# PARALLEL FOUR-DIMENSIONAL ANISOTROPIC MESH ADAPTATION

Philip Claude Caplan

*Middlebury College Department of Computer Science, Middlebury, VT, U.S.A.,*
*pcaplan@middlebury.edu*

## ABSTRACT

Anisotropic mesh adaptation is important for accurately predicting engineering quantities of interest with high-order finite element discretizations. Spacetime approaches, whereby time is treated as an additional spatial dimension and the mesh is fully coupled in space and time, have also shown to be effective in reducing the number of degrees of freedom needed to achieve a certain level of accuracy. Previous efforts have successfully demonstrated four-dimensional mesh adaptation capabilities so as to support adaptive simulations in $3d + t$. However, these algorithms have been restricted to sequential implementations. Parallel mesh adaptation strategies have become increasingly important in $3d$ numerical simulations. Therefore, the goal of this work is to extend those strategies to perform parallel four-dimensional anisotropic mesh adaptation. We employ a functionality-first approach to parallelize an existing sequential mesh adaptation tool. This approach iteratively adapts the mesh while keeping partition interfaces fixed, and then migrates partition interfaces into the interior so they can be adapted during a subsequent pass within an adaptation iteration. We demonstrate that the algorithm scales well as the number of processors increases while maintaining good metric conformity for three- and four-dimensional problems.

**Keywords: mesh adaptation, anisotropic, parallel, four-dimensional, pentatope, spacetime**

## 1. INTRODUCTION

Anisotropic mesh adaptation is fundamental for accurately predicting engineering quantities of interest with high-order finite element discretizations [1, 2]. To resolve unsteady physical phenomena in $d$ dimensions, space and time can be coupled and the problem can be solved in a spacetime domain, whereby time is treated as an additional spatial dimension. This approach requires a $(d + 1)$-dimensional mesh, which means $4d$ meshes are needed to solve unsteady $3d$ problems.

Adaptation strategies in this spacetime domain can be categorized into (1) uniform, (2) tensor-product [3, 4, 5, 6, 7, 8] or (3) unstructured anisotropic approaches [9, 10, 11, 12, 13]. Yano [9] demonstrated that tensor-product approaches are effectively isotropic and, whereas they offer a substantial degree-of-freedom (DOF) savings over uniform refinement, they are dramatically outperformed by a fully unstructured anisotropic approach. In the context of the one-dimensional problem of Figure 1 with a propagating feature of characteristic size $\delta$, Yano [9] experimentally observes that uniform, tensor-product, and unstructured anisotropic refinement approaches respectively require $\mathcal{O}(\delta^{-2})$, $\mathcal{O}(\delta^{-1})$ and $\mathcal{O}(1)$ DOF to achieve the same level of accuracy in the simulation. This unstructured anisotropic spacetime approach has been extended to $2d + t$ for convection-diffusion [9], wave equation [14] and oil reservoir simulations [11] and to $3d + t$ for convection-diffusion problems [13]. Jayasinghe also demonstrates that a spatiotemporal approach scales very well with the number of parallel processors when compared to time-marching approaches [12]. However, to fully realize the potential of spacetime adaptive methods, large meshes, with millions or even billions of elements, are needed for practical engineering problems, thereby requiring a parallel mesh adaptation algorithm.

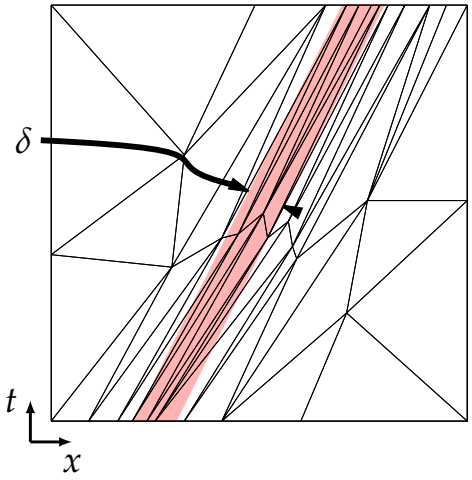



**Figure 1**: Unstructured spacetime mesh adaptivity can capture a feature with nominal size $\delta$ using $\mathcal{O}(1)$ elements, whereas tensor-product and uniform refinement methods would require $\mathcal{O}(\delta^{-1})$ and $\mathcal{O}(\delta^{-2})$ elements, respectively, to achieve the same level of accuracy.

Chrisochoides describes a telescopic approach for parallel mesh generation and adaptation at various levels of granularity [15]. Further, Tsolakis et al. give an excellent overview of the goals and strategies for parallel mesh adaptation [16]. The authors identify five criteria which can be used to evaluate parallel mesh adaptation codes: (1) stability, (2) reproducibility, (3) robustness, (4) scalability and (5) code reuse. Similar to the work of Tsolakis, we aim for stability and scalability. That is, we aim to maintain good metric conformity while ensuring the algorithm scales well as the number of processors is increased. Tsolakis also identifies two different approaches for parallel mesh generation and adaptation: functionality-first and scalability-first approaches. Codes such as EPIC [17] and Feflo.a [18] fit into the functionality-first paradigm, whereby minimal changes are made to existing sequential mesh adaptation software to extend them to a parallel setting. Codes such as CDT3D [19] and refine [20] are classified as scalability-first approaches, whereby functionalities are completed as needed, but designed to be scalable across parallel processes first.

The primary goal of this work is to parallelize a four-dimensional mesh adaptation algorithm and demonstrate that good metric conformity is obtained as the number of processors increases. Our contribution is an extension of existing techniques for $3d$ parallel mesh adaptation to $4d$, with a demonstration on benchmark cases. We also show the algorithm scales well with modest processor counts - we only present results for up to 36 processors due to limitations on our computational resources. We use a functionality-first approach whereby we maximize the code re-use of an existing sequential mesh adaptation algorithm. We first elaborate upon the sequential and parallel mesh adaptation strategies. The latter builds from the idea of fixing partition boundaries, and migrating the partition interfaces into the interior. We then study the algorithm on three- and four-dimensional problems and demonstrate that the parallel algorithm scales well as the number of processors increases, while also maintaining good metric conformity.

## 2. SEQUENTIAL MESH ADAPTATION STRATEGY

Our sequential mesh adaptation algorithm is based on the local cavity operator approach, initially described by Coupez [21, 22], later used by Loseille [23, 18] and extended to $4d$ by Caplan [13, 24, 25]. The inputs to the algorithm are (1) an initial mesh, (2) a metric field described at the vertices of the mesh and (3) the geometry entity associated with each vertex of the mesh. These entities could be geometry Nodes, Edges, Faces or Volumes (for a $4d$ geometry) which can be specified via a geometry engine, such as EGADS [26, 27] for $3d$ geometries. In our work, we study the geometry of a unit tesseract, which is bounded by 8 Cubes (Volumes), 10 Faces, 32 Edges and 16 Nodes. The topology of the tesseract is prescribed by traversing the vertex-facet incidence relations of each geometry entity [13]. Each vertex in the mesh is initially tagged with the lowest-dimensional geometry entity on which it lies, which is null for interior vertices that do not lie on the geometry.

