Engineering with Computers manuscript No.
(will be inserted by the editor)

Parallel anisotropic mesh adaptation with boundary layers for automated CFD simulations

Onkar Sahni · Aleksandr Ovcharenko · Kedar C. Chitale · Kenneth E. Jansen · Mark S. Shephard

Received: date / Revised version: date

Abstract This paper presents a set of parallel procedures for anisotropic mesh adaptation accounting for mixed element types used in boundary layer meshes, i.e., the presented procedures operate in parallel on distributed boundary layer meshes. The procedures accept anisotropic mesh metric field as an input for the desired mesh size field and apply local mesh modifications to adapt the mesh in order to match the specified mesh size field. The procedures fully account for the parametric geometry of curved domains and maintain the semi-structured nature of the boundary layer elements. The effectiveness of the procedures are demonstrated based on three viscous flow examples that include the ONERA M6 wing, a heat transfer manifold, and a scramjet engine.

Keywords parallel mesh adaptation; boundary layer mesh; semi-structured mesh; parallel adaptive viscous flow simulations

1 Introduction

The application of finite elements for reliable numerical simulations requires that the simulations are executed in an automated manner with explicit control of the approximations made. Since there are no *a priori* methods to control the approximation errors for complex problems, *a posteriori* methods along with adaptive discretization control must be applied [2, 62, 6, 24]. Adaptive meshing is therefore an important component for reliable simulation of complex problems, such as flow problems that develop highly anisotropic solutions which can only be located and resolved through anisotropic adaptivity (e.g., see [51, 10, 11, 49, 21, 54, 9]). Furthermore, in a number of problem cases it is desirable in specific locations to use highly anisotropic elements (e.g., with aspect ratio above 1000) that have a semi-structured nature and that this underlying structure is maintained during mesh adaptation [53, 34]. Of particular interest in this study are viscous boundary layers that form near solid surfaces in wall-bounded flows.

The two major classes of mesh adaptation techniques are adaptive re-meshing methods and methods that use local mesh modification. Re-meshing methods [51, 21, 23, 26, 19] construct the desired mesh by regenerating the entire mesh through the application of automatic mesh generation algorithms governed by specified element size and shape information while accounting for curved domains. This comes at the cost of re-meshing the entire domain along with global transfer of the solution fields to the new mesh. On the other hand, methods based on local mesh modifications retriangulate local subdomains (or cavities) until the specified mesh size field is satisfied (e.g., see [49, 7, 37]). Effectiveness of local methods depends on the richness of the underlying local mesh modification operations that are employed. Some local mesh modification methods strictly use subdivision operations which can be limited in both coarsening and anisotropy. For example, in [5, 40, 31] the coarsening or merging of child elements to recover parent elements is done by reversing

O. Sahni (corresponding author) · A. Ovcharenko · K.C. Chitale · M.S. Shephard
Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY, 12180, USA
E-mail: sahni@rpi.edu, shurik@scorec.rpi.edu, chitak2@rpi.edu, shephard@rpi.edu

K.E. Jansen
University of Colorado at Boulder, Boulder, CO 80309
E-mail: kenneth.jansen@colorado.edu

previous subdivision operations (i.e., with a derefinement step) at desired locations. Thus, in such methods coarsening cannot be applied to create elements larger than those in the initial mesh. This aspect also limits the amount of anisotropy that can be achieved in elements. Similarly, some local mesh modification schemes only adapt to the faceted geometry (i.e., based on the initial mesh) and do not improve the geometric approximation of curved domains as the mesh is refined. In contrast, other research work has shown that a richer set of local mesh modification operations [49, 7, 37, 29, 20] can be utilized to support general (local) coarsening, reconnection and anisotropy in the mesh as well as to account for curved domains [38]. These local operations also support localized transfer of solution fields [5, 44] at the cavity level as the mesh is incrementally modified to attain the adapted mesh.

In wall-bounded flows with boundary layers, hybrid or semi-structured mesh generation methods have been extensively used [52, 15, 43, 25, 41, 22, 33, 8, 28, 42, 27]. For such problems, local mesh modification operations have been extended to account for mixed topology elements [53, 34, 30], wherein the semi-structured nature of the mesh is taken into consideration such that the layers or stacks of wedges (triangular prisms) or hexes are modified in order to attain the desired local mesh resolution while the overall layered structure is maintained. In [34] subdivision of mixed elements is employed along with mesh movement to improve geometric approximations for curved domains and in [30] derefinement is also followed for transient problems, whereas in [53] a more richer set of local mesh modification operations are utilized for mixed element meshes. In these studies, layered mesh is modified in conjunction with the rest of the interior mesh consisting of unstructured tetrahedral elements, where pyramids are used when necessary to transition between semi-structured and unstructured portions of the mesh. These studies have focused on serial boundary layer meshes, i.e., where the mesh is not partitioned or distributed.

Mesh adaptation techniques must operate in parallel on distributed meshes because most problems of interest involve complicated geometries and complex physics that even with adaptivity the resulting meshes are very large. Adaptive simulations for such problems, where only the analysis or solve step is parallelized (see, for example, [65]), turn out to be limited in problem size and/or time-to-solution due to the serial adaptation step that may take as much or even more time (compared to the analysis step). Thus, to efficiently execute parallel adaptive simulations, both the analysis and mesh adaptation steps must be parallelized and run on distributed or partitioned meshes (e.g., see [13, 59, 66]).

Performing mesh adaptation in parallel requires that all mesh operations are carried out in such a way that the resulting distributed mesh properly fits together (at inter-part boundaries). Subdivision or refinement operations can be understood at the level of single element and therefore, can be performed in parallel on each processor including lower-order mesh entities that reside on inter-part boundaries. This must be followed by a communication round between processors to update the inter-part links based on new mesh entities introduced at inter-part boundaries (e.g., see [16, 47]). In [47, 50] parallel refinement and derefinement is used for unsteady problems, where child elements of a given parent element reside on the same processor. This makes the merging of child elements straightforward and communication is required to delete the necessary vertices at inter-part boundaries due to derefinement. As in the serial case, this parallel approach is limited in terms of the amount of coarsening and anisotropy that can be achieved in elements. In contrast to a parallel scheme based on refinement and derefinement, parallel re-meshing is used in [26]. In this study marked elements (based on an adaptation criterion) are removed leading to holes in the distributed mesh. This mesh is repartitioned with the constraint that every hole to be re-meshed resides on a single processor in the re-distributed mesh. This scheme is more flexible in terms of shape and orientation of resulting elements, however, the overall process can be time consuming. For example, due to global repartition of the mesh or when the resulting hole to be re-meshed is relatively large, due to concentrated adaptation in a contiguous portion of the domain, leading to significant work and memory imbalances between processors. On the other hand, in an adaptation approach that is based on local mesh modifications only small portions of the mesh are affected at any given time. Therefore, mesh operations for which the related cavity resides solely on one part can be carried out in parallel (similar to the serial case) while a mesh migration is needed for operations in which the cavity spans multiple parts. A naive sequence of steps, that intermingle on-part mesh modification and mesh migration at a low level, will be ineffective due to significant wait times. However, with a proper control of on-part mesh modification and mesh migration steps, parallel mesh adaptation based on local mesh modifications has been shown to be efficient [16, 3]. These parallel mesh adaptation methods focus on general unstructured meshes and do not take boundary layer meshes into consideration.

A parallel mesh adaptation scheme for distributed boundary layer meshes has been presented in [32], where refinement and derefinement is employed for mixed elements. parallel refinement and derefinement is implemented on mixed element meshes. Similar to refinement and derefinement of fully unstructured/tetrahedral distributed meshes discussed above [47, 50], the technique in [32] requires the child elements of a given parent element to reside on the same processor such that mesh derefinement is completed with minimal communication. As mentioned before, such a scheme limits the amount of coarsening and anisotropy that can be achieved for hybrid meshes. Whereas an approach based on a richer set of local mesh modification operations for distributed boundary layer meshes can overcome these limitations but to the best of our knowledge there has been no study on such an approach. The current work presents such an approach which is based on parallelization of a richer set of local mesh modification operations for distributed boundary layer meshes (i.e., this paper generalizes and extends the operations presented for serial boundary layer meshes in [53]).

The organization of the paper is as follows. Section 2 briefly provides the terminology used for boundary layer meshes. Section 3 discusses the local mesh modification operators that are used for the layered portion of the mesh and its interface with the rest of the interior mesh. Section 4 describes the current procedures for parallel implementation of local mesh modification operations for distributed boundary layer meshes. Section 5 demonstrates the effectiveness of the current procedures based on three viscous flow problems.

1.1 Nomenclature

$\{M^d\}$	the set of topological mesh entities of dimension d. $d = 0$: vertex, $d = 1$: edge, $d = 2$: face, $d = 3$: region.
M_i^d	the ith mesh entity of dimension d.
$\{\partial M_i^d\}$	the entities on the boundary or closure of M_i^d.
$\{M_i^d\{M^D\}\}$	the set of mesh entities of order D adjacent to M_i^d.

2 Boundary layer mesh terminology

A common method to construct boundary layer meshes with semi-structured portions of regions in the domain is an advancing layers method [52, 15, 43, 25, 41, 22, 33, 8, 28, 42, 27]. It inflates the unstructured surface mesh on no-slip walls, where the boundary layers form, into the volume along the local surface normals in stack of

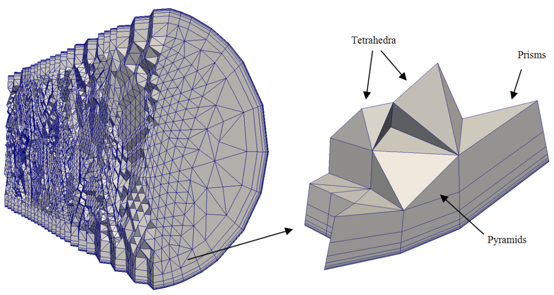


Fig. 1 Cut of the mesh with boundary layers on the pipe geometry.

layers of elements in a graded fashion up to a specified distance. Rest of the domain is filled with unstructured tetrahedral and pyramidal elements. The example of a simple mesh with boundary layers defined on pipe geometry is presented in Figure 1. In addition to the boundary layer prisms and interior tetrahedra, this example includes a few pyramids. Pyramids are used to transition to the unstructured tetrahedral mesh when quadrilateral faces of the layered mesh are exposed, for example, when the number of prisms in neighboring stacks change due to difference in number of layers.

The boundary layer mesh contains a structure that can be decomposed into a product of a layer surface (2D) and a thickness (1D) [53]. The mesh composed of triangles located at the top of each layer is referred to as layer surface, while the lines orthogonal to the wall composed of edges are called growth curves as shown in Figure 2. The edges that belong to layer surfaces are referred to as layer edges and ones that reside on growth curves are called growth edges as depicted in the figure. Each layer of elements is formed with the help of layer surfaces above and below with growth edges in between. The height of each layer is referred to as layer thickness whereas the collective height of all layers as adds up to total thickness of a boundary layer. The number of total vertices (or edges) on growth curves determine its level. The vertices on walls from which growth curves originate are referred to as originating vertices. The top most layer of the stack of boundary layer elements shares an interface with the unstructured volume mesh. The interior tetrahedral or prismatic elements, sharing lower-order mesh entities with layered portion of the mesh, are referred to as interface elements.

3 Mesh modifications in the layered portion of the mesh

To maintain the boundary layer stack, the mesh modification operations are decomposed such that those

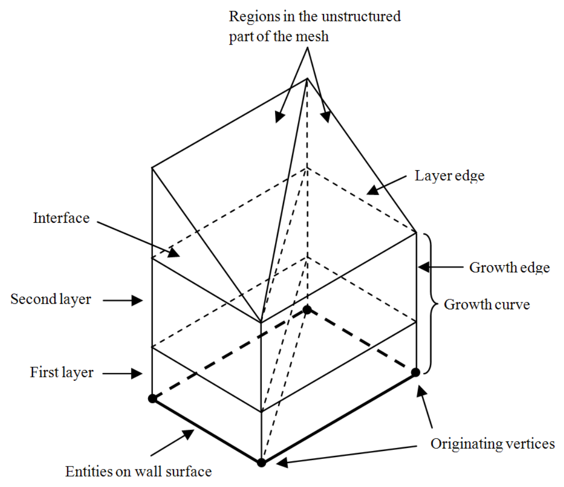


Fig. 2 Decomposition of a boundary layer stack.

that affect the "surface triangulation" of the boundary layer are done consistently through the stack. It is also necessary to apply the corresponding unstructured mesh modification at the top of the stack. This section describes the management of the modification of the boundary layer stack including consideration of the transition into the unstructured mesh at the top of the stack.

3.1 Mesh metric tensor

A mesh metric field is used to specify the anisotropic mesh sizes over the problem domain (e.g., see [51, 49, 37]). In an adaptive process, the error estimator or indicator information is used to specify the desired size or mesh-metric field. This specification at any given point P is done by a symmetric definite positive tensor $T(P)$, referred to as mesh metric tensor, such that the desired directional mesh resolution at this point follows an ellipsoidal surface. That is this tensor transforms an ellipsoid into a unit sphere. The transformation $e^T Te = 1$, where e denotes the edge vector, defines a mapping of the edge in the physical space into a unit edge in the metric space. Any tetrahedron that perfectly satisfies the mesh metric field should be a unit equilateral tetrahedron in the metric space as depicted in Figure 3. However, in an unstructured mesh it is often not possible to satisfy this exactly. Therefore, mesh modification algorithms constrain edge lengths in the metric space to be within an interval $[L_{min}, L_{max}]$ (e.g., $[1/\sqrt{2},\sqrt{2}]$), and the mean ratio of elements in the metric space to be close to 1 (with 1 being the ideal value), for example, see [49, 37].

In the layered part of the mesh, the mesh metric tensor can be decomposed into (2D) ellipse as the planar part along a layer surface, which dictates local in-plane mesh resolution, and (1D) normal component which controls local layer thickness [53]. Note that layer thickness can also be based on flow physics, for example, in turbulent boundary layer flows [14]. Figure 4 illustrates the decomposition of mesh metric field in layered part of the mesh.

