doi.org/10.1145/1322436.1322442>
- [17] Linardakis, L., and Chrisochoides, N., "Graded Delaunay Decoupling Method for Parallel Guaranteed Quality Planar Mesh Generation," *SIAM Journal on Scientific Computing*, Vol. 30, No. 4, 2008, pp. 1875–1891. <https://doi.org/10.1137/060677276>
- [18] Thomadakis, P., Tsolakis, C., Vogiatzis, K., Kot, A., and Chrisochoides, N., "Parallel Software Framework for Large-Scale Parallel Mesh Generation and Adaptation for CFD Solvers," AIAA Paper 2018-2888, 2018. <https://doi.org/10.2514/6.2018-2888>
- [19] Feng, D., Chernikov, A. N., and Chrisochoides, N. P., "Two-Level Locality-Aware Parallel Delaunay Image-to-Mesh Conversion," *Parallel Computing*, Vol. 59, Nov. 2016, pp. 60–70. <https://doi.org/10.1016/j.parco.2016.01.007>
- [20] Feng, D., Tsolakis, C., Chernikov, A. N., and Chrisochoides, N. P., "Scalable 3D Hybrid Parallel Delaunay Image-to-Mesh Conversion Algorithm for Distributed Shared Memory Architectures," *Computer-Aided Design*, Vol. 85, April 2017, pp. 10–19. <https://doi.org/10.1016/j.cad.2016.07.010>

- [21] Feng, D., Chernikov, A. N., and Chrisochoides, N. P., "A Hybrid Parallel Delaunay Image-to-Mesh Conversion Algorithm Scalable on Distributed-Memory Clusters," *Computer-Aided Design*, Vol. 103, Jan. 2018, pp. 34–46.
<https://doi.org/10.1016/j.cad.2017.11.006>
- [22] Tsolakis, C., Chrisochoides, N., Park, M. A., Loseille, A., and Michal, T., "Parallel Anisotropic Unstructured Grid Adaptation," AIAA Paper 2019-1995, 2019.
<https://doi.org/10.2514/6.2019-1995>
- [23] Michal, T., and Krakos, J., "Anisotropic Mesh Adaptation Through Edge Primitive Operations," AIAA Paper 2012-0159, 2012.
<https://doi.org/10.2514/6.2012-159>
- [24] Park, M. A., and Darmofal, D. L., "Parallel Anisotropic Tetrahedral Adaptation," AIAA Paper 2008-0917, 2008.
<https://doi.org/10.2514/6.2008-917>
- [25] Schloegel, K. D., Karypis, G., and Kumar, V., "Parallel Static and Dynamic Multi-Constraint Graph Partitioning," *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 3, 2002, pp. 219–240.
<https://doi.org/10.1002/cpe.605>
- [26] Devine, K. D., Boman, E. G., Heaphy, R. T., Hendrickson, B. A., Teresco, J. D., Faik, J., Flaherty, J. E., and Gervasio, L. G., "New Challenges in Dynamic Load Balancing," *Applied Numerical Mathematics*, Vol. 52, Nos. 2–3, 2005, pp. 133–152.
<https://doi.org/10.1016/j.apnum.2004.08.028>
- [27] Loseille, A., and Alauzet, F., "Continuous Mesh Framework Part I: Well-Posed Continuous Interpolation Error," *SIAM Journal on Numerical Analysis*, Vol. 49, No. 1, 2011, pp. 38–60.
<https://doi.org/10.1137/090754078>
- [28] Alauzet, F., "A Changing-Topology Moving Mesh Technique for Large Displacements," *Engineering with Computers*, Vol. 30, No. 2, 2014, pp. 175–200.
<https://doi.org/10.1007/s00366-013-0340-z>
- [29] Bossen, F. J., and Heckbert, P. S., "A Pliant Method for Anisotropic Mesh Generation," *Proceedings of the 5th International Meshing Roundtable*, Sandia National Laboratories, Albuquerque, New Mexico, 1996, pp. 63–74.
- [30] Haimes, R., and Drela, M., "On the Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design," AIAA Paper 2012-0683, 2012.
<https://doi.org/10.2514/6.2012-683>
- [31] Haimes, R., and Dannenhoffer, J. F., III, "EGADSLite: A Lightweight Geometry Kernel for HPC," AIAA Paper 2018-1401, 2018.
<https://doi.org/10.2514/6.2018-1401>
- [32] Loseille, A., Menier, V., and Alauzet, F., "Parallel Generation of Large-Size Adapted Meshes," *Procedia Engineering*, Vol. 124, Jan. 2015, pp. 57–69.
<https://doi.org/10.1016/j.proeng.2015.10.122>
- [33] Loseille, A., and Löhner, R., "Anisotropic Adaptive Simulations in Aerodynamics," AIAA Paper 2010-0169, 2011.