The local cavity operator approach iteratively modifies the mesh $\mathcal{T}^k$ by removing a set of cavity elements $\mathcal{C}^k$ around a mesh facet $f$ (such as a vertex, edge, triangle or tetrahedron), and then selecting a re-insertion vertex $p$ to connect to the cavity boundary $\partial C^k$, thereby creating new elements denoted by $\mathcal{B}^k$:

$$\mathcal{T}^{k+1} = \mathcal{T}^k \setminus \mathcal{C}^k(f) \cup \mathcal{B}^k(p,\ \partial\mathcal{C}^k). \qquad (1)$$

The advantage of the cavity-based approach is that all classical mesh modification operators (splits, collapses, swaps and smoothing) can be reformulated in terms of Eq. 1 by the appropriate selection of $f$ and $p$, thus simplifying the implementation in $4d$.

To schedule the local mesh modification operators, a metric field is specified at each vertex of the initial mesh. That is, a symmetric, positive-definite $d \times d$ tensor is prescribed at each vertex. The input mesh is retained in the background of the adaptation process - the metric at a new location in the mesh (either through an edge split or a vertex relocation) is ob-

tained by the log-Euclidean weighted average of the metrics from an element in this background mesh.

Following the conventions of the Unstructured Grid Adaptation Working Group (UGAWG) [16, 28, 29], the length of an edge between two vertices $p$ and $q$ is computed as

$$\ell_{\mathbf{m}}(\mathbf{p},\mathbf{q}) \approx \ell_{\mathbf{m}(\mathbf{p})} \frac{r-1}{r \log r} \quad \text{with } r \equiv \frac{\ell_{\mathbf{m}(\mathbf{p})}}{\ell_{\mathbf{m}(\mathbf{q})}}, \qquad (2)$$

where $\ell_{\mathbf{m}(\mathbf{x})} = \sqrt{(\mathbf{q}-\mathbf{p})^t \mathbf{m}(\mathbf{x})(\mathbf{q}-\mathbf{p})}$.

The quality of a $d$-simplex $\boldsymbol{\kappa}$ ($d = 2$ for triangles, 3 for tetrahedra and 4 for pentatopes) is computed as

$$q_{\mathbf{m}}(\boldsymbol{\kappa}) = \beta_n \frac{v_{\mathbf{m}}(\boldsymbol{\kappa})^{2/d}}{\sum\limits_{e \in \mathcal{E}(\kappa)} \ell^2_{\mathbf{m}}(e)} \qquad (3)$$

where $\mathcal{E}$ is the set of $d(d+1)/2$ edges of the simplex, and $v_{\mathbf{m}}(\boldsymbol{\kappa})$ is the volume under the metric field:

$$v_{\mathbf{m}}(\boldsymbol{\kappa}) \approx \sqrt{\det \mathbf{m}_\nu}\, v(\boldsymbol{\kappa}), \quad \text{with } \nu = \arg\max_{\nu \in \kappa} \det \mathbf{m}_\nu. \qquad (4)$$

The goal of the mesh adaptation algorithm is to produce a metric-conforming mesh in which all edges of the mesh have a length in the range $[\sqrt{2}/2, \sqrt{2}]$ and all elements have a quality greater than 0.8. To achieve these goals, the algorithm begins by iteratively collapsing all edges shorter than $\sqrt{2}/2$ and then performing edge splits on edges longer than 2 without introducing new short edges. Next, edge swaps are used to improve the quality of the elements in two passes: first all edges adjacent to an element with quality less than 0.4 are targeted, then all elements with a quality less than 0.8 are targeted. A local vertex smoothing procedure is then used to drive the edge lengths surrounding a particular vertex to 1. This overall procedure is then repeated using a target edge length of $\sqrt{2}$ for the split operator. For further details on this algorithm, please see the work of Caplan [13, 25].

## 3. PARALLEL MESH ADAPTATION STRATEGY

We use a functionality-first approach to parallelize the mesh adaptation algorithm and attempt to re-use as much of the sequential mesh adaptation algorithm as possible. This approach is similar to the work of Digonnet [30, 31, 32] and Lachat [33, 34, 35, 36] in which the mesh is divided into *processing elements* to be remeshed by a third-party sequential remesher, such as MMG3D [37]. Other coarse-grained parallel implementations which modify partition interiors, while keeping interfaces fixed include REFINE [20],

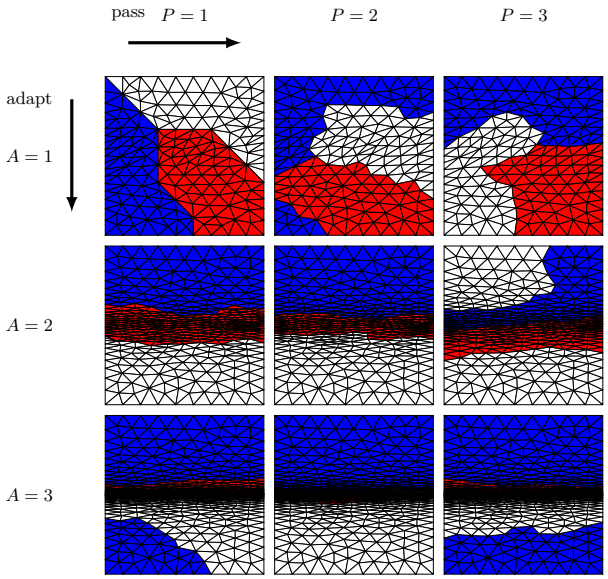



**Figure 2**: Parallel adaptation strategy. Each row corresponds to a single adaptation iteration ($A$), whereas each column corresponds to a pass ($P$) within the adaptation iteration. Each partition (three in this example) is highlighted with a different color. The goal of the repartitioner after each pass is to migrate the interfaces between partitions into the partition interiors.

EPIC [17] and FEFLO.A [38, 39, 18]. CDT3D uses a fine-grained approach to modifying the mesh, designed for shared-memory parallel mesh adaptation [40].

The main idea behind the parallel mesh adaptation algorithm is to decompose the initial mesh (or receive a decomposed mesh from a numerical simulator) into partitions and adapt the interior of each partition while keeping the partition boundaries fixed. As long as the sequential mesh adaptation algorithm allows for these partition boundaries to be frozen, then the sequential algorithm can be used within each partition without requiring any intrusive changes to communicate boundary modifications. Communicating these boundary modifications would require special treatment for each mesh modification operator and would need to be directly incorporated into the sequential mesh adaptation code [41], which we chose to avoid. In our setting, however, the difficulty lies in the fact that partition boundaries must be migrated to the partition interiors so as to ensure the mesh is metric-conforming after each adaptation iteration. In order to migrate the partition boundaries into the interior, a series of "passes" are used within a particular iteration of the global mesh adaptation procedure - each pass is then followed by a repartition of the mesh. Here we use PARMETIS [42] to repartition the mesh, though other

tools such as PT-SCOTCH [43] or ZOLTAN [44] could be used. The simplex-to-simplex adjacencies are used to construct the graph that is passed to PARMETIS. That is, each simplex is a vertex of the graph, and an edge in the graph is created if two simplices share a common $(d-1)$-facet (i.e. a $(d-1)$-simplex). The primary goal of the repartitioning procedure is to migrate existing partition boundaries into the interior for subsequent passes. However, some $(d-1)$-facets in the partition interfaces may not require modification, therefore, it would be less necessary to migrate these into the interior, when other areas in the mesh may require more attention. As a result, we penalize edges by imposing edge weights which are computed via the "age" of the vertices of the elements in the mesh, an idea inspired by REFINE [20]. The age of a vertex is defined as the number of rejected mesh modifications of any facet surrounding the vertex. Whenever a local cavity operator is accepted, the age of any vertices in the re-inserted ball become zero. Note that this procedure is heuristic, and does not *guarantee* that partition boundaries are migrated to interior.