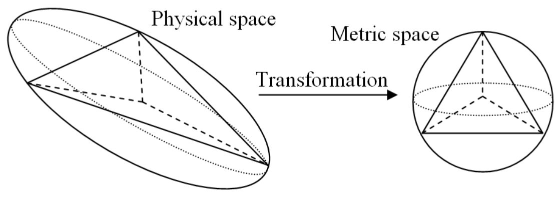


Fig. 3 Transformation of a tetrahedral element based on a mesh metric tensor.

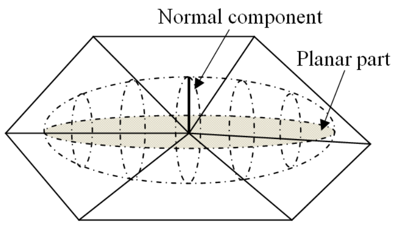


Fig. 4 Decomposition of mesh metric tensor in layered part of the mesh.

3.2 Local mesh modification cavity

With the input of mesh metric field, the local mesh modifications are carried out to adapt the mesh in order to match the specified size field. As discussed before, a richer set of local mesh modification operations [49, 7, 37] is needed for fully unstructured anisotropic meshes. Each modification operation involves a local cavity or subdomain which is retriangulated. The cavity for a given operation is defined as the union of sets of mesh entities that are changed by the application of the modification operation with the restriction that the triangulation of the cavity's boundary remains unchanged. This means that the cavity's boundary allows to share unchanged mesh entities with the other unaffected portions of the mesh.

In 3D, the cavity is defined as the set of mesh regions along with its closure (i.e., lower-order mesh entities), which will be modified by the modification associated with entity M_k^d and is denoted as:

$$\{C(M_k^d)\} = \{M_i^3 \bigcup \{\partial M_i^3\} \,|M_i^3 \text{ is affected by mesh modification operation applied to} M_k^d\}. \quad (1)$$

The cavity boundary then can be defined as:

$$\{\partial C(M_k^d)\} = \{M_j^\alpha, \alpha = 0,1,2 | M_j^\alpha \in \{\partial M_i^3\} \\ \forall M_i^3 \notin \{C(M_k^d)\}\}. \tag{2}$$

Eq. 2 states that the cavity boundary contains lower-order entities M_j^α located on the boundary of regions comprising cavity $\{C(M_k^d)\}$. This way the cavity boundary is shared with adjacent mesh regions that are not part of the cavity.

The application of a local mesh modification operation then is a retriangulation of the cavity in which the mesh topology changes, $\{C(M_k^d)\}$, into a set of mesh entities contained in the set $\{S\}$ with the following conditions:

$$\{S\} \neq \{C(M_k^d)\}\,, \tag{3}$$

$$\{\partial S\} = \{\partial C(M_k^d)\}\,. \tag{4}$$

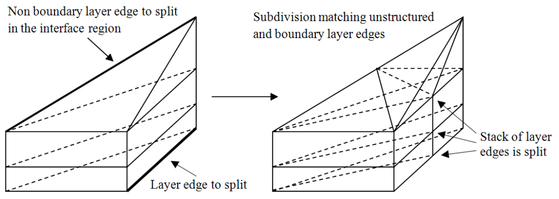
There can be situations when an entity M_k^d which requests a mesh modification operation might be moved in the boundaries of a 3D subdomain with no change in local mesh topology (for example, vertex motion operation considered in Section 3.3). In this scenario, Eq. 3 would have equality.

3.3 Boundary layer stack modification

To preserve the layered nature of the boundary layer the "surface triangulation" mesh adaptation utilizes layer-edge split, collapse and swap operations [53], while layer thicknesses adjustment and growth curve vertices movement to the geometrical model boundaries are accomplished through vertex repositioning operations.

The layer edge split operation splits edges in the boundary layer stack and applies the appropriate subdivisions at the interface in the unstructured mesh. When edge split is requested for a single layer edge, all edges in the stack are subdivided. This scheme is conservative in nature in that it may provide a finer mesh than desired for some layer surfaces. Namely, if M_I^1, where $I \in [1..N]$, is the layer edge to be split in the stack of N layer edges, then the cavity associated with it consists of a set of unique regions $\{C\{M_i^1\}\} = \left\{\bigcup_{i=1}^N \{M_i^1\{M^3\}\}\right\}$. Figure 5 illustrates the layer edge split operation. The subdivision of pyramids and tetrahedra at the interface follows the stack split.

The layer edge collapse operation is performed on stacks that contain short edges in the local mesh metric in a manner that avoids oscillation between collapse and split operations [53]. The edge collapse operations can only be applied when the affected unstructured mesh entities at the top of the stack also remain valid after the collapse. If $\{(M_i^1, M_i^0)\}$ are the


Non boundary layer edge to split in the interface region
Layer edge to split
Subdivision matching unstructured and boundary layer edges
Stack of layer edges is split
Fig. 5 Example of boundary layer edge split operation.

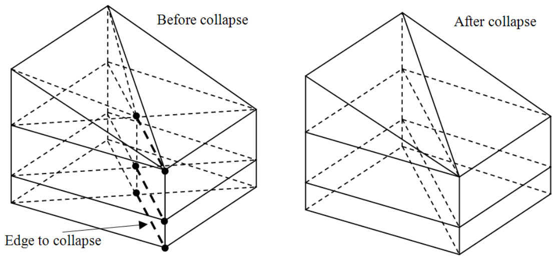


Fig. 6 Example of boundary layer edge collapse operation.

pairs of stacks of N layer edges to be collapsed and their corresponding vertices being deleted in the stack, then the cavity associated with the collapse operator is $\{C\{M_i^0\}\} = \left\{\bigcup_{i=1}^N \{M_i^0\{M^3\}\}\right\}$ with deletion of a set of regions $\left\{\bigcup_{i=1}^N \{M_i^1\{M^3\}\}\right\}$. Figure 6 shows local mesh cavity before and after the layer edge collapse operation.

The layer edge swap operation changes the connectivity of neighboring boundary layer stacks. In comparison with tetrahedra which are reconfigured based on the equatorial plane, there is only one other possible configuration in case of layer edge swap for layer faces [53]. If $\{M_i^1\}$ are the layer edges to be swapped, then the layer edge swap operation retriangulates the cavity $\{C\{M_i^1\}\} = \left\{\bigcup_{i=1}^N \{M_i^1\{M^3\}\}\right\}$ with new layer edges being introduced inside the cavity and deletion of $\{M_i^1\}$. The interior regions at the interface adjacent to the top most layer edge being swapped are retriangulated based on the equatorial plane and are not guaranteed to have a valid configuration. Thus, appropriate checks are required in order to figure out if the layer edge swap operation results in a valid mesh after its completion. Figure 7 gives an example of layer edge swap operation.

When edge splits are applied to boundary layer edges on curved wall surfaces, the newly introduced vertices must be moved to the curved boundary to maintain the proper geometric approximation. All the vertices in the growth curve should gradually move following the correspondent originating vertices with the help of a movement vector [53]. The movement vector is determined based on the originating vertex target location as: $\mathbf{m} = \mathbf{v_0^t} - \mathbf{v_0}$, where $\mathbf{v}_0^t$ is the target location of orig-

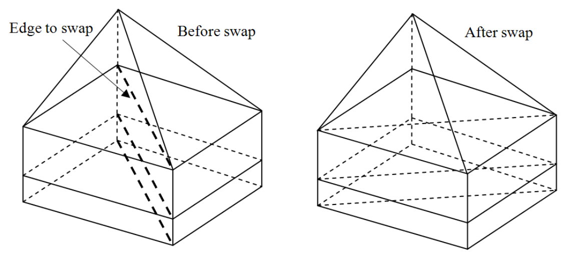


Fig. 7 Example of boundary layer edge swap operation.

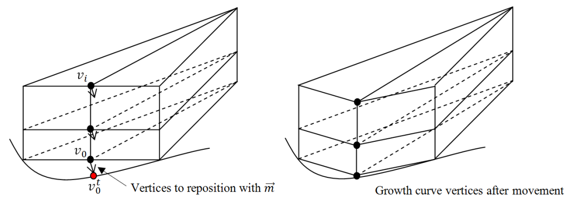

v_i
v_0
v_0^t
Vertices to reposition with $\vec{m}$
Growth curve vertices after movement

Fig. 8 Repositioning of boundary layer vertices due to movement of newly created originating vertex to the domain boundary.

inating vertex on the curved boundary and $\mathbf{v}_0$ is the original location of the originating vertex. The movement vector $\mathbf{m}$ is then applied to all the vertices on that growth curve. The procedure first evaluates target locations for vertices on all the growth curves, with each vertex's target location calculated as: $\mathbf{v}_i^t = \mathbf{v}_i + \mathbf{m}$, where $\mathbf{v}_i$ is the current i-th vertex location on a growth curve corresponding to its originating vertex location $\mathbf{v}_0$. It then moves vertices to their computed target locations as depicted in Figure 8. Similar to unstructured vertex projection to the curved boundary, layer vertex movement through repositioning is not always possible without additional mesh modification, especially for the top most vertices, as it may introduce inverted elements. In this case local mesh modification operations are applied to the interior volume mesh to allow the successful repositioning of top-most vertex which is followed by the rest of vertices on the growth curve.

3.4 Subdivision of transition pyramids

For pyramids we consider more subdivision templates than those presented in our previous work [53]. This results in more flexibility in parallel mesh adaptation of boundary layer meshes. Pyramids are subdivided in the unstructured part of the refinement procedure since they have to account for both splits of layer edges performed during boundary layer refinement and interior edge subdivisions applied for tetrahedra refinement templates.

There are three ways the quadrilateral face of a pyramid capping a boundary layer edge can be subdivided as depicted in Figure 9. While refinement in a layer direction splits the quadrilateral face using layer edges only (left of Figure 9), the request to change the resolution along the thickness direction or number of layers can be achieved with bisection of the quadrilateral face using growth edges (middle of Figure 9). Additionally, subdivision can be performed in both directions with split of both layer and growth edges (right of Figure 9).

In the current study templates subdividing growth edges are not exploited. The reason for this is not due to any limitation in the ability to split the elements, but rather because thickness adjustment based on vertex repositioning was found to be sufficient for the current problem cases.

The rest of triangular faces of a pyramid are subdivided by counting the number of edges tagged for refinement as shown in Figure 10. The triangular face templates are the same for the subdivision of layer faces in a boundary layer and interior faces adjacent to unstructured regions, and thus do not introduce additional ambiguities associated with triangular face split. Ultimately, there are eleven possible combinations of edges in a pyramid that can be tagged for refinement. Depending on how edges are marked and which diagonal edges are chosen for a face with two tagged edges [37], there are a total of twenty-five subdivision templates. The most frequently used templates for splitting pyramids are presented in Figure 11.

3.5 Unstructured decomposition of boundary layers

Figure 12 shows a close-up of an adapted mesh on a pipe geometry without decomposed entities on the left and on the right with prisms divided into tetrahedra and pyramids. One can observe that on the left part of the figure there are boundary layer regions with low aspect ratio. In order to satisfy the desired mesh resolution in such cases the corresponding portion of the top of the boundary layer is converted into unstructured part of the mesh such that more flexible interior mesh modifications can be applied to achieve appropriate level of mesh anisotropy. This is an additional consideration than previous work [53].

The process of reducing the numbers of layers in selected stacks, referred to as trimming or unstructured decomposition of boundary layers, leads to introduction of additional quadrilateral faces that must be capped with pyramids so they can be matched to the tetrahedra used in the unstructured part. This step can be reversed when a thicker boundary layer is desired. It can be done by splitting growth edges that will reintroduce more layers (locally). However, as mentioned before, split of

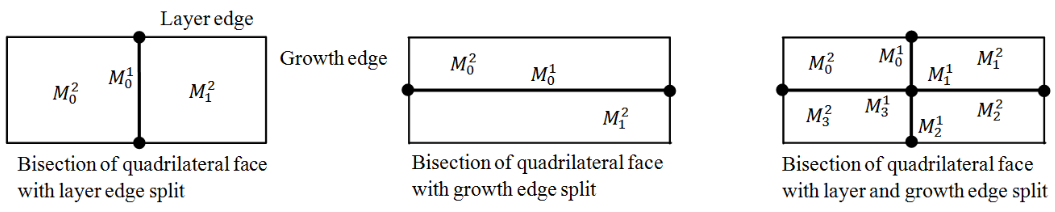

Layer edge
M_0^2
M_0^1
M_1^2
Bisection of quadrilateral face with layer edge split
Growth edge
M_0^2
M_0^1
M_1^2
Bisection of quadrilateral face with growth edge split
M_0^2
M_0^1
M_1^1
M_1^2
M_3^2
M_3^1
M_2^1
M_2^2
Bisection of quadrilateral face with layer and growth edge split

Fig. 9 Subdivision of quadrilateral face accounting for the edge split requests.

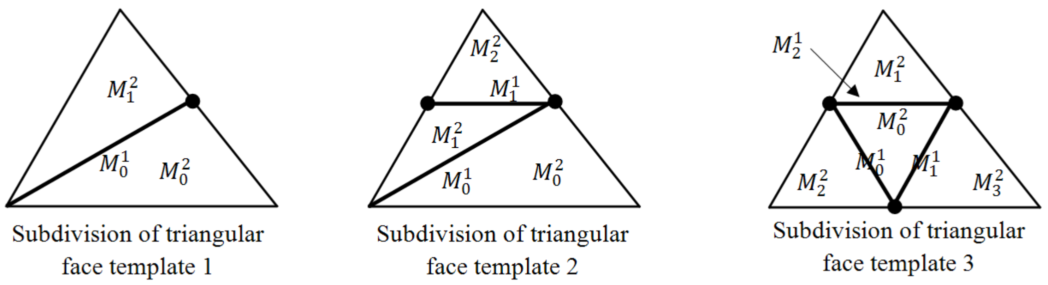

M_1^2
M_0^1
M_0^2
Subdivision of triangular face template 1
M_2^2
M_1^1
M_1^2
M_0^1
M_0^2
Subdivision of triangular face template 2
M_2^1
M_1^2
M_0^2
M_0^1
M_1^1
M_2^2
M_3^2
Subdivision of triangular face template 3

Fig. 10 Subdivision templates for triangular face.