<https://doi.org/10.2514/6.2010-169>
- [34] Loseille, A., "Unstructured Mesh Generation and Adaptation," *Handbook of Numerical Methods for Hyperbolic Problems: Applied and Modern Issues, Handbook of Numerical Analysis*, Vol. 18, edited by R. Abgrall, and C.-W. Shu, Elsevier, New York, 2017, pp. 263–302.
<https://doi.org/10.1016/bs.hna.2016.10.004>
- [35] Loseille, A., and Menier, V., "Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive," Springer International, Cham, Switzerland, 2014, pp. 541–558.
https://doi.org/10.1007/978-3-319-02335-9_30
- [36] Loseille, A., and Löhner, R., *Robust Boundary Layer Mesh Generation*, Springer, Berlin, 2013, pp. 493–511.
https://doi.org/10.1007/978-3-642-33573-0_29
- [37] Loseille, A., "Metric-Orthogonal Anisotropic Mesh Generation," *Procedia Engineering*, Vol. 82, Jan. 2014, pp. 403–415.
<https://doi.org/10.1016/j.proeng.2014.10.400>
- [38] Loseille, A., and Alauzet, F., "Continuous Mesh Framework Part II: Validations and Applications," *SIAM Journal on Numerical Analysis*, Vol. 49, No. 1, 2011, pp. 61–86.
<https://doi.org/10.1137/10078654X>
- [39] Park, M. A., Loseille, A., Krakos, J. A., and Michal, T., "Comparing Anisotropic Output-Based Grid Adaptation Methods by Decomposition," AIAA Paper 2015-2292, 2015.
<https://doi.org/10.2514/6.2015-2292>
- [40] Arsigny, V., Fillard, P., Pennec, X., and Ayache, N., "Log-Euclidean Metrics for Fast and Simple Calculus on Diffusion Tensors," *Magnetic Resonance in Medicine*, Vol. 56, No. 2, 2006, pp. 411–421.
<https://doi.org/10.1002/mrm.20965>
- [41] Alauzet, F., "Size Gradation Control of Anisotropic Meshes," *Finite Elements in Analysis and Design*, Vol. 46, Nos. 1–2, 2010, pp. 181–202.
<https://doi.org/10.1016/j.finel.2009.06.028>
- [42] Caplan, P. C., Haimes, R., Darmofal, D. L., and Galbraith, M. C., "Four-Dimensional Anisotropic Mesh Adaptation," *Computer-Aided Design*, Vol. 129, Dec. 2020, Paper 102915.
<https://doi.org/10.1016/j.cad.2020.102915>
- [43] Caplan, P. C. D., "Four-Dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations," Ph.D. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, 2019.
- [44] Alauzet, F., and Loseille, A., "High-Order Sonic Boom Modeling Based on Adaptive Methods," *Journal of Computational Physics*, Vol. 229, No. 3, 2010, pp. 561–593.
<https://doi.org/10.1016/j.jcp.2009.09.020>
- [45] Park, M. A., Balan, A., Anderson, W. K., Galbraith, M. C., Caplan, P. C., Carson, H. A., Michal, T., Krakos, J. A., Kamenetskiy, D. S., Loseille, A., Alauzet, F., Frazza, L., and Barral, N., "Verification of Unstructured Grid Adaptation Components," AIAA Paper 2019-1723, 2019.
<https://doi.org/10.2514/6.2019-1723>
- [46] Loseille, A., Alauzet, F., and Menier, V., "Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes," *Computer-Aided Design*, Vol. 85, April 2017, pp. 53–67.
<https://doi.org/10.1016/j.cad.2016.09.008>
- [47] Gustafson, J. L., "Reevaluating Amdahl's law," *Communications of the ACM*, Vol. 31, No. 5, 1988, pp. 532–533.
<https://doi.org/10.1145/42411.42415>
- [48] Zhou, B. Y., Diskin, B., Gauger, N. R., Pardue, J., Chernikov, A., Tsolakis, C., Drakopoulos, F., and Chrisochoides, N., "Hybrid RANS/LES Simulation of Vortex Breakdown Over a Delta Wing," AIAA Paper 2019-3524, 2019.
<https://doi.org/10.2514/6.2019-3524>
- [49] Ibanez, D. A., "Conformal Mesh Adaptation on Heterogeneous Supercomputers," Ph.D. Thesis, Rensselaer Polytechnic Inst., Troy, NY, Nov. 2016, <https://search.proquest.com/docview/1858722545/abstract/8598B86A0EC4FABPQ/1> [retrieved 2021].
- [50] Tsolakis, C., "A Unified Framework for Parallel Anisotropic Mesh Adaptation," Ph.D. Thesis, Old Dominion Univ., Norfolk, VA, Aug. 2021.

C. Pantano
Associate Editor