An example of the parallel mesh adaptation strategy in $2d$ is given in Fig. 2. Each row corresponds to a particular adaptation iteration $(A)$, whereas each column corresponds to a pass within the adaptation iteration $(P)$. To clarify, each adaptation iteration corresponds to an iteration of the adaptive numerical simulation, whereby a solution is obtained, the error is estimated and an optimal metric field is computed. This metric field, along with the current mesh, is then passed to the mesher. Within this call to the mesher, multiple *passes* are performed (with the same metric field) with the goal of migrating the partition interfaces so as to adapt any necessary regions of the domain before returning the next mesh to the solver.

In Fig. 2, we can see that the boundaries of the partitions are migrated to the partition interiors from one pass to the next - however, in some cases it is not possible to migrate *every* interface $(d-1)$-facet. Also observe that partitions may be divided across multiple connected components, as in Fig. 2 for $A=2$ and $P=3$. Furthermore, partition boundaries may exhibit arbitrarily shaped geometries, especially in $3d$ and $4d$. The only modification required in our sequential mesh adaptation algorithm was to allow for vertices to be fixed during the adaptation. In particular, each partition fixes any vertex along a partition interface. We additionally needed to add checks for the non-manifold vertices that are produced for arbitrarily shaped partitions.

One difficulty is that we would like all global indices of the fixed vertices to remain the same across all processors. This is not a problem when an edge is split, swapped, or when a vertex is smoothed, however, special care is required for the collapse operator. Since the collapse requires the removal of a vertex, any element that references a vertex with a higher index than the removed vertex must have its vertex indices decremented. This would require communication between processors whenever a collapse is performed. Therefore, all fixed vertices are initially moved to the front of the global list of vertices in the mesh. Other synchronization techniques are possible, but this was the simplest to implement within our framework.

After each re-partitioning procedure, the processors must communicate which elements are retained, which are sent to other partitions, and which are received from other processors. We use the Message Passing Interface (MPI) to exchange this information between processors [45]. The communication cost of this exchange operation can be expensive, which we study in the following sections.

Similar to the example in Fig. 2, we use three passes per adaptation iteration since we experimentally observed that this achieved good metric conformity while not being too costly.

## 4. NUMERICAL EXPERIMENTS

To evaluate the parallel mesh adaptation algorithm, we follow an approach similar to the work of Tsolakis [16, 46]. Starting with an initial mesh, conforming to a prescribed metric, we scale the metric and call our parallel mesh adaptation algorithm using a prescribed number of processors. The difference between our work and the work of Tsolakis, however, is that since our sequential mesh adaptation algorithm is designed to make small changes in a mesh, we scale the metric iteratively instead of scaling it to a desired complexity all at once. Specifically, our sequential mesh adaptation algorithm works best when the incoming edge lengths are between $[0.5, 2.0]$, motivated by the MOESS algorithm [9, 47, 48].

We study two types of cases: (1) when the metric field is analytic and (2) when the metric field is obtained from the MOESS algorithm, and only available at the discrete vertices of the initial mesh. When the metric field is analytic, each iteration $i$ of our scaling procedure consists of evaluating the prescribed metric on the current mesh and then scaling the metric by a factor $s = \sqrt{2}^i$. When the metric field is discrete, we simply use a scaling factor of $s = \sqrt{2}$ on the metric of the *current* mesh in the scaling iteration. Each component of the metric field is multiplied by the scaling factor $s$, which amounts to a multiplication of each eigenvalue of the metric field by $s$. Thus, the number of elements increases by a factor of $\sqrt{s}^d$ after each iteration, while maintaining the same anisotropic ratios as in the initial mesh.

```
scaleMeshParallel

    input: initial mesh M, metric m
    output: M
1   M_p ← partition(M, p)
2   for i = 1 to n_iter
3       m_p = evaluateMetric(m, M_p)
4       fix partition boundary ∂M_p
5       for j = 1 to n_pass
6           M_p ← adapt(M_p, m_p)
7           balance the mesh (repartition & exchange)
8   accumulate meshes from all processors into M
```

**Algorithm 1:** Scaling an initial mesh $\mathcal{M}$ according to a metric field $\mathbf{m}$ (either discrete or analytic) in parallel. The input mesh is initially partitioned and distributed across all parallel processors. For each adaptation iteration, the metric is re-evaluated on the current mesh and scaled to produce the next mesh in the adaptation sequence. Each adaptation iteration involves $n_{\text{pass}}$ passes of the parallel adaptation algorithm so as to migrate partition interfaces to the interior. The final mesh is then accumulated on the root processor in order to evaluate metric conformity.

This procedure is outlined in Algorithm 1. After performing an initial partitioning of the mesh (line 1), each processor receives a subdomain mesh $\mathcal{M}_p$. We then iterate until the desired complexity is reached. Each processor evaluates the metric on the current mesh and scales it according to the scaling iteration counter. Then the parallel mesh adaptation procedure begins by computing the partition boundaries, labelling vertices which should be fixed, and then moving fixed vertices to the beginning of the vertex list (line 4). Each processor then performs 3 passes of adapting the mesh, migrating partition interfaces to the interior (with a penalty on the age of an element), and then exchanging the elements (and discrete metrics at the vertices) with all other processors. When the procedure is complete, all meshes are accumulated into the final mesh $\mathcal{M}$ (line 8) which is then used to evaluate metric conformity. The latter is simply needed as a post-processing procedure which does not contribute to the analysis of the timing results. In the following, we measure the time spent adapting the mesh (line 6) and performing the load balancing, which consists of repartitioning and exchanging elements across processors (line 7) since these are the most costly operations. Adapting the mesh is clearly the bottleneck we wish to parallelize, but a poor design of the mesh exchange data structures and procedure can cause a significant overhead that is counterproductive to the parallel mesh adaptation process.

In each of the following subsections, we will first assess the ability of the parallel implementation to conform to the scaled metric field, whether analytic or discrete. We will then revisit the scalability of the algorithm in the final subsection.

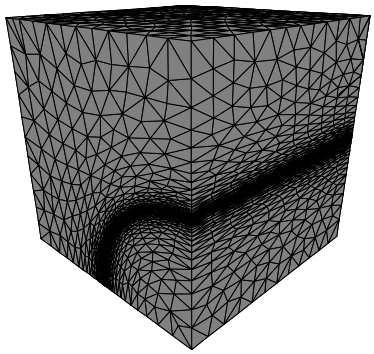


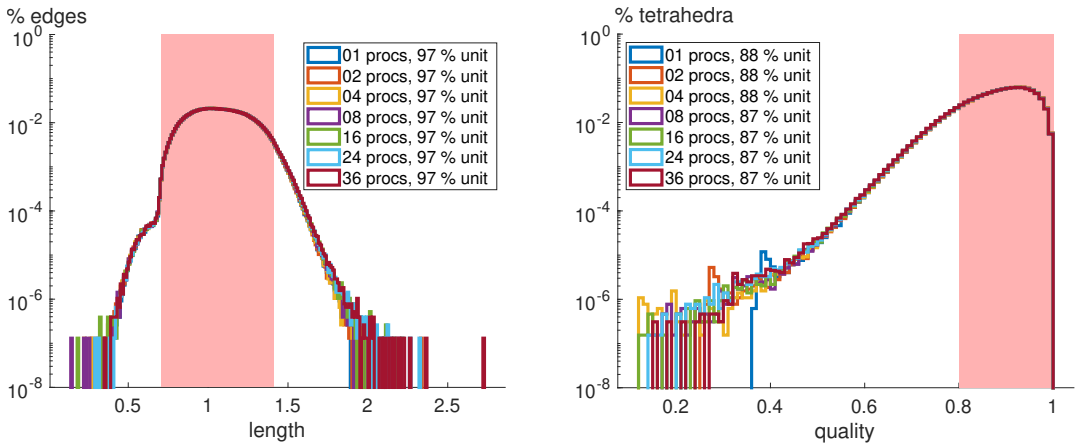
**Figure 3**: Initial 61k tetrahedra mesh for the Cube-Polar case.