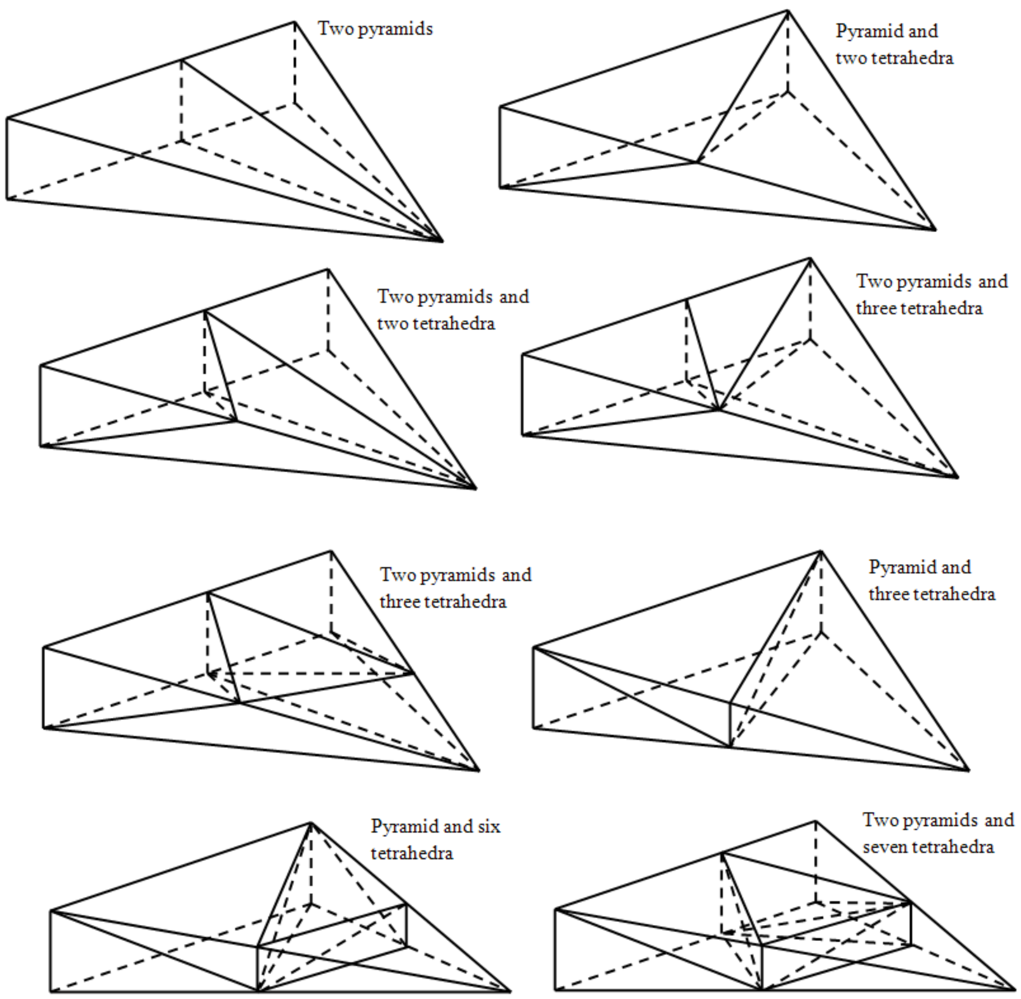

Two pyramids
Pyramid and two tetrahedra
Two pyramids and two tetrahedra
Two pyramids and three tetrahedra
Two pyramids and three tetrahedra
Pyramid and three tetrahedra
Pyramid and six tetrahedra
Two pyramids and seven tetrahedra

Fig. 11 Subdivision templates for pyramids.

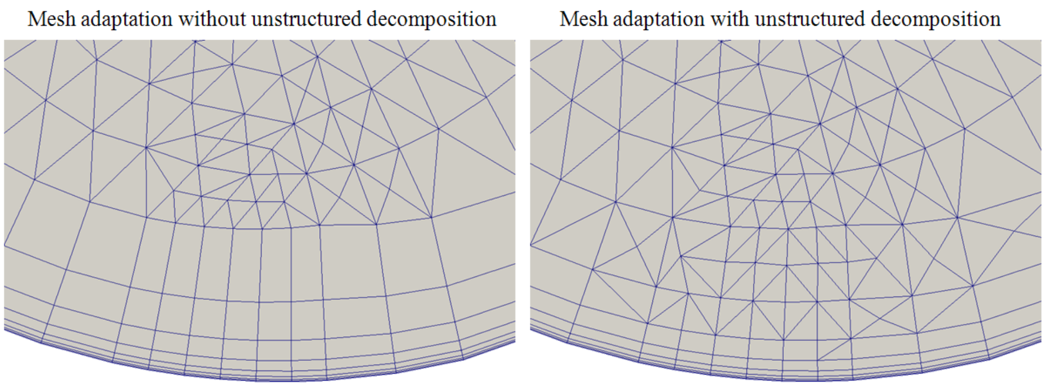

Mesh adaptation without unstructured decomposition
Mesh adaptation with unstructured decomposition

Fig. 12 Examples of mesh adaptation without and with unstructured decomposition of low-aspect ratio regions.

growth edges were not need for currently considered problem cases.

The stack decomposition algorithm relies on the following criterion that adjacent stacks sharing a mesh face must differ in number of layers only by 1, in which case connected prism into two tetrahedra and a pyramid; the latter is used to transition between different number of layers. This condition can be satisfied by controlling the number of vertices in neighboring growth curves in a preprocessing step before executing the trimming step.

Figure 13 illustrates boundary layer stacks with different number of layers in face-neighbor stacks and have no more than one layer difference. Note that with this condition the corresponding number of vertices between growth curves of a given stack can vary by as much as two. In the trimming step, the number of layers for a given growth curve is dictated by the lowest level regions being decomposed. The lowest level boundary layer vertices are given a priority to bisect quadrilateral faces adjacent to it. If vertices next to each other are of the same level, the priority is granted to one having the smallest local vertex identifier (ID) which eliminates any possible ambiguity in selecting diagonal edge for face subdivision.

The application of this restriction for trimming of stacks yields a situation in which all prisms that need to be subdivided into tetrahedra can be triangulated with templates that do not require introduction of an interior vertex [60]. Avoiding the need to insert an interior vertex eliminates specific algorithmic complexities and typically results in better element shape quality control [35].

Algorithm 1 Pseudo code for the adaptive unstructured decomposition procedure.

```
1:  for each boundary layer $BL_i$ do
2:      for each region in the stack $M_j^3 \in BL_i$ from bottom to top do
3:          assign: $ar_j$ = aspect ratio of quad face $M_k^2 \in \{\partial M_j^3\}$
4:          if $ar_j < \alpha$ (where $\alpha$ is a prescribed aspect ratio value) then
5:              consider $BL_i$ for the decomposition
6:              assign boundary layer vertex level $M_b^0 \in \{\partial M_j^3\}$ to the corresponding originating vertex
7:              break
8:          end if
9:      end for
10: end for
11: for each $level$ starting from 0 do
12:     if neighboring growth curves' number of vertices is more than 1 after decomposition then
13:         reduce the number of vertices in the neighboring growth curve to $level + 1$
14:     end if
15: end for
16: bisect quad faces of each $BL_i$ tagged for the decomposition
17: subdivide regions adjacent to bisected faces
18: tetrahedralize pyramids sharing quad face
```

A prism can be subdivided into a pyramid and a tetrahedron if two of its quadrilateral faces are split and bisecting edges share a common vertex, or into three tetrahedra if all quadrilateral faces are subdivided and there is a common vertex between introduced diagonal edges as depicted in Figure 14. The decomposition procedure logic eliminates the possibility of having situations when only one quadrilateral face is bisected or there is no common vertex connecting diagonals in a prism being split.

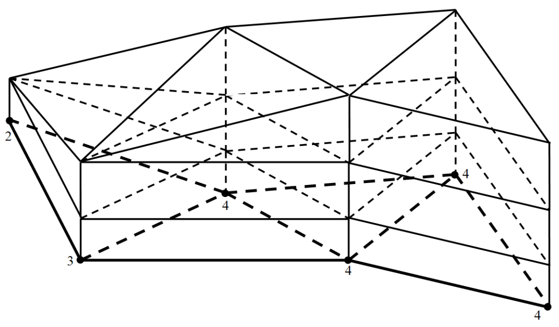
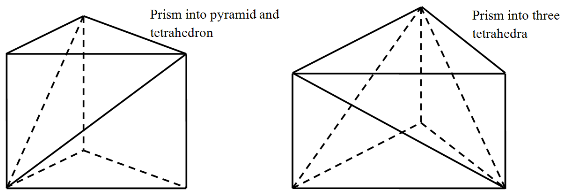


Fig. 13 Different number of layers in neighboring boundary layer stacks and the corresponding number of (indicated by counts shown at originating vertices). Bold lines represent edges on the wall surface.


Prism into pyramid and tetrahedron
Prism into three tetrahedra
Fig. 14 Subdivision of a prism with diagonal edges bisecting quadrilateral faces having a common vertex.

The algorithm for unstructured decomposition of portions of a boundary layer mesh consists of three parts. The first is responsible for getting the initial number of layers for each growth curve having its adjacent boundary layers decomposed. The second adjusts the number of layers for growth curves such that they have difference of no more than one vertex after the decomposition is accomplished. The third assigns each quadrilateral face being bisected with the appropriate vertex originating a subdivision using the rules described above. Algorithm 1 presents the overall approach of the unstructured decomposition.

3.6 Overall boundary layer mesh adaptation algorithm

Before we present our approach for parallel boundary layer mesh adaptation, the overall adaptation procedure is described. It is executed in three stages: mesh coarsening, iterative mesh refinement, and shape correction [53]. The first two stages are controlled by mesh edge length analysis in the transformed space, whereas the third stage is dominated by both element quality and mesh edge length control. The boundary layer part of the mesh has priority in applying mesh modification operations of a specific type followed by the same operation for the unstructured entities. This is done because size requests for entities in boundary layer stacks are more involved and thus, resolved first.

The coarsening stage applies local mesh modification operations to eliminate the majority of edges shorter that requested by the local mesh size field. A mesh edge is considered to be short if its length in the transformed space is smaller than the specified value L_{min} [37]. An advantage to coarsening first is that it will make the traversals required during mesh adaptation faster and limit the peak memory used to that needed by the final adapted mesh. Thickness adjustment is applied on the coarsened mesh so that it is only applied to entities that will remain in the mesh.

Algorithm 2 Pseudo code for mesh adaptation with boundary layers using the mesh metric field.

```
1: get mesh metric field from the application
2: apply metric decomposition over layered part of mesh
3: coarsen short layer edges in metric space
4: coarsen short interior edges in metric space
5: adjust thickness for each growth curve by vertex movement
6: while mesh resolution is not satisfied for [$L_{min}, L_{max}$] do
7:     tag longer layer edges in metric space
8:     split tagged layer edges, their adjacent faces and regions except for the interior ones
9:     tag longer interior edges in metric space
10:    split tagged interior edges, their adjacent faces and regions
11:    move new boundary layer vertices at curved walls onto solid surface
12:    move new vertices at curved walls onto solid surface for the unstructured part of the mesh
13:    coarsen short layer edges in metric space
14:    apply unstructured decomposition to parts of boundary layers with lower aspect ratio
15:    coarsen short interior edges in metric space
16: end while
17: eliminate poorly shaped layer faces by layer edge swaps or layer edge collapses
18: eliminate sliver interior regions by shape control
```

The second stage refines mesh regions using refinement templates that split the mesh edges longer than L_{max} in the transformed space. L_{min} and L_{max} are typically selected to be $1/\sqrt{2}$ and $\sqrt{2}$ [37], respectively. The procedure ensures that the refinement is applied to stack of prisms along with interior elements located at the interface. This stage also places newly created boundary vertices onto the domain boundary (e.g., as defined by the CAD model). It also coarsens any new short mesh edges introduced by refinement templates. At the end of an iteration of this step any elements at the top of boundary layers, that have their aspect ratios reduced to a certain value (typically aspect ratio below 1), are tetrahedralized and made part of the interior unstructured mesh. Thus removing them from the boundary layer stack by unstructured decomposition.

The third stage applies shape improvement operations, which improve the quality of poorly shaped entities in the transformed space. Those entities are modified using sets of swap and compound operators [53, 37, 35] to obtain the best possible element quality while preserving the correct edge length in the metric space [17, 39, 4]. Again, the shape correction operations are carried out in the boundary layer part first with the following mesh optimization in the interior part.

4 Parallel implementation

The execution of parallel mesh adaptation is based on the fact that the mesh is distributed [3, 57] into a number of parts, where each part consists of set of adjacent mesh entities. Each part is treated as a serial mesh with the addition of mesh part boundaries, that are managed by the parallel mesh database that describe the groups of mesh entities residing on inter-part boundaries.

The application of a local mesh modification will involve a cavity of mesh entities that are either all on one part, or are on multiple parts. In the case where the cavity is on a single part, the mesh modification can be carried out on that part, thus making parallel execution of the adaptation process trivial. In those cases where the entities for the cavity being modified are distributed on multiple parts, some form of coordinated inter-processor communication is needed. The approach used in this work follows the one presented in reference [53] with specific extensions introduced to effectively parallelize boundary layer adaptation.

4.1 Distributed mesh infrastructure

The effective implementation of parallel mesh modification requires a parallel mesh infrastructure and associated parallel mesh control tools. The parallel mesh representation used here [57] maintains the information on the mesh entities on the part boundaries such that all parts sharing those entities can obtain information about those entities on other parts. It supports the ability to update mesh entities on those part boundaries if they are modified (e.g. an edge split). It also supports the movement of mesh entities from one part to another (referred to as migration) with the inter-part boundaries being automatically updated.