## 4.1 Cube Polar (CP)

Before delving into some $4d$ cases, let us first evaluate the algorithm on a $3d$ benchmark case proposed by the UGAWG. The initial mesh passed into Alg. 1 is generated by adapting an initially structured mesh to the analytic metric:

$$\mathbf{m}(\mathbf{x}) = \mathbf{q}\,\mathrm{diag}(h_r^{-2}, h_\theta^{-2}, h_z^{-2})\,\mathbf{q}^t \qquad (5)$$

where $h_z = 0.1$, $h_r = 0.001 + 0.198|r - 0.5|$ and $h_\theta = 0.1d + 0.025(1-d)$ where $d = \min(10|r-0.5|, 1)$. Note that $r = \sqrt{x^2+y^2}$ and $\theta = \arctan(y, x)$. The eigenvectors $\mathbf{q}$ consists of the unit vectors representing the cylindrical system unit vectors in a Cartesian system. The initial mesh, consisting of 63,237 tetrahedra and 12,242 vertices, is shown in Fig. 3. Alg. 1 is then used to scale this initial mesh and analytic metric to a mesh with approximately 6.5 million tetrahedra using various numbers of processors. Specifically, Alg. 1 is run with 1, 2, 4, 8, 16, 24 and 36 processors. The edge length and quality histograms of Fig. 4 demonstrate that the final meshes exhibit excellent metric conformity, regardless of the number of processors that were used to scale the mesh. Table 2 further shows that about 97% of the edges are in the quasi-unit range and the number of tetrahedra with a quality greater than 0.8 is at least 87% for all cases. Furthermore, the number of elements in the final mesh is almost identical across each test case. Although this test case is still relatively small in the context of a practical engineering simulation, it is large enough such that the processor interface elements can be effectively migrated to the interior so as to achieve good metric conformity.




(a) Edge length (b) Tetrahedron quality

**Figure 4**: Edge length and pentatope quality distributions for the $3d$ Cube-Polar case.

**Table 1**: Metric conformity for the Cube-Polar case running with $p$ processors, generating $n_{\text{elem}}$ tetrahedra: min. ($\ell_{\min}$) and max. ($\ell_{\max}$) edge lengths, percentage of edges in the quasi-unit range ($\%\ell_{\text{unit}}$), min. quality ($q_{\min}$) and percentage of tetrahedra with quality $\geq 0.8$ ($\%q_{\text{unit}}$).

| $p$ | $\ell_{\min}$ | $\ell_{\max}$ | $\%\ell_{\text{unit}}$ | $q_{\min}$ | $\%q_{\text{unit}}$ | $n_{\text{elem}}$ |
|---|---|---|---|---|---|---|
| 1 | 0.41 | 2.11 | 97% | 0.36 | 88% | 6.46m |
| 2 | 0.28 | 2.33 | 97% | 0.25 | 88% | 6.46m |
| 4 | 0.22 | 2.10 | 97% | 0.12 | 88% | 6.45m |
| 8 | 0.14 | 2.19 | 97% | 0.17 | 87% | 6.45m |
| 16 | 0.18 | 2.21 | 97% | 0.12 | 87% | 6.45m |
| 24 | 0.29 | 2.33 | 97% | 0.14 | 87% | 6.45m |
| 36 | 0.36 | 2.73 | 97% | 0.15 | 87% | 6.45m |

**Table 2**: Metric conformity for the Tesseract-Linear case running with $p$ processors, generating $n_{\text{elem}}$ pentatopes: min. ($\ell_{\min}$) and max. ($\ell_{\max}$) edge lengths, percentage of edges in the quasi-unit range ($\%\ell_{\text{unit}}$), min. quality ($q_{\min}$) and percentage of pentatopes with quality $\geq 0.8$ ($\%q_{\text{unit}}$).

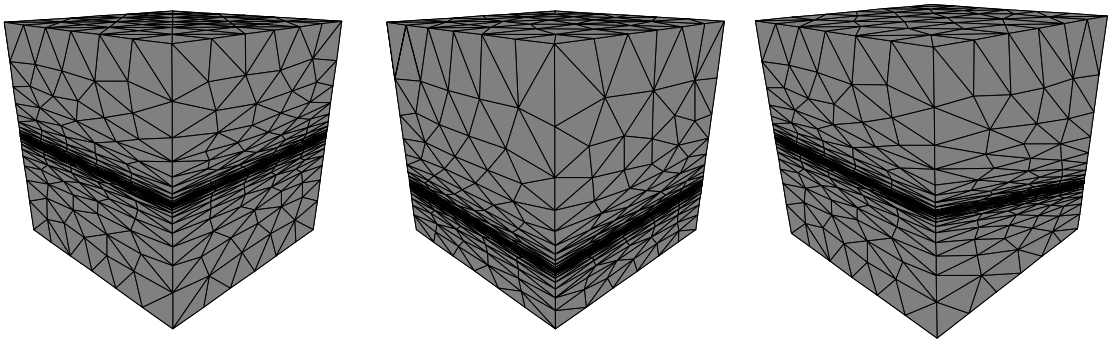
| $p$ | $\ell_{\min}$ | $\ell_{\max}$ | $\%\ell_{\text{unit}}$ | $q_{\min}$ | $\%q_{\text{unit}}$ | $n_{\text{elem}}$ |
|---|---|---|---|---|---|---|
| 1 | 0.16 | 2.43 | 93% | 0.25 | 52% | 7.47m |
| 2 | 0.23 | 2.53 | 93% | 0.29 | 51% | 7.42m |
| 4 | 0.20 | 3.00 | 93% | 0.27 | 51% | 7.37m |
| 8 | 0.16 | 4.60 | 92% | 0.20 | 50% | 7.32m |
| 16 | 0.08 | 4.88 | 89% | 0.11 | 45% | 6.66m |
| 24 | 0.11 | 5.37 | 86% | 0.11 | 41% | 5.95m |
| 36 | 0.07 | 4.55 | 84% | 0.07 | 37% | 5.63m |

## 4.2 Tesseract Linear (TL)

Now, we will study some $4d$ problems. The initial mesh for the Tesseract Linear (TL) case was generated from our sequential $4d$ mesh adaptation code using an analytic metric that emulates a moving boundary layer, described by