Since the mesh modifications to boundary layer stacks are always done to the entire stack, maintaining the knowledge of the stack is critical. Managing this information would be difficult if the mesh regions in a stack were distributed over multiple processors. Thus the implementation of parallel boundary layer mesh adaptation requires the mesh regions in a boundary layer stack are placed in a mesh set [64] where each such mesh set is required to always be on a single part with the ability to migrate that set and its entities together to another part. Since the adaptive mesh modification process will alter the numbers of mesh regions on parts, it is necessary to dynamically repartition the mesh at times to help control parallel operations and memory usage, and to improve scalability. The Zoltan library [56] is used to perform the dynamic load balancing. Since the mesh modification operations involve irregularly structured messages of a small size, parallel efficiency and scalability depends on effectively controlling the underlying message-passing processes. The Inter-Processor Communication Manager (IPComMan) is used [48] for efficient parallel communications between processors. It is a general-purpose communication package built on top of MPI [1] which significantly improves inter-processor communications independently of the architecture and underlying network being used.

The direct consideration of cavities on the part boundary for such mesh modification operations as collapse and swap is a complex and expensive procedure since it leads to a number of communication steps to properly update the parts with the mesh modification decisions carried out. Thus, regions from such cavities are localized on one processor such that the cavity retriangulation is applied in serial. To support cavity localization, a mesh migration is used, where all regions and stacks of regions involved in the mesh modification operation are migrated onto a single part [3, 57].

4.2 Refinement and vertex reposition to geometrical boundaries

Subdivision of mesh edges and their adjacent mesh faces on part boundaries happens the same way it is done in serial [16, 3] since replicated faces across part boundaries have their bounding edges and vertices in the same order to ensure the triangulations are consistent across face neighbors (see Figures 9 and 10). Note that triangular faces can be split using any combination of edges tagged for refinement, whereas quadrilateral faces as part of the boundary layer stack always have opposite edges split so they can be easily bisected. When quadrilateral face bisection is performed during the decomposition of boundary layer prisms into unstructured mesh entities (see Figure 14), the procedures ensure that the diagonal bisecting the face is created in the same way on both parts sharing the face such that there is no mesh triangulation invalidity introduced during the matching set up of the newly created entities on the part boundary.

The inter-part links between newly created mesh entities are updated across the part boundary in a communication step after refinement such that the distributed mesh is correctly connected across the inter-part boundary. In the execution of that operation the corresponding old-to-new entity mapping is formed such that the newly created entity links are effectively set during the communication step. After all tagged edges and faces have been split, the communication round for subdivided entities on the part boundary is performed to create inter-part links between new mesh entities by matching locally mapped old-new entity relationships with what is received from other parts corresponding to the specific old entity. The mesh regions were subdivided using the same templates as in serial without any inter-part communication since they are not part of an inter-processor boundary. The pseudo code of the parallel refinement algorithm is given in Algorithm 3.

Algorithm 3 Pseudo code for the parallel mesh refinement procedure.

```
1: for all edges $M_i^1$ (and faces $M_i^2$) which are tagged for refinement do
2:   split $M_i^1(M_i^2)$ as depicted in Figure 15
3:   if the entity being split is on part boundary then
4:     attach the list of newly created entities $NewEntList$ to corresponding $M_i^1(M_i^2)$
5:     add $M_i^1(M_i^2)$ to the list of entities for linkage update $UpdateLinksList$
6:   end if
7: end for
8: for all $M_i^1(M_i^2)$ from $UpdateLinksList$ do
9:   send the attached $NewEntList$ to remote copies of $M_i^1(M_i^2)$
10:  when received, set up links between $NewEntList$ of $M_i^1(M_i^2)$ on local part with ones from remote copies
11: end for
12: subdivide boundary layer stacks and mesh regions as in serial
```

The algorithm involves the same logic of updating inter-part links for both boundary layer and unstructured parts of the mesh. The only difference for the boundary layer procedure is that once the stack of quadrilateral faces exposed to the part boundary is split, it has to be completely updated with the corresponding one on the part sharing common faces.

Algorithm 4 Pseudo code for the parallel boundary layer vertex repositioning procedure.

```
1: for all originating vertices $M_i^0$ with model boundary target location $T_{M_i^0}(x,y,z)$ do
2:   calculate target locations for each vertex in growth curve $GC_j$ where $M_i^0 \in GC_j$
3:   add top-most vertex $M_{i_{top}}^0 \in GC_j$ with its corresponding $T_{M_{i_{top}}^0}(x,y,z)$ to $VtxListToSnap$
4: end for
5: for all $M_i^0 \in VtxListToSnap$ do
6:   move $M_i^0$ to its target location
7:   if movement introduces flat or inverted regions then
8:     if $M_i^0$ is on part boundary then
9:       add $M_i^0$ to $VtxAdjRgnMigrate$
10:    else
11:      apply mesh modification procedures to insure the movement of $M_i^0$ to its target location
12:    end if
13:  else
14:    remove $M_i^0$ from $VtxListToSnap$
15:  end if
16: end for
17: send each vertex $M_i^0 \in VtxAdjRgnMigrate$ to its remote copies
18: vertices which received messages from remote copies in $VtxAdjRgnMigrate$ move to their original location
19: perform mesh migration originated by $VtxAdjRgnMigrate$
20: update $VtxListToSnap$ on each part in terms of vertices migrated
21: repeat 5-20 until $VtxListToSnap$ is empty on all parts
22: for all growth curves $GC_j$ where top-most vertex repositioning was performed do
23:   move each vertex in the stack to its calculated position
24: end for
```

Figure 15 demonstrates an example of the parallel refinement procedure. The initial distributed mesh is depicted in Figure 15(a), where thick lines indicate edges and their bounding faces which are going to be split and communicated during the refinement step. One of those edges is represented as M_0^1 on P_0 and M_1^1 on P_1 (Figure 15(a)). Consider a face on view (Figures 15(b) and 15(c)) of the stack of boundary layer edges originating from M_0^1 and M_1^1 respectively, and

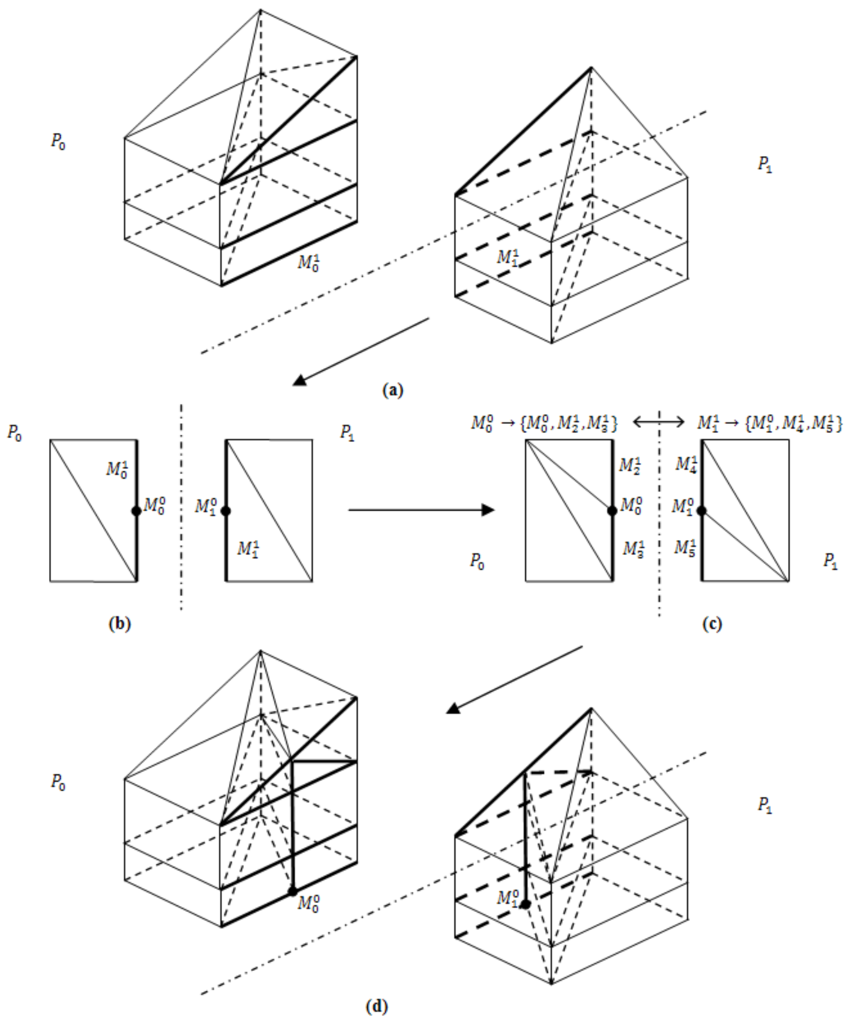


Fig. 15 Example of distributed mesh refinement step.

one edge from the interface region on top of each boundary layer. Figure 15(b) shows the introduction of the vertex M_0^0 splitting the edge M_0^1, and M_1^0 splitting the edge M_1^1. The newly created vertex and two edges are the child entities and attached to the corresponding edge being split on the part boundary, namely: $M_0^1 \to \{M_0^0, M_2^1, M_3^1\}$ on P_0 and $M_1^1 \to \{M_1^0, M_4^1, M_5^1\}$ on P_1. In order to set up the correct links between the new entities, a communication round is carried out to create remote copies between $\{M_0^0, M_2^1, M_3^1\}$ on P_0 and $\{M_1^0, M_4^1, M_5^1\}$ on P_1. M_0^1 is sent to P_1 with the list of attached entity addresses, whereas M_1^1 is sent to P_0. When P_1 receives the message with M_0^1 and its list of children and their addresses, it goes to M_1^1 and with the ordered local old-new mapping updates the links such that M_0^0 corresponds to M_1^0, M_2^1 corresponds to M_4^1, and M_3^1 corresponds to M_5^1. Similarly the remote copies are set up for P_0 receiving the message with M_1^1 from P_1 as depicted on Figure 15(c). After all edges and faces are split on the part boundary and communication links are updated, the regions are subdivided in serial and no further communication is needed. The resulting mesh is depicted in Figure 15(d).

Once refinement is completed, each part holds a list of mesh vertices that are classified on curved geometries and need to be projected onto the model boundaries [3]. For the boundary layer part of the mesh, the newly created originating vertices are projected onto to the solid model surfaces with the help of the movement vector as described in Section 3.3. In cases where the projection alone will introduce mesh invalidities in the unstructured mesh at the top of the stack, a more extensive sets of local mesh modification operations are required that include collapses and/or splits [38] and their parallel execution [16]. Algorithm 4 describes parallel boundary layer vertex movement routine.

4.3 Coarsening and swapping

Layer edge collapse operation is always performed on the on-part localized cavity [16, 3]. The boundary layer stacks adjacent to the same growth curve starting with a vertex M_d^0 and ending with top most growth curve vertex $M_{d_{top}}^0$, are checked for the layer edge collapse operation with corresponding layer edges adjacent to growth curve vertices being considered . If neither of stacks of layer edges shorter than the desired size in a metric space can be collapsed locally, the boundary layer coarsening procedure migrates all the boundary layer stacks and interface regions adjacent to the growth curve vertices from M_d^0 to $M_{d_{top}}^0$ to one part and checks for the possibility of layer edge collapse operation with the full local cavity.

Figure 16 shows the example of the edge collapse operation requesting migration to be accomplished. It can be seen from the figure that boundary layer vertices $M_d^0..M_{d_{top}}^0$ for surrounding boundary layers are located on the part boundary and layer edge collapse for its adjacent edges cannot be evaluated. Thus, they request all the adjacent boundary layer stacks and interface regions to be migrated to one part P_2 in order to perform layer edge collapse operation locally.

Algorithm 5 Pseudo code for the parallel boundary layer coarsening procedure.

```
 1: for all originating vertices $M_d^0$ from the list of vertices $VtxList$ do
 2:     check the stack of shortest adjacent edges corresponding to vertices $M_d^0$ in growth curve $GC_j$
 3:     if boundary layer stacks and interface regions associated with operation are on one part then
 4:         apply layer edge collapse operation and update $VtxList$
 5:     else
 6:         add vertices on part boundary $\{M_{d_i}^0..M_{d_{top}}^0\} \in GC_j$ into the list $VtxToMigrate$
 7:     end if
 8: end for
 9: perform mesh migration using the list $VtxToMigrate$
10: update $VtxList$ in order to be able to traverse newly migrated vertices
11: repeat 1-10 until $VtxLst$ is empty
```

Algorithm 5 presents the pseudo code for the parallel boundary layer coarsening. Given a dynamic list of originating vertices to apply layer edge collapse with, it is repeatedly traversed until the list is empty. To be able to reduce the number of total mesh migrations, prior to each traversal, a list of corresponding top-most vertices is initialized on each part to keep track of vertices on part boundaries which are responsible to request a migration of boundary layer stacks and mesh regions. During traversal, the boundary layer stacks are checked for layer edge collapse operation using the current vertex and adjacent shortest edges. If all boundary layer stacks and interface regions are on one part, the operation is applied as in serial and the local dynamic list is updated. Otherwise, corresponding boundary layer vertices on part boundaries are added to the list of vertices initiating mesh migration. When all parts are done traversing their dynamic vertex lists, requests to migrate boundary layer stacks and mesh regions are made using the list of vertices on part boundaries. These requests then drive the application of mesh migration, and at the same time updating the dynamic vertex list of corresponding originating vertices on each part in terms of which of them need to be checked for the layer edge collapse operation locally.