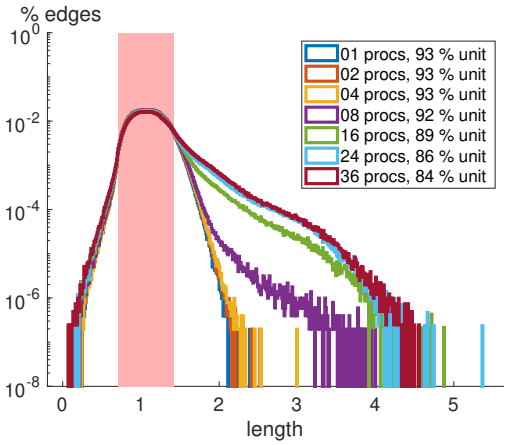
$$\mathbf{m}(\mathbf{x}) = \mathbf{q}\,\text{diag}(h_x^{-2}, h_y^{-2}, h_z^{-2}, h_t^{-2})\,\mathbf{q}^t$$

where $\mathbf{q}$ is a rotation matrix of an angle $\alpha = \arctan(0.25, 1)$ radians about the $xy$ plane, $h_x = h_y = h_z = 0.2$ and $h_z = 0.002 + 0.396d$ with $d = |\cos\alpha z - \sin\alpha t - 0.5\cos\alpha|$), which is the distance to the rotated plane of the boundary layer. The initial mesh contains 120,054 pentatopes and 8,297 vertices. The boundaries of this initial mesh, extracted at the $t = 0$, $t = 1$ and $x = 0$ hyperplanes are shown in Fig. 5. Using Alg. 1 with this initial mesh, the analytic metric is scaled to preserve the anisotropy ratios to eventually produce meshes with 6-7 million pentatopes. Fig. 6 shows the length and quality histograms, measured under the scaled analytic metric, for the final meshes produced using 1, 2, 4, 8, 16,, 24 and 36 processors. Overall, metric conformity is quite similar for all numbers of processors, with a slight degradation in the fraction of quasi-unit edges and quasi-unit pentatopes as the number of processors increases. The final meshes are still quite small - the added number of processors for a small mesh produces more interface elements that are frozen during the parallel adaptation process. These metric conformity results are summarized in Table 2. With $1-8$ processors, over 90% of the edges are in the quasi-unit range, and over 50% of the pentatopes have a quality above 0.8. These results are similar to initial ratios of the edges and pentatopes within the quasi-unit ranges for the initial mesh of 120k pentatopes. These statistics, however, degrade to 89%, 86% and 84% in the fraction of quasi-unit edges, and 45%, 41% and 37% in quasi-unit pentatopes for 16, 24 and 36 processors, respectively.

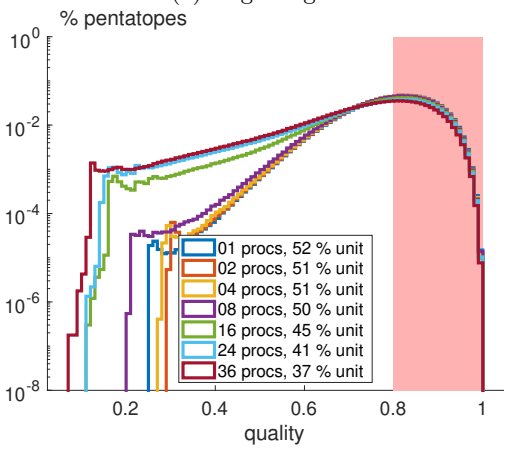


(a) $t = 0$ (b) $t = 1$ (c) $x = 1$

**Figure 5**: Boundaries of the initial 100k pentatope mesh for the Tesseract-Linear case.




(a) Edge length




(b) Pentatope quality

**Figure 6**: Edge length and pentatope quality distributions for the $4d$ Tesseract Linear case.

## 4.3 Tesseract Wave (TW)

Next, we consider a case in which the initial mesh was obtained from the Mesh Optimization via Error Sampling and Synthesis (MOESS) algorithm. Specifically, the mesh was obtained by adapting to the $L^2$ error in the function

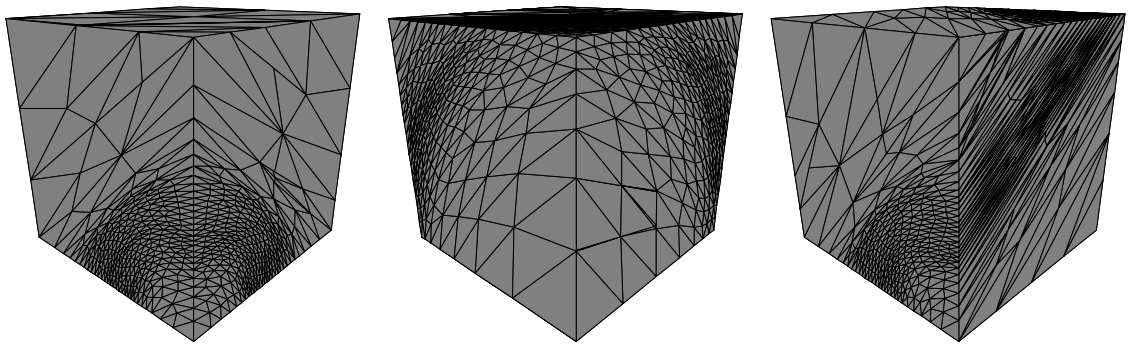
$$u(\mathbf{x}, t) = \exp(-t) \exp\left(-200\left(r(t) - ||\mathbf{x}||\right)^2\right),$$

with $r(t) = 0.4 + 0.7t$, using a linear discontinuous Galerkin discretization. The initial mesh consists of 233,248 pentatopes and 14,515 vertices - the boundaries of this mesh, extracted at the $t = 0$, $t = 1$ and $x = 0$ hyperplanes are shown in Fig. 7. Due to symmetry, we are only modeling one eighth of the expanding spherical wave. The sphere at it's initial radius is visible at $t = 0$ (Fig. 7a), and at it's final radius at $t = 1$ (Fig. 7b). Along hyperplanes with a non-constant $t$, we should see the geometry of a $3d$ cone, which is the projection of the expanding sphere in $4d$ (a $4d$ hypercone). The final metric obtained from the MOESS algorithm is then saved and used as the initial discrete metric, which is then scaled according to Alg. 1 to achieve meshes with approximately 7 million pentatopes.

As in the previous case, metric conformity is excellent (90% of the edges have a length in the quasi-unit range, and 40% of the pentatopes have a quality greater than 40%) when the number of processors is still quite low - specifically up to 8 processors. The edge length and pentatope quality histograms are shown in Fig. 8, and the metric conformity statistics are summarized in Fig. 3. Again, metric conformity slightly degrades when the mesh is over decomposed.

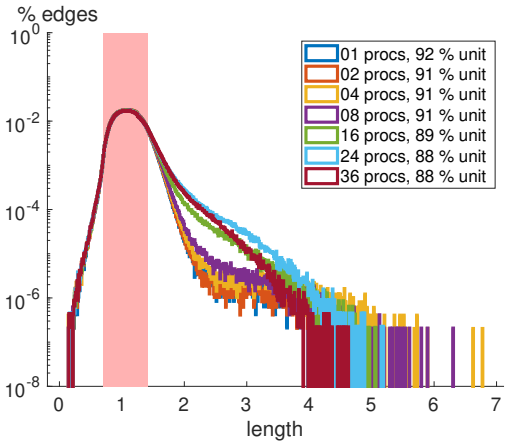
## 4.4 Scalability

Finally, we analyze the scalability of our parallel mesh adaptation implementation. Due to limitations on our

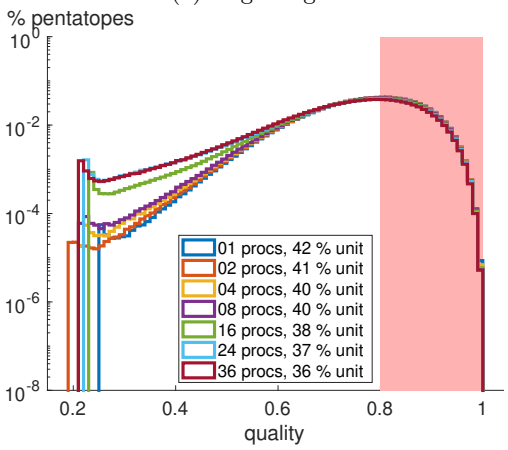


(a) $t = 0$ (b) $t = 1$ (c) $x = 1$

**Figure 7**: Boundaries of the initial 200k pentatope mesh for the Tesseract-Wave case.




(a) Edge length




(b) Pentatope quality

**Figure 8**: Edge length and pentatope quality distributions for the 4$d$ Tesseract Wave case.

**Table 3**: Metric conformity for the Tesseract-Wave case running with $p$ processors, generating $n_{\text{elem}}$ pentatopes: min. ($\ell_{\min}$) and max. ($\ell_{\max}$) edge lengths, percentage of edges in the quasi-unit range ($\%\ell_{\text{unit}}$), min. quality ($q_{\min}$) and percentage of pentatopes with quality $\geq 0.8$ ($\%q_{\text{unit}}$).

| $p$ | $\ell_{\min}$ | $\ell_{\max}$ | $\%\ell_{\text{unit}}$ | $q_{\min}$ | $\%q_{\text{unit}}$ | $n_{\text{elem}}$ |
|---|---|---|---|---|---|---|
| 1 | 0.20 | 4.82 | 92% | 0.25 | 42% | 6.86m |
| 2 | 0.20 | 5.37 | 91% | 0.19 | 41% | 6.99m |
| 4 | 0.14 | 6.77 | 91% | 0.21 | 40% | 7.05m |
| 8 | 0.14 | 6.31 | 91% | 0.21 | 40% | 6.90m |
| 16 | 0.18 | 5.06 | 89% | 0.23 | 38% | 6.59m |
| 24 | 0.17 | 5.19 | 88% | 0.22 | 37% | 6.19m |
| 36 | 0.15 | 4.63 | 88% | 0.21 | 36% | 6.37m |

computational resources, this study is restricted to smaller numbers of processors (up to 36). For runs with $1-24$ processors, we used `node A` of Table 4 and we used `node B` for runs with 36 processors. For every test case, we report the total time to scale the initial mesh to the final mesh (i.e. the total time spent in Alg. 1). We also break down this total cost into the time spent adapting and balancing the mesh, since these were the most costly. The remaining time was spent synchronizing the indices, pre-processing the partitions (including moving the fixed vertices to the front of the arrays), and the actual re-partitioning of the mesh using ParMETIS.

The timing results are tabulated in Tables 5, 6 and 7. The total number of iterations used in Alg. 1 was 7, 6 and 10 for the Tesseract Linear, Tesseract Wave and Cube Polar cases, respectively. We also report the number of elements $n_{\text{elem}}$ and partition interface ($d-1$)-facets $n_{\partial\text{P}}$. Note that, in the case of the sequential algorithm ($p = 1$), we still perform the algorithm with 3 passes, but only report the timing for a single pass.

**Table 4**: Machines used in the scalability analysis.

| | |
|---|---|
| `node A` | 28 × Intel Xeon Gold 5120 @ 2.20GHz |
| `node B` | 36 × Intel Xeon Gold 6140 @ 2.30GHz |

Next, we analyze the scalability of the parallel algorithm with respect to the sequential version. To be fair to the sequential adaptation algorithm, we only include adaptation time for one pass in our scalability analysis - there is clearly no need to include the time to migrate and re-partition the mesh, but we still perform some of the pre-processing steps which is ignored in the timing of the sequential algorithm. Thus, the final speedup of the parallel algorithm is measured as the total time (for all passes of the parallel algorithm) divided by the time for one pass of the sequential algorithm (accumulated across all iterations), which is further normalized by the number of elements since each test case produced a different number of elements. The speedup obtained with various processors for all three test cases considered in this paper is plotted in Fig. 9. A dashed line is used to show the ideal linear speedup. Our algorithm exhibits a linear speedup with only a few processors (up to 8), however, it begins to exhibit a superlinear speedup with more processors.

**Table 5**: Total time ($t_{\text{total}}$) for the Cube-Polar case running with $p$ processors, generating $n_{\text{elem}}$ tetrahedra, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface triangles ($n_{\partial P}$) in the final repartitioned mesh.

| $p$ | $n_{\text{elem}}$ | $n_{\partial P}$ | $t_{\text{total}}$ | % adapt | % bal. |
|---|---|---|---|---|---|
| 1 | 6.46m | - | 1d:4h | 100.0% | - |
| 2 | 6.46m | 20.0k | 1d:24m | 93.1% | 6.3% |
| 4 | 6.45m | 39.3k | 7h:9m | 90.1% | 8.9% |
| 8 | 6.45m | 72.4k | 2h:33m | 87.8% | 10.7% |
| 16 | 6.45m | 118.3k | 56m:32s | 86.7% | 11.5% |
| 24 | 6.45m | 144.5k | 31m:11s | 86.2% | 11.8% |
| 36 | 6.45m | 178.5k | 16m:37s | 82.6% | 15.5% |

The work performed by the mesh adaptation algorithm is not linear in the size of the mesh, so the speedup will be superlinear when the problem size is evenly distributed across parallel processors. A previous analysis of the sequential mesh adaptation operators [25] suggests the work performed by, for example, the insertion operator increases linearly with the size of the mesh. Thus, the total work required to insert vertices for a particular problem is roughly the number of edges to be split times the size of the mesh. In parallel, the number of edges to be split is ideally distributed evenly across all processors, and the size of the partition is clearly smaller than the total size of the mesh. Thus the cost of the insertion operator is dramatically reduced (more than linear) when the work is divided across a large number of parallel processors. The fact that the mesh adaptation algorithm exhibits this superlinear convergence while maintaining reasonable metric conformity is a promising result for large-scale $3d + t$ numerical simulations. Finally, we report the time (and number of adaptation iterations) to scale the mesh to much larger sizes with 36 processors for the same three test cases. All results were obtained on `node B` of Table 4. Total run time is reasonable, taking less than two days to perform 10 adaptation iterations to achieve a mesh with 69 million pentatopes, and roughly 9 hours to perform 14 adaptation iterations to achieve a mesh with about 52 million tetrahedra. Furthermore, metric conformity is better with these large meshes (compared with the results on smaller meshes in the previous section).

**Table 6**: Total time ($t_{\text{total}}$) for the Tesseract-Linear case running with $p$ processors, generating $n_{\text{elem}}$ pentatopes, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface tetrahedra ($n_{\partial P}$) in the final repartitioned mesh.

| $p$ | $n_{\text{elem}}$ | $n_{\partial P}$ | $t_{\text{total}}$ | % adapt | % bal. |
|---|---|---|---|---|---|
| 1 | 7.25m | - | 2d:6h | 100.0% | - |
| 2 | 7.42m | 57.7k | 2d:1h | 98.2% | 1.4% |
| 4 | 7.37m | 111.0k | 15h:57m | 97.7% | 1.7% |
| 8 | 7.32m | 193.2k | 7h:35m | 97.4% | 1.9% |
| 16 | 6.66m | 129.0k | 2h:24m | 97.6% | 1.7% |
| 24 | 5.95m | 69.8k | 1h:21m | 98.0% | 1.3% |
| 36 | 5.63m | 285.7k | 42m:11s | 97.2% | 2.1% |

**Table 7**: Total time ($t_{\text{total}}$) for the Tesseract-Wave case running with $p$ processors, generating $n_{\text{elem}}$ pentatopes, broken into adaptation (% adapt) and interface migration (% bal.) times, with the number of interface tetrahedra ($n_{\partial P}$) in the final repartitioned mesh.