Algorithm 6 Pseudo code for the parallel boundary layer surface optimization procedure.

```
 1: for all stacks starting from zero-level faces $M_i^2$ from the list of zero-layer faces $FaceLst$ do
 2:     check the best mesh modification operation to be applied to surrounding boundary layers
 3:     if boundary layer stacks and interface regions associated with operation are on one part then
 4:         apply local mesh modification operation and update $FaceLst$
 5:     else
 6:         consider $M_{k_{top}}^2$, a top-most face of the boundary layer stack growing from $M_i^2$
 7:         add top-most vertices $M_{j_{top}}^0 \in \{\partial M_{k_{top}}^2\}$ into the list $VtxToMigrate$
 8:     end if
 9: end for
10: perform mesh migration using the list $VtxToMigrate$
11: update $FaceList$ in order to be able to traverse newly migrated zero-level layer faces
12: repeat 1-10 until $FaceList$ is empty
```

Parallelization of swap operations follows the same overall logic as the collapse operation. In the case of swaps within the boundary layer they are done at the end of the process as part of a mesh optimization process to improve the quality of layer element shapes [53]. The only difference in this optimization step is driven by a traversal process focused on improving the shape of

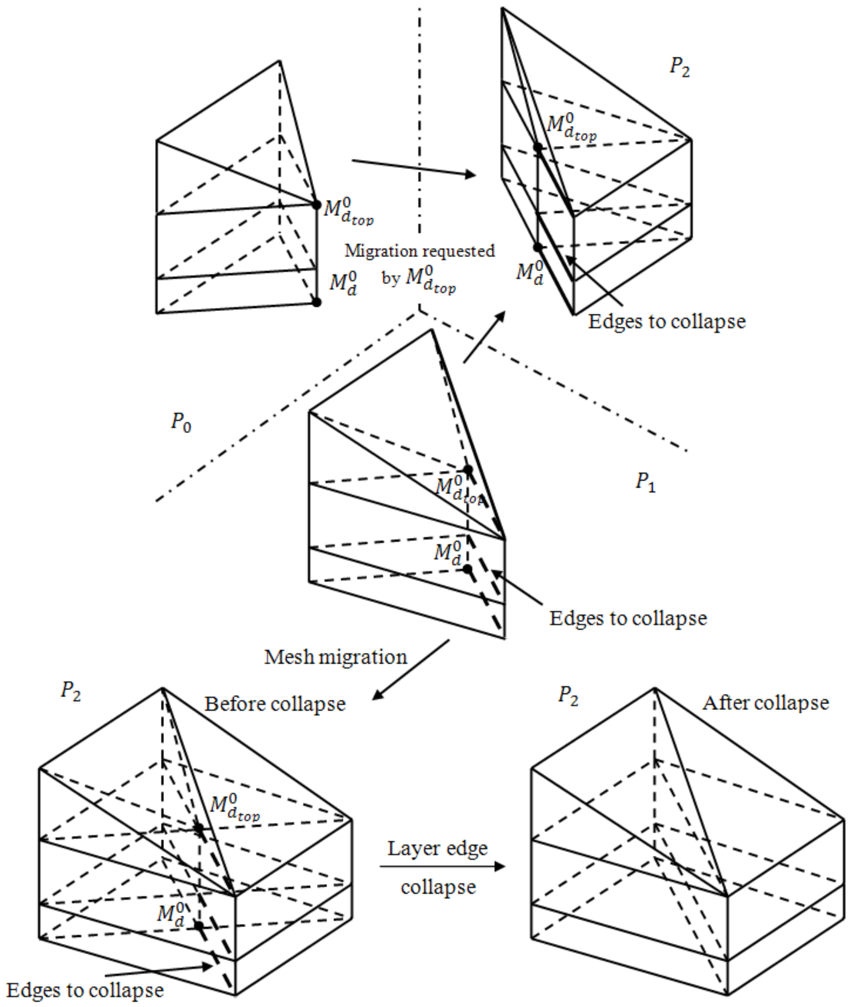


Fig. 16 Example of parallel layer edge collapse operation involving mesh migration.

elements in the boundary layer. Algorithm 6 describes the parallel approach of surface optimization routines.

5 Application results

5.1 Adaptive loops and applications

An adaptive loop is created using a set of interoperable components that link analysis codes with geometry-based problem definitions, automatic mesh generation, error estimation procedures and generalized mesh modification procedures (e.g., see [58, 12]). The adaptive loop connects the analysis and adaptation components needed for the successful simulation on the problem domain. The solution obtained by the analysis routines is evaluated and used to provide information to adapt the mesh, which in turn enriches the solution approximation. The error distribution is determined on the current mesh and is converted to the mesh metric size field which is adapted and then sent back to the solver. During the mesh adaptation execution the necessary solution fields are transferred in order to be a correct input for the flow solver on the next analysis step. The overall structure of the adaptive loop procedure is shown on Figure 17.

The capabilities of the parallel anisotropic mesh adaptation with boundary layers are demonstrated with three flow applications. The first case involves the ONERA M6 wing [61] for which FUN3D solver [46] was used. In the second case, the simulation of a heat transfer manifold was executed. The CFD analysis for this test was performed using the PHASTA Navier-Stokes solver [63]. The third test case involves a scramjet engine (of NASA CIAM configuration [45]), where the analysis was performed using the FUN3D solver.

These studies have been executed on Hopper Cray XE6 [18] at National Energy Research Scientific Computing Center. It is configured with 2 twelve-core AMD 2.1 GHz processors per node, with separate L3 caches

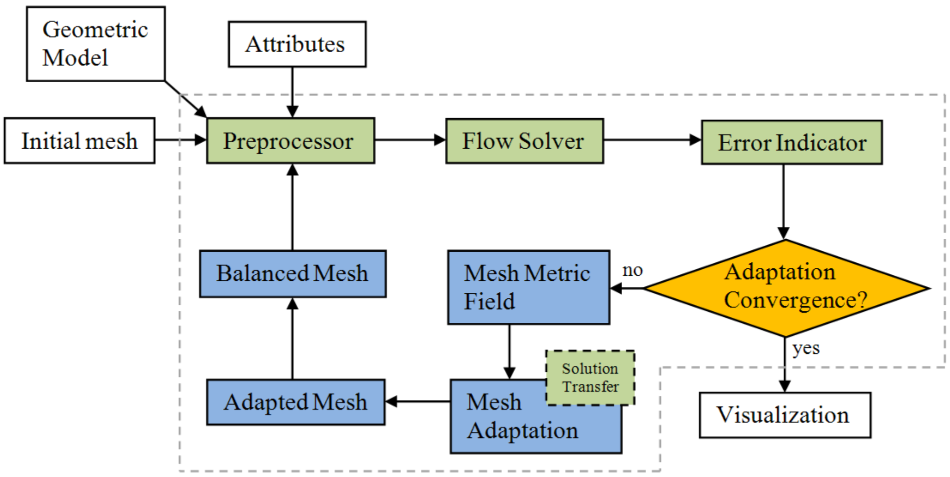


Fig. 17 A schematic of the adaptive loop procedure.

and memory controllers, 32 GB or 64 GB DDR3 SDRAM per node. Hopper has a Gemini interconnect with a 3D torus topology.

5.2 ONERA M6 wing

The ONERA M6 wing is one of the classic CFD validation cases [61]. Air enters the wind tunnel at transonic speed and is accelerated over the wing to supersonic speeds causing a shock to appear on the upper surface of the wing. The free stream Mach number is 0.84, and the angle of attack is 3.06°. The free stream pressure and temperature are 42.89 *psi* and 255.5 *K*. The Reynolds number is 11.72 million based on the mean aerodynamic chord. This flow marks a strong need for adaptive grids due to unknown shock location *a priori* and complex structure of the lambda shock. The reference experimental data is from Schmitt and Charpin in 1979 [61]. We used FUN3D flow solver for this case.

Three cycles of mesh adaptation were applied for this case, where Hessian of pressure was used to compute the mesh metric field. Initial mesh contained 0.28M regions (where M denotes a million). The first adapted mesh had 0.37M regions, the second adapted mesh had 1.24M regions while the third and finest adapted mesh had 3.8M regions. Figure 18 presents surface mesh for the initial and three adapt meshes. The imprint of lambda shock on the adapted mesh can be clearly seen.

Figure 19 presents the pressure coefficient for the initial and three adapted meshes. The surface pressure contours along with surface meshes in Figure 18 show that the mesh is refined in the shock region and the lambda shape of the shock is clearly captured. The mesh away from the shock is coarsened, due to low values of pressure gradients in this region. The surface pressure contours become sharper and more regular with adaptivity. One thing to notice is that the elements start to align with the shock in the first adapted mesh.

In order to perform a more quantitative comparison, we look at pressure coefficient profiles along the chord at certain spanwise locations on the wing. Figure 20 shows pressure coefficient along the local chord at two spanwise locations. In this figure, experimental data is also included [61]. These plots show that as the mesh is adapted the pressure coefficient is more accurate. To establish this aspect further we look at a zoomed view near the suction peak in Figure 21. The zoomed view clearly shows that agreement between experimental and numerical results are improved as the mesh is adapted further, with finest or third adapted mesh showing the best agreement among all meshes. For example, at non-dimensional span location of $y/b = 0.9$ the peak pressure value is captured far better on the finest adapted mesh as compared to other adapted meshes. Results on the initial mesh are least accurate among all meshes.

To evaluate the parallel performance for boundary mesh adaptivity, a strong-scaling study was conducted. In this study we consider a finer mesh of the adapted mesh resulting in 160M regions. Strong scaling study was executed on processor cores ranging from 512 to 8192. The scaling is based on the execution time of 512 processors and defined as following:

$$Scaling = (n_{proc-base} * time_{base})/(n_{proc-test} * time_{test}), \tag{5}$$

where $n_{proc-base}$ is the number of base processors, $time_{base}$ is the execution time on a base number of processors, $n_{proc-test}$ is the number of following test processors, and $time_{test}$ is the execution time on that number of

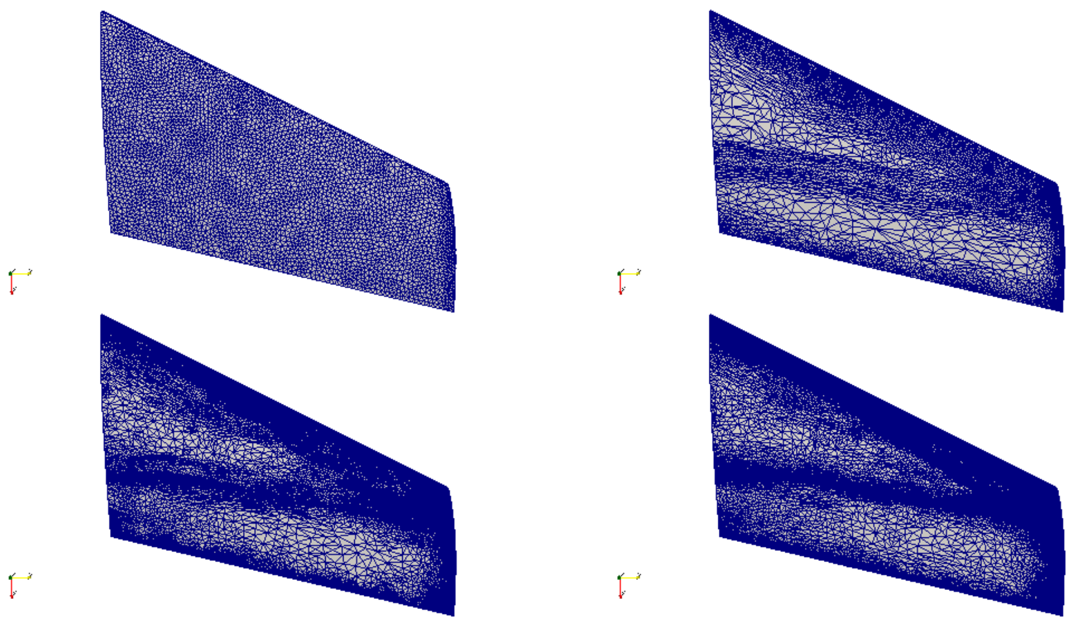


Fig. 18 ONERA M6 wing: initial and three adapted meshes.

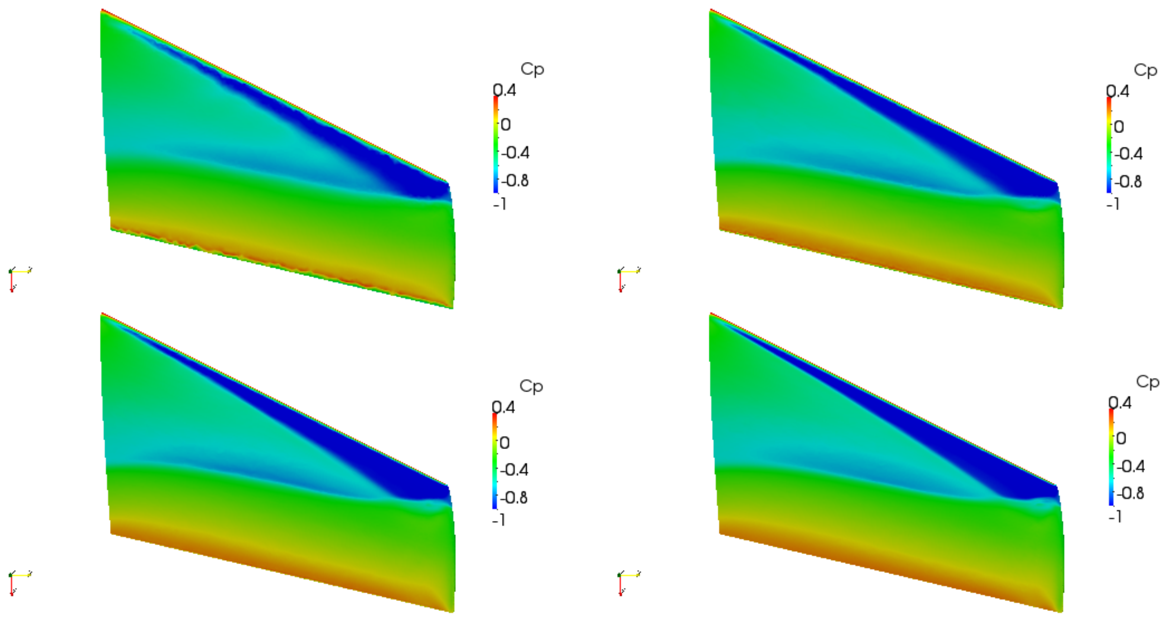


Fig. 19 ONERA M6 wing: pressure on initial and three adapted meshes.

processors. All available cores per node were requested during the adaptation runs. As indicated in Table 1 the mesh adaptation times decrease with the increased number of cores. As the given mesh is distributed to more processors, there is little computation performed during mesh modification operations relative to the substantial increase in communications, and the scaling decreases on high core counts. Note that Table 1 demonstrates a strong scaling study where the problem size is fixed.

Table 1 Mesh adaptation run times and strong scaling for the ONERA M6 case.

N/proc	512	1024	2048	4096	8192
Time	1212.83	812.68	507.36	322.82	241.94
Scaling	1	0.75	0.60	0.47	0.31

5.3 Heat transfer manifold

The heat transfer manifold test case consists of a large diameter cylindrical pipe as the inlet, a relatively thin and flat manifold section, and twenty outlet pipes. Flow

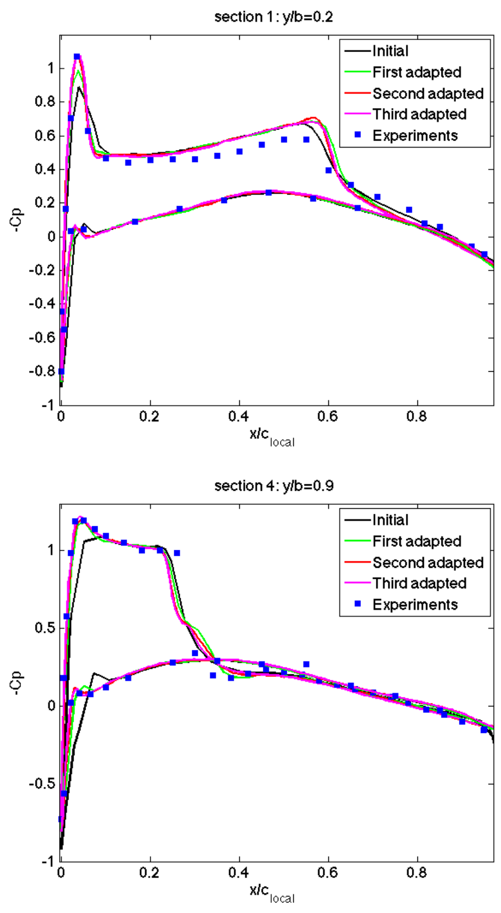


Fig. 20 Pressure coefficient profiles along the local chord on initial and three adapted meshes for two spanwise locations.

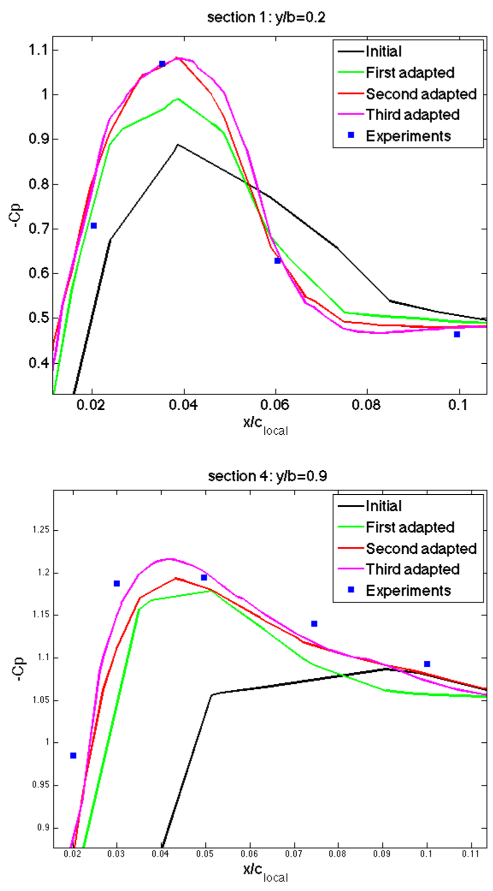


Fig. 21 Zoom of pressure coefficient profiles, near suction peak, on initial and three adapted meshes for two spanwise locations.

simulations for this case were done using steady, incompressible RANS with the Spalart-Allmaras turbulence model applied. A turbulent velocity profile with a Reynolds number of 1 million was used at the inlet pipe. No-slip boundary conditions were assumed at walls and a natural pressure of zero was prescribed at the outlet. The solution parameter used in the Hessian-based error indicator is the static pressure combined with the scaled dynamic pressure defined as $P + 0.5\alpha\rho u^2$, where the scaling factor $\alpha = 0.2$ was chosen so that an appropriate balance of the static and dynamic pressure was considered. Eigenvalues were computed from the Hessian matrix and were scaled appropriately to get the mesh sizes along with three orthogonal directions.

The adaptive loop (which consists of a flow solve and mesh adaptation within each cycle) was carried out twice, and at each cycle, flow solver was started from the previous loop's solution. Each cycle was divided into 1000 time steps with a constant time step size of 0.1s. The initial computation used a mesh of 3M elements with pre-defined boundary layers. The first adapted mesh has 16M regions, and the second adapted boundary layer mesh results in 81M regions. The initial, first and second adapted meshes are depicted in Figure 22.

For this case we provide a qualitative assessment of numerical results due to the lack of experimental or any reference data for comparison. The pressure distribution near the inlet pipe is provided in Figure 23, whereas the outlet pipe cuts are presented in Figure 24. The initial mesh is too coarse and these figures demonstrate its inability to capture the flow features. Critical flow regions including stagnation points and fillets of the pipes get significantly refined, as reflected by smoother solution results. The walls of the manifold, especially the wall closest to the inflow pipe, get refined to a higher degree. The fillets of outlet pipes also get more refinement. The central part of the manifold gets relatively lesser refinement because of relatively less gradation in solution fields. Moreover, away from flow regions with stagnation and turns, highly anisotropic mesh elements are created to effectively model the flow anisotropy, result-

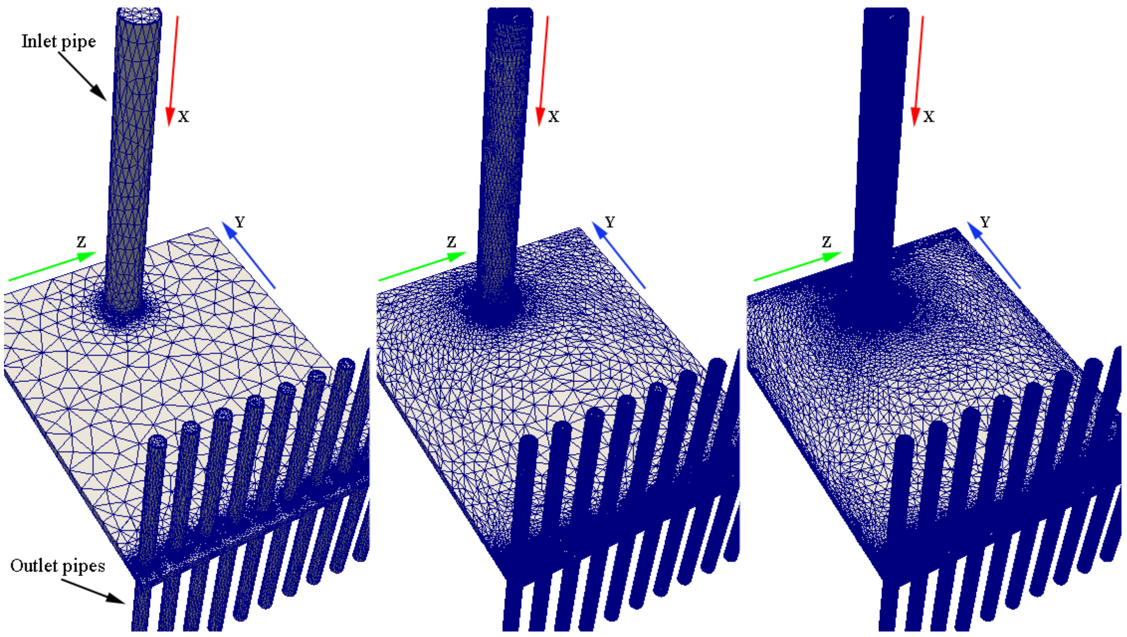


Fig. 22 Heat transfer manifold: initial (left), first adapted (middle) and second adapted (right) meshes.

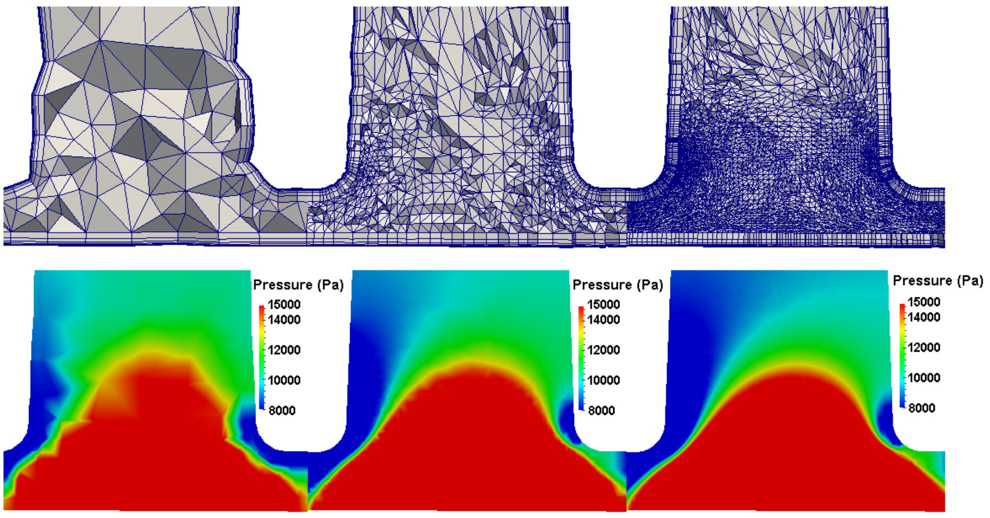


Fig. 23 Initial (left), first (middle) adapted and second (right) adapted meshes (top row) and pressure distribution (bottom row) for the heat transfer manifold test case. The cut is applied to the inflow pipe and the manifold.

ing in significant computational savings over isotropic meshes of equivalent solution quality.

Figure 25 shows the magnitude of wall shear stress on the surface of the manifold geometry. With adaptivity, smoothness of the field improves and features are captured in a better way. It has been shown in [53] that wall shear stress computed by adapted boundary layer meshes is superior to fully unstructured adapted meshes.

In addition to these results, mesh statistics were also collected for this case. Specifically, this was done for three quantities in the metric or transformed space: layer edge length, interior edge length and mean ratio for interior regions (for such mesh statistics of fully unstructured, anisotropically adapted meshes see [49, 37]). These statistics were collected for the final adaptation cycle, where it was done for both the input mesh and the resulting adapted mesh, with respect to the specified mesh metric field. Figure 26 shows these statistics. It can be seen that the input mesh has large number of interior and layer edges whose lengths in the metric space are outside the desired interval of $[1/\sqrt{2}, \sqrt{2}]$, however, for the adapted mesh interior edges fall in this interval indicating the satisfaction of the specified metric field. Note that a large number of layer edges in the adapted mesh have length (in the metric space) close to

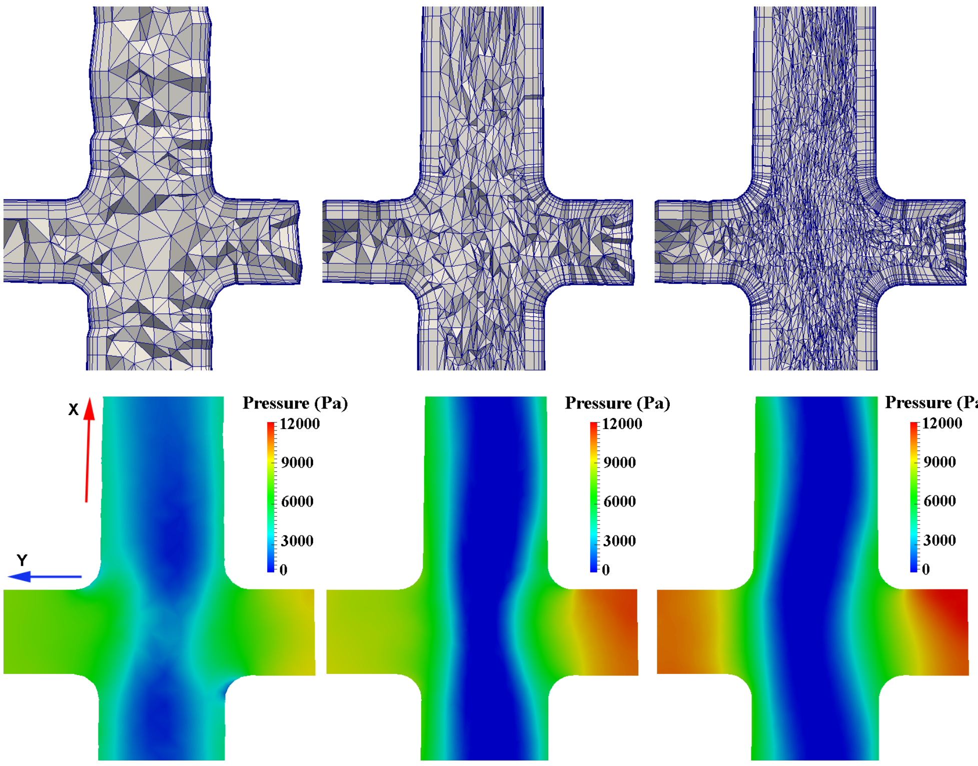


Fig. 24 Initial (left), first (middle) and second (right) adapted meshes (top row) and pressure distribution (bottom row) for the cut of an outflow pipe and the manifold in the heat transfer manifold test case.

0.5. This is due to the conservative nature of the split scheme for layer edges (i.e., edge split for a single layer results in split of all layer edges in the stack). Mean ratio plot also shows that the shape of the elements in the adapted mesh respects the specified mesh metric field.

To evaluate the parallel performance for boundary mesh adaptivity, as before, a strong-scaling study was conducted, where mesh adaptation in the second cycle of the adaptive loop was executed on a range of processors: 256 to 4096 cores. Table 2 gives the scaling of second-cycle mesh adaptation run times with the initial mesh of 16M regions, and the final one consisting of 81M regions. In Table 2 the mesh adaptation times decrease with the increased number of cores. As before, in this set of strong scaling runs with given mesh on high core counts there is little computation performed during mesh modification operations relative to the substantial increase in communications, and the scaling decreases on high core counts.

The solver has been shown to strongly scale [66, 55] on a large nu mber of cores, i.e., for a fixed amount of work. It is the analysis part of a simulation which defines the number of processors on which the particular problem is being executed. If possible, it makes sense to run mesh adaptation routines on the same number of cores since bringing the mesh and size field to a smaller number of cores and repartition it again to a bigger number after mesh adaptation is done introduces a substantial amount of additional work and expensive data movement.