| $p$ | $n_{\text{elem}}$ | $n_{\partial P}$ | $t_{\text{total}}$ | % adapt | % bal. |
|---|---|---|---|---|---|
| 1 | 6.72m | - | 2d:17h | 100.0% | - |
| 2 | 6.99m | 100.7k | 3d:1h | 98.4% | 1.3% |
| 4 | 7.05m | 155.2k | 1d:2h | 98.3% | 1.3% |
| 8 | 6.90m | 191.3k | 8h:19m | 97.9% | 1.6% |
| 16 | 6.59m | 214.7k | 2h:59m | 97.7% | 1.7% |
| 24 | 6.19m | 184.0k | 1h:21m | 97.5% | 1.9% |
| 36 | 6.37m | 259.9k | 53m:50s | 96.7% | 2.6% |

## 5. CONCLUSIONS

We have demonstrated the implementation of a parallel four-dimensional mesh adaptation for $3d + t$ spacetime numerical simulations. The developed algorithm builds off existing $3d$ mesh adaptation algorithms in which the mesh adaptation problem is divided across a specific number of processors. Within a particular pass of an adaptation iteration, the boundaries of each

**Table 8**: Total time to adapt the mesh to 52 (Polar), 69 (Linear) and 47 (Wave) million simplices, along with metric conformity statistics on quasi-unit edge lengths and simplex quality.

| Case | iter. | $n_{\text{elem}}$ | $n_{\partial P}$ | $t_{\text{total}}$ | % adapt | % bal. | $\%\ell_{\text{unit}}$ | $\%q_{\text{unit}}$ |
|---|---|---|---|---|---|---|---|---|
| Polar (3d) | 14 | 51.70m | 703.7k | 8h:59m | 70.8% | 28.1% | 97.0% | 88.0% |
| Linear (4d) | 10 | 69.00m | 1.779m | 1d:17h | 97.7% | 1.7% | 87.0% | 45.0% |
| Wave (4d) | 9 | 47.40m | 1.173m | 17h:24m | 97.4% | 2.2% | 90.0% | 45.0% |

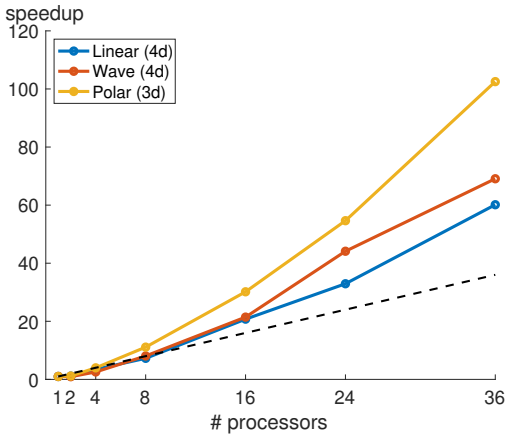



**Figure 9**: Parallel speedup when adapting the meshes for the Cube-Polar (Polar), Tesseract Linear (Linear) and Tesseract Wave (Wave) cases.

partition interface are kept fixed while still allowing the true geometric boundaries to be modified. Multiple passes of the mesh adaptation algorithm are used to migrate the previous partition interfaces to the interior of the partition so as to ensure the global mesh is adapted before returning the mesh to the solver. The algorithm was verified to be (1) scalable and (2) stable, in which (1) good speedup was observed as the number of processors was increased and (2) the produced meshes are metric-conforming for $3d$ and $4d$ problems. In the future, we will investigate the performance of the algorithm with larger numbers (hundreds or thousands) of processors.

Future work involves integrating the parallel mesh adaptation algorithm with a $3d + t$ numerical simulation tool so as to perform large-scale mesh adaptation for time-dependent partial differential equations in $3d$ [49, 50, 51, 52]. Furthermore, we hope to apply the algorithm to problems in which the domain geometry is moving in time, particularly if the domain topology is changing. A tool capable of generating an initial $4d$ mesh within such a domain would first be required.

The code associated with this work is publicly available in the LGPL-licensed software package: `https://gitlab.com/philipclaude/avro`

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1827373.

## References

[1] Yano M., Modisette J.M., Darmofal D.L. "The Importance of Mesh Adaptation for Higher-Order Discretizations of Aerodynamic Flows." *20th AIAA Computational Fluid Dynamics Conference*, 3852. Jun. 2011

[2] Slotnick J., Khodadoust A., Alonso J., Darmofal D.L., Gropp W., Lurie E., Mavriplis D.J. "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences." Tech. Rep. NASA/CR-2014-218178, 2014

[3] Bangerth W., Rannacher R. "Finite Element Approximation of the Acoustic Wave Equation: Error Control and Mesh Adaptation." *East-West Journal of Numerical Mathematics*, vol. 7, no. 4, 263–282, 1999

[4] Hartmann R. "Adaptive FE Methods for Conservation Equations." H. Freistühler, G. Warnecke, editors, *Hyperbolic Problems: Theory, Numerics, Applications*, vol. 141 of *International Series of Numerical Mathematics*, pp. 495–503. Feb. 2001

[5] Erickson J., Guoy D., Sullivan J., Üngör A. "Building Space-Time Meshes over Arbitrary Spatial Domains." *Engineering with Computers*, vol. 20, 342–353, 08 2005

[6] Behr M. "Simplex Space-Time Meshes in Finite Element Simulations." *International Journal for Numerical Methods in Fluids*, vol. 57, no. 9, 1421–1434, 7 2008

[7] Bangerth W., Geiger M., Rannacher R. "Adaptive Galerkin Finite Element Methods for the Wave Equation." *Computational Methods in Applied Mathematics*, vol. 10, no. 1, 3–48, 2010

[8] Fidkowski K.J. "Output-Based Space-Time Mesh Optimization for Unsteady Flows Using Continuous-in-Time Adjoints." *Journal of Computational Physics*, vol. 341, no. 15, 258–277, 2017

[9] Yano M. *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. PhD thesis, Massachusetts Institute of Technology, Jun. 2012

[10] Yano M., Darmofal D.L. "An Optimization-Based Framework for Anisotropic Simplex Mesh Adaptation." *Journal of Computational Physics*, vol. 231, no. 22, 7626–7649, Sep. 2012

[11] Jayasinghe S. *An Adaptive Space-Time Discontinuous Galerkin Method for Reservoir Flows*. PhD thesis, Massachusetts Institute of Technology, Jun. 2018

[12] Jayasinghe S., Darmofal D.L., Burgess N.K., Galbraith M.C., Allmaras S.R. "A Space-Time Adaptive Method for Reservoir Flows: Formulation and One-Dimensional Application." *Computational Geosciences*, vol. 22, no. 1, 107–123, Feb. 2018

[13] Caplan P.C. *Four-dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations*. PhD thesis, Massachusetts Institute of Technology, Jun. 2019

[14] Yano M., Darmofal D.L. "A Fully-Unstructured Space-time Adaptive Method for Wave Propagation." *Computer Methods in Applied Mechanics and Engineering*, 2014

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

[16] Tsolakis C., Chrisochoides N., Park M., Loseille A., Michal T. "Parallel Anisotropic Unstructured Grid Adaptation." *2019 AIAA Science and Technology Forum*, 2019–1995. 2019

[17] Michal T., Krakos J. "Anisotropic Mesh Adaptation through Edge Primitive Operations." *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 159. Jan. 2012

[18] Loseille A., Alauzet F., Menier V. "Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes." *Computer-Aided Design*, vol. 85, 53 – 67, 2017