Table 2 Mesh adaptation run times and strong scaling for the heat transfer manifold case.

N/proc	256	512	1024	2048	4096
Time	1194.34	785.44	514.45	421.09	339.38
Scaling	1	0.76	0.58	0.36	0.22

The work required for mesh adaptation took 0.7% of the total simulation time on 256 processors and 3.2% of the simulation time on 4096 processors. In either case, this cost would not dominate the time spent for the analysis routines. Thus, even with the loss of strong scaling in the mesh adaptation step, it is reasonable to run it on the same number of processors together with the flow solver.

A weak scaling study was performed for this case on three unadapted meshes starting with a first mesh on 256 processors. The second mesh was obtained by uniform refinement of the first and the third by uniform refinement of the second. The mesh metric field for the second and third meshes was constructed from that on the first unadapted mesh by scaling by 1/2 and 1/4 respectively. Because of the existence of the boundary layers in the mesh the second and third unadapted

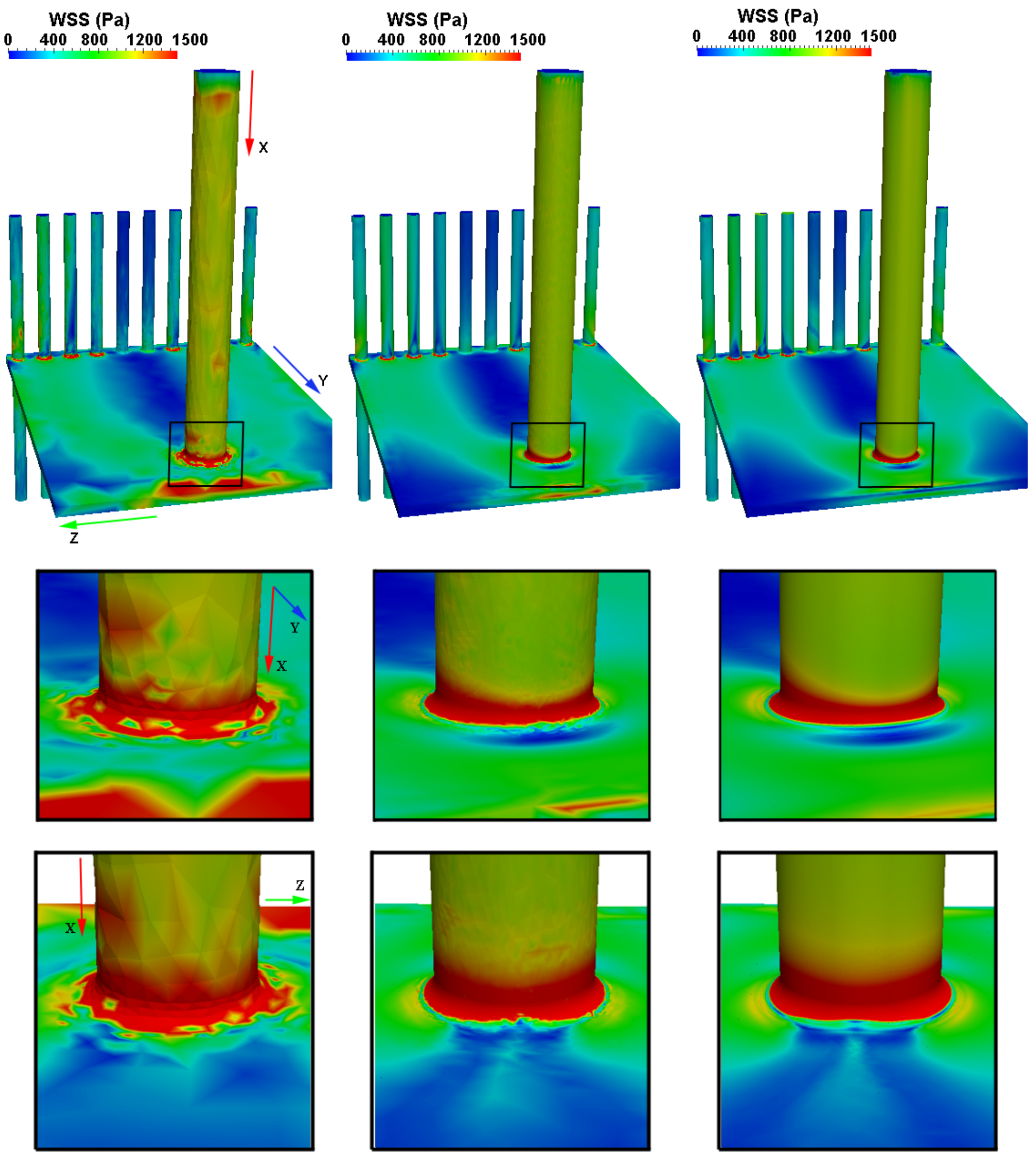


Fig. 25 Wall shear stress of the initial (left), first (middle) and second (right) adapted meshes.

meshes had roughly six times more regions than the first and second meshes respectively. Thus the numbers of processors used to adapt the second and third meshes was set to 1536 and 9216. Because of the heuristics of mesh adaptation methods and the fact that the next meshes are not precisely six times bigger than the previous mesh, the scaling is calculated using a correction factor as follows:

$$Af = M_{incr-test}/M_{incr-base}, \tag{6}$$

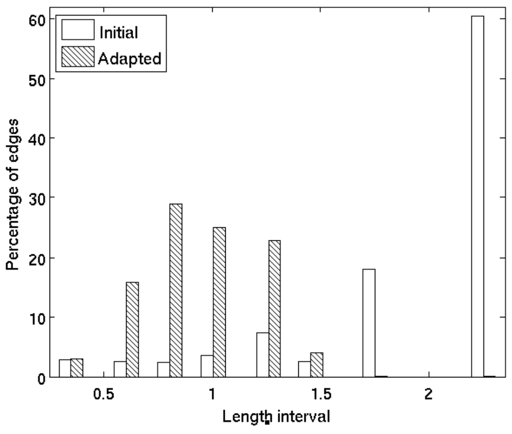
$$Scaling = Af * time_{base}/time_{test}, \tag{7}$$

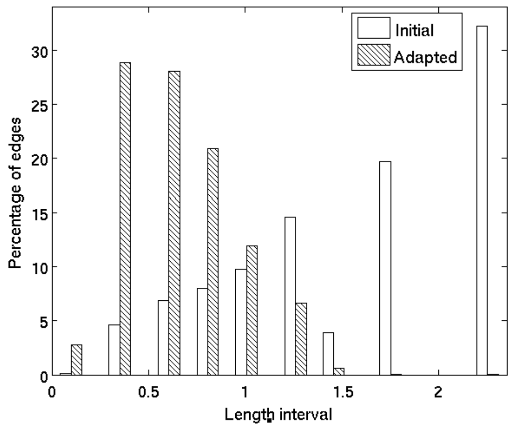
where $M_{incr-test}$ and $M_{incr-base}$ are mesh increase factors for the number of regions from unadapted to adapted meshes respectively, and Af is a factor which defines the difference between number of regions for the adapted test mesh and the adapted base mesh. The weak scaling results are presented in Table 3.

It can be inferred from Table 3 that the scaling does not degrade substantially with the increase in the number of processors while having relatively the same amount of workload in each test run. The scaling is affected not only by the growing data exchange between processors for larger processor counts, but also by the asynchronous application of mesh modification operations. Note that although the average workload is approximately the same per processor, it might be very different for each specific part depending on the amount of different types of mesh modification operations ap-

Table 3 Mesh adaptation run times and weak scaling for the heat transfer manifold case.

Number of processors	256	1536	9216
Initial mesh number of regions	16,319,606	94,487,988	604,853,414
Adapted mesh number of regions	80,890,803	527,501,893	3,702,376,095
Mesh increase factor (M_{incr})	4.97	5.58	6.12
Time	1194.34	1381.55	1716.23
Scaling	1	0.97	0.86





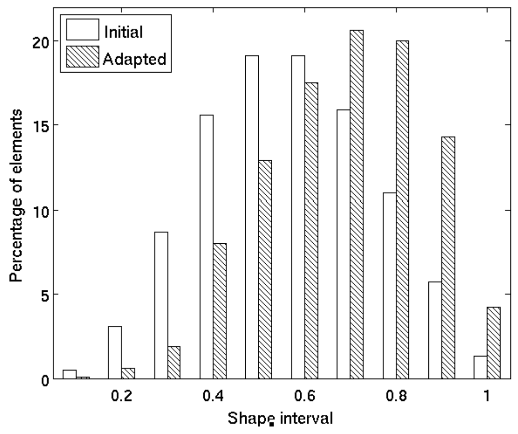


Fig. 26 Edge length (leftmost for interior edges and center for layer edges) and mean ratio (rightmost) distribution in the transformed space from the final adaptation cycle for the heat transfer manifold case.

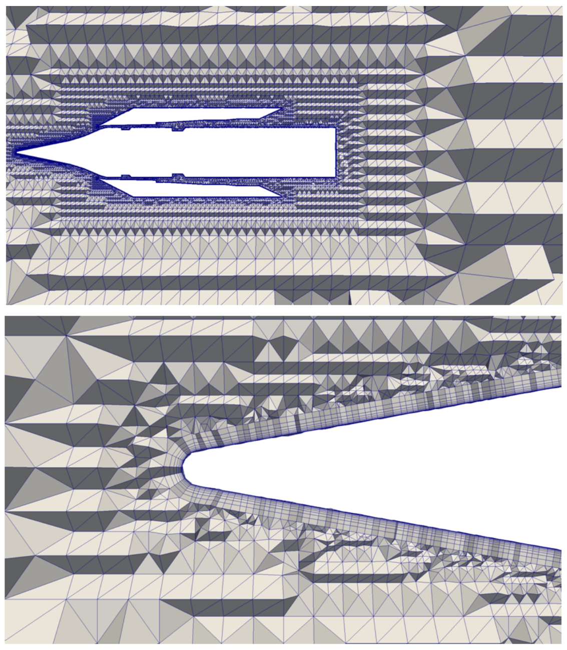


Fig. 27 Slice of the initial mesh for scramjet case: whole engine (top) and a zoomed view of nose cone tip (bottom).

plied locally on the processor and on neighboring parts across part boundaries.

5.4 Scramjet engine

The NASA CIAM scramjet case [45] was run with a free stream Mach number of 6.2, and a free stream reference temperature of 203.5 Kelvin. The initial mesh had 2.86M regions and its slice is depicted in Figure 27. Hessian calculations were based on the Mach number in order to compute the mesh size field.

Two adaptation cycles were completed for the scramjet case. The first adapted mesh had 7.2M regions and the second adapted mesh consisted of 16M regions. Figure 28 presents a mesh clip-plane view of the first and second adapted meshes, whereas Figure 29 shows a zoomed view near the tip of the scramjet engine. For this case also we provide a qualitative assessment of numerical results. Figure 30 presents the Mach number contour plots for the initial, first and second adapted meshes. In addition, Figure 31 shows adapted meshes and con-

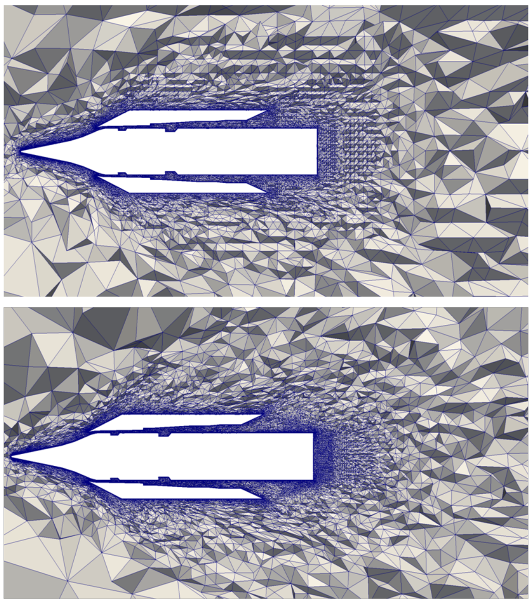


Fig. 28 Clip-plane view of the engine for the first (top) and second (bottom) adapted meshes for the scramjet case.

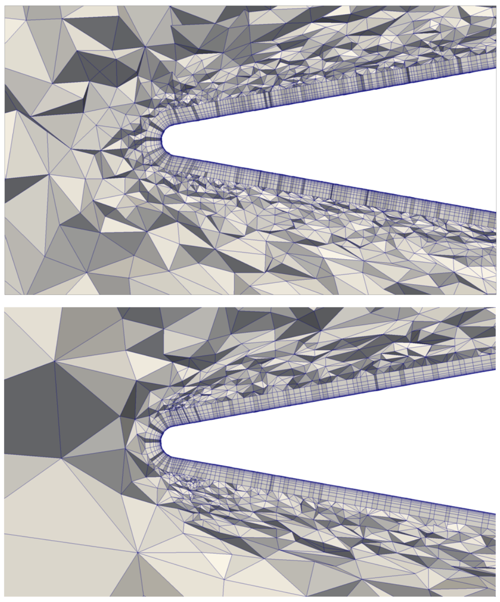


Fig. 29 Clip-plane view near engine tip for the first (top) and second (bottom) meshes for the scramjet case.

tours of the computed Mach number near the cowl lip region of the inlet of the combustor area.

The flow solution resolution is greatly improved through the use of anisotropic mesh adaptation. The second adapted mesh captures the shock far better than the initial mesh. In the far field upstream of the shock, where flow is uniform and parallel, the mesh was appropriately coarsened. Expected mesh refinement was obtained at nose cone tip, at the cowl lip, within the combustor area, at the sharp edges of the combustor area liner, as well as behind the engine. Mesh anisotropy follows the shock emanating from the nose cone tip, with coarsening in the direction tangential to the shock.

Changes in the mesh reflect evidently sharper resolution of flow features in the relevant regions of the domain. The adapted mesh captures the shock better than the initial mesh, and a sharper resolution of the shock can be seen in Figure 30. The resolution of shock in the far-field is limited since it is currently not of primary concern and far-field resolution can easily be improved with more stringent adaptation criteria. Behind the engine also, the flow features are better resolved while using the adapted mesh. Finally, the flow solution in the combustor area is also better resolved which is important in proceeding forward with combustion simulation.