[19] Tsolakis C., Drakopoulos F., Chrisochoides N. "Sequential Metric-Based Adaptive Mesh Generation." *2018 Modeling, Simulation, and Visualization Student Capstone Conference*. Suffolk, VA, Apr. 2018

[20] Park M.A., Darmofal D.L. "Parallel Anisotropic Tetrahedral Adaptation." *46th AIAA Aerospace Sciences Meeting and Exhibit*, 917. 2008

[21] Coupez T., Digonnet H., Ducloux R. "Parallel Meshing and Remeshing." *Applied Mathematical Modelling*, vol. 25, no. 2, 153–175, 2000

[22] Coupez T. "Génération de Maillage et Adaptation de Maillage par Optimisation Locale." *Revue Européenne des Éléments Finis*, vol. 9, no. 4, 403–423, 2000

[23] Loseille A., Löhner R. "Cavity-Based Operators for Mesh Adaptation." *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition.*, 152. 2013

[24] Caplan P.C., Haimes R., Darmofal D.L., Galbraith M.C. "Extension of Local Cavity Operators to $3d+t$ Spacetime Mesh Adaptation." *2019 AIAA Science and Technology Forum*. AIAA, 2019

[25] Caplan P.C., Haimes R., Darmofal D.L., Galbraith M.C. "Four-Dimensional Anisotropic Mesh Adaptation." *Computer-Aided Design*, vol. 129, 102915, 2020

[26] Haimes R., Dannenhoffer J. "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry." *21st AIAA Computational Fluid Dynamics Conference*, 2013–3073. 2013

[27] Haimes R., Dannenhoffer J. "EGADSlite: A Lightweight Geometry Kernel for HPC." *AIAA Aerospace Sciences Meeting*, 2018–1401. 1 2018

[28] Ibanez D., Barral N., Krakos J., Loseille A., Michal T., Park M. "First Benchmark of the Unstructured Grid Adaptation Working Group." *Procedia Engineering*, vol. 203, 154 – 166, 2017. 26th International Meshing Roundtable

[29] 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., Barral N. "Verification of Unstructured Grid Adaptation Components." , no. 2019–1723, 2019

[30] Digonnet H., Silva L., Coupez T. "Massively Parallel Computation on Anisotropic Meshes." *Proceedings of the 6th International Conference on Adaptive Modeling and Simulation*, pp. 199–211. 2013

[31] Digonnet H., Coupez T., Laure P., Silva L. "Massively Parallel Anisotropic Mesh Adaptation." *The International Journal of High Performance Computing Applications*, vol. 33, no. 1, 3–24, 2017

[32] Digonnet H., Coupez T., Laure P., Silva L. "Massively Parallel Anisotropic Mesh Adaptation." *The International Journal of High Performance Computing Applications*, vol. 33, no. 1, 3–24, 2019

[33] Lachat C., Pellegrini F., Dobrzynski C. "PaMPA: Parallel Mesh Partitioning and Adaptation." *21st International Conference on Domain Decomposition Methods (DD21)*. INRIA Rennes-Bretagne-Atlantique, Rennes, France, Jun. 2012

[34] Lachat C. *Conception et Validation d'Algorithmes de Remaillage Parallèles à Mémoire Distribuée basés sur un Remailleur Séquentiel*. PhD thesis, Université de Nice-Sophia Antipolis, Dec. 2013

[35] Lachat C., Dobrzynski C., Pellegrini F. "Parallel Mesh Adaptation using Parallel Graph Partitioning." *5th European Conference on Computational Mechanics*, vol. 3, pp. 2612–2623. CIMNE - International Center for Numerical Methods in Engineering, Jul. 2014

[36] Lachat C., Pellegrini F., Dobrzynski C., Staffelbach G. "Fast Parallel Remeshing for Accurate Large-Eddy Simulations on Very Large Meshes." Research Report RR-9133, Inria Bordeaux Sud-Ouest, Dec. 2017

[37] Dobrzynski C. "MMG3D: User Guide." Technical Report RT-0422, INRIA, Mar. 2012

[38] Loseille A., Menier V., Alauzet F. "Parallel Generation of Large-size Adapted Meshes." *Procedia Engineering*, vol. 124, 57–69, 2015. 24th International Meshing Roundtable

[39] Loseille A. "Unstructured Mesh Generation and Adaptation." *Handbook of Numerical Methods for Hyperbolic Problems*, vol. 18 of *Handbook of Numerical Analysis*, chap. 10, pp. 263 – 302. 2017

[40] Drakopoulos F., Tsolakis C., Chrisochoides N.P. "Fine-Grained Speculative Topological Transformation Scheme for Local Reconnection Methods." *AIAA Journal*, vol. 57, no. 9, 4007–4018, 2019

[41] Alauzet F., Li X., Seol E.S., Shephard M.S. "Parallel Anisotropic 3D Mesh Adaptation by Mesh Modification." *Engineering with Computers*, vol. 21, no. 3, 247–258, May 2006

[42] Schloegel K., Karypis G., Kumar V. "Parallel Static and Dynamic Multi-Constraint Graph Partitioning." *Concurrency Computation*, vol. 14, no. 3, 219–240, Mar. 2002

[43] Chevalier C., Pellegrini F. "PT-Scotch: A Tool for Efficient Parallel Graph Ordering." *Parallel Computing*, vol. 34, no. 6, 318–331, 2008. Parallel Matrix Algorithms and Applications

[44] Devine K.D., Boman E.G., Heaphy R.T., Hendrickson B.A., Teresco J.D., Faik J., Flaherty J.E., Gervasio L.G. "New Challenges in Dynamic Load Balancing." *Applied Numerical Mathematics*, vol. 52, no. 2, 133–152, 2005. ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation

[45] Lusk E., Gropp W. "The MPI Message-Passing Interface Standard: Overview and Status." J. Dongarra, G. Joubert, L. Grandinetti, J. Kowalik, editors, *High Performance Computing*, vol. 10 of *Advances in Parallel Computing*, pp. 265–269. North-Holland, 1995

[46] Tsolakis C., Chrisochoides N., Park M.A., Loseille A., Michal T. "Parallel Anisotropic Unstructured Grid Adaptation." *AIAA Journal*, vol. 0, no. 0, 1–13, 0

[47] Carson H.A. *Provably Convergent Anisotropic Output-based Adaptation for Continuous Finite Element Discretizations*. PhD thesis, Massachusetts Institute of Technology, Jun. 2020

[48] Carson H.A., Huang A.C., Galbraith M.C., Allmaras S.R., Darmofal D.L. "Anisotropic Mesh Adaptation for Continuous Finite Element Discretization through Mesh Optimization via Error Sampling and Synthesis." *Journal of Computational Physics*, vol. 420, 109620, 2020

[49] Galbraith M.C., Allmaras S.R., Darmofal D.L. "A Verification Driven Process for Rapid Development of CFD Software." *53rd AIAA Aerospace Sciences Meeting*, 2015-0818. January 2015

[50] Nishikawa H., Padway E. *An Adaptive Space-Time Edge-Based Solver for Two-Dimensional Unsteady Inviscid Flows*

[51] Frontin C.V., Walters G.S., Witherden F.D., Lee C.W., Williams D.M., Darmofal D.L. "Foundations of Space-Time Finite Element Methods: Polytopes, Interpolation, and Integration." *Applied Numerical Mathematics*, vol. 166, 92–113, 2021

[52] Williams D.M., Frontin C.V., Miller E.A., Darmofal D.L. "A family of symmetric, optimized quadrature rule for pentatopes." *Computers & Mathematics with Applications*, vol. 80, no. 5, 1405–1420, 2020