In this case, an anisotropic mesh gradation procedure [36] is also used to reduce high jumps in the mesh size near the nose cone tip. Figure 32 illustrates the impact of anisotropic grading near the nose cone tip. The requested sizes at the wall surface are over an order of

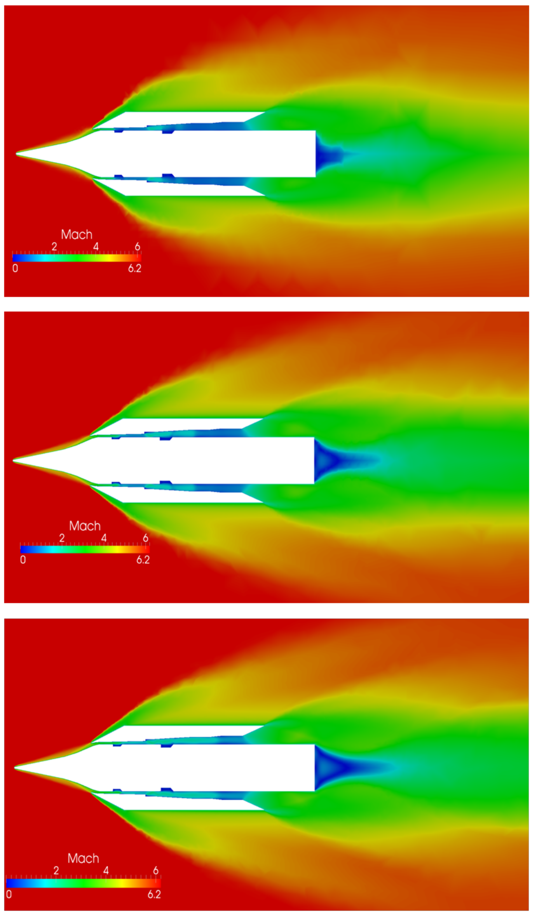


Fig. 30 Clip-plane view of Mach contours for initial (top), first adapted (middle) and second adapted (bottom) meshes.

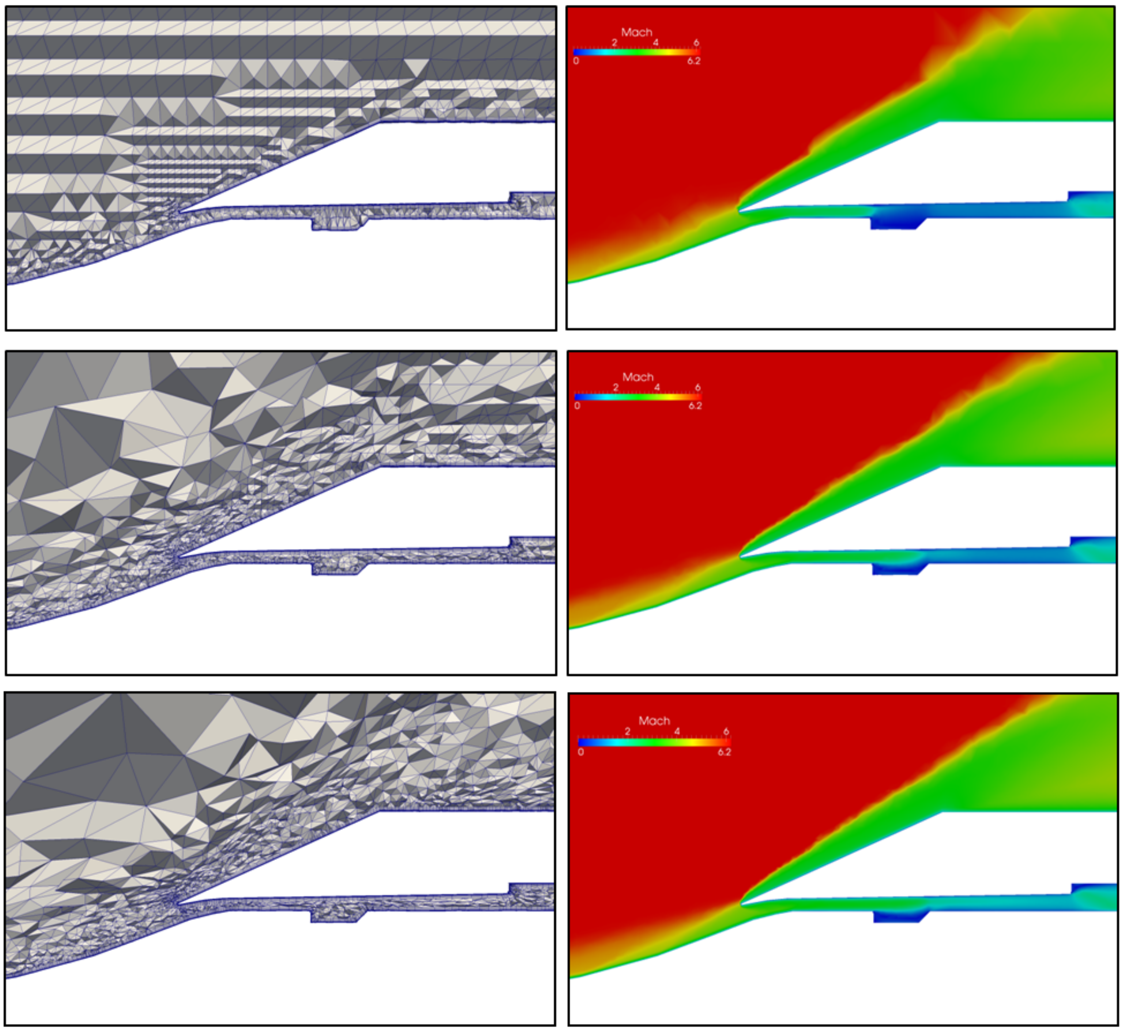


Fig. 31 Initial (top), first adapted (middle) and second adapted (bottom) meshes (left column) and Mach number contours (right column) near the cowl lip and at the entry to the combustor region.

magnitude smaller than the requested sizes in the unstructured mesh region directly outside the boundary layer. As a result, the boundary layer stack at the tip is much more refined than in the adjacent unstructured mesh, leading to so-called "spider elements" and poorly shaped elements locally. As confirmed in Figure 32, gradation of the size field alleviates this issue.

Mesh statistics were also collected for this case for the same three quantities and in the final adaptation cycle. Figure 33 shows these statistics. It can be seen that the input mesh has large number of interior and layer edges whose lengths in the metric space are outside the desired interval whereas interior edges in the adapted mesh fall in this interval. As before, many layer edges in the adapted mesh are finer (or shorter) than desired, which is due to the conservative nature of the split scheme used for layer edges. The percentage of finer layer edges is higher in this case because the computed sizefield has more variation along the thickness of the layered mesh (e.g., near the nose cone tip). Mean ratio also demonstrates that the shape of the elements in the adapted mesh respects the specified mesh metric field.

Table 4 provides timings and scalability for mesh adaptation based on the second adapted meshes. The strong scaling studies were performed on 128 to 4096 processors. The scaling is based on the execution time of 128 processors and is calculated the same way it is defined in the heat transfer manifold test case.

Table 4 supports the observation that the mesh adaptation is able to decrease run times with the growing number of cores. Although the simulation study experiences the fixed size problem phenomena on a high core count observed in the previous test case, the scalability factors show better parallel performance of the scramjet test case compared to the heat transfer manifold case.

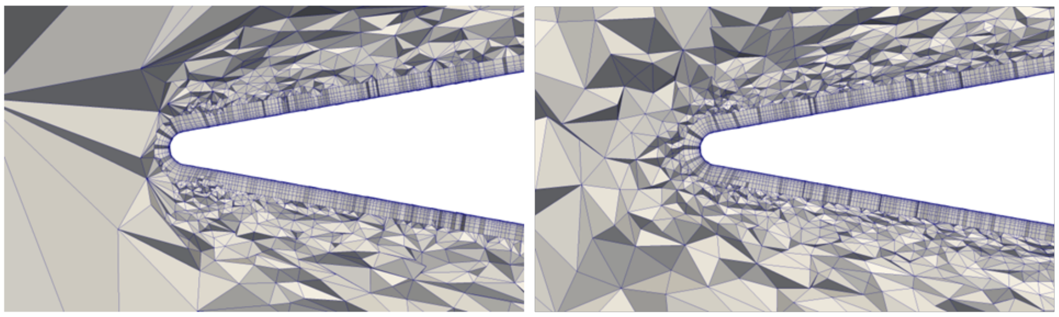


Fig. 32 Impact of anisotropic smoothing for the scramjet adaptation near nose cone tip without grading (left) and with grading (right).

Note that mesh adaptation took 1.2% of the adaptation cycle time on 128 processors and 3.6% of the simulation time on 4096 processors, which is not a significant fraction of time spent for the analysis routines. Thus, even for a fixed size problem, mesh adaptation routines are able to efficiently provide adaptation functionality on high core counts.

Table 5 gives weak scaling results for the scramjet engine test case, where the scaling factor is calculated the same way it was described for the heat transfer manifold test case. It can be found from Table 5 that having a relatively equal amount of workload per part in each test run, the scaling does not severely drop with the increase in number of processors. Unlike the strong scaling results, the scalability factors show better parallel performance of the heat transfer manifold test case compared to the scramjet engine case, even considering the fact that the mesh has increased less in size for the scramjet engine test case during mesh adaptation. As noted earlier, the mesh adaptation is mesh modification type-specific and the number of operations on a part defines the amount of computation and communication which need to be performed for a specific part.

6 Closing remarks

In this article, an adaptive parallel boundary layer meshing procedure is presented. The approach described successfully works with the distributed meshes and effectively supports layered structure of the mesh while providing mesh modification routines in parallel using the anisotropic size field. The parallelization of the anisotropic mesh adaptation routines with boundary layers allows the procedure to be applied to large and complex 3D problem cases.

The procedure has been executed in parallel for three viscous flow problem cases, namely: the ONERA M6 wing case, a heat transfer manifold case and a scramjet engine case. It has been demonstrated that boundary layer mesh adaptation leads to accurate prediction of flow quantities of interest (e.g., surface pressure) and appropriately resolves critical flow regions (e.g., lambda shock). In the ONERA M6 wing case, numerical results on the finest mesh showed good agreement with the experimental data. However, in the other two cases a qualitative assessment was made.

The parallel performance results carried out on problem domains indicate that mesh adaptation is capable of decreasing the simulation run times while being executed on higher number of cores. With the adaptation time taking a small fraction of the adaptation loop, parallel anisotropic mesh adaptation with boundary layers presented in this work is able to support efficient large-scale automated flow simulations. In the future, we plan to include cases where change in number of layers is required and far-field solution features (e.g., far-field shocks) play an important role.

7 Acknowledgments

This work is supported by the National Science Foundation under Grant No. 0749152, and by the U.S. Department of Energy under DOE Grant No. DE-FC02-06ER25769, and by the NASA STTR Part II Grant No. BEE103/NNX11CC69C. Computing support is provided by National Energy Research Scientific Computing Center for granting access to the Cray XE6 supercomputers. Resources at the Computational Center for Nanotechnology Innovation (CCNI) at Rensselaer were also used for testing and development. The authors would like to acknowledge L. Fovargue for the efforts in helping with the ONERA M6 case and F. Nihan Cayan and O. Beslouer for help with the scramjet case.

References

1. MPI: A message-passing interface standard version 2.2. MPI Forum (2009)

Table 4 Mesh adaptation run times and strong scaling for the scramjet case.

N/proc	128	256	512	1024	2048	4096
Time	957.80	532.29	339.39	202.17	136.83	90.85
Scaling	1	0.90	0.70	0.59	0.44	0.33

Table 5 Mesh adaptation run times and weak scaling for the scramjet engine.

Number of processors	256	1536	9216
Initial mesh number of regions	16,166,918	91,201,986	594,353,904
Adapted mesh number of regions	43,444,375	265,189,371	1,776,274,993
Mesh increase factor (M_{incr})	2.69	2.91	2.99
Time	532.29	606.12	734.82
Scaling	1	0.95	0.80

2. Ainsworth, M., Oden, J.T.: A Posteriori Error Estimation in Finite Element Analysis. John Wiley & Sons, New York (2000)
3. Alauzet, F., Li, X., Seol, E.S., Shephard, M.S.: Parallel anisotropic 3D mesh adaptation by mesh modification. Engineering with Computers **21**(3), 247–258 (2006). DOI 10.1007/s00366-005-0009-3
4. Au, P., Dompierre, J., Labbe, P., Guibault, F., Camarero, R.: Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization. In: Proceedings of the 7th International Meshing Roundtable, pp. 459–478 (1998)
5. Bänsch, E.: Local mesh refinement in 2 and 3 dimensions. IMPACT of Computing in Science and Engineering **3**(3), 181–191 (1991)
6. Becker, R., Rannacher, R.: An optimal control approach to a posteriori error estimation in finite element methods. Acta numerica **10**(1), 1–102 (2001)
7. Botasso, C.L.: Anisotropic mesh adaption by metric-driven optimization. International Journal for Numerical Methods in Engineering **60**(3), 597–639 (2004). DOI 10.1002/nme.977
8. Bottasso, C.L., Detomi, D.: A procedure for tetrahedral boundary layer mesh generation. Engineering with Computers **18**(1), 66–79 (2002). DOI 10.1007/s003660200006
9. Bourgault, Y., Picasso, M., Alauzet, F., Loseille, A.: On the use of anisotropic a posteriori error estimators for the adaptative solution of 3D inviscid compressible flows. International Journal for Numerical Methods in Fluids **59**(1), 47–74 (2009). DOI 10.1002/fld.1797
10. Buscaglia, G.C., Dari, E.A.: Anisotropic mesh optimization and its application in adaptivity. International Journal for Numerical Methods in Engineering **40**, 4119–4136 (1997)
11. Castro-Diáz, M.J., Hecht, F., Mohammadi, B., Pironneau, O.: Anisotropic unstructured mesh adaption for flow simulations. International Journal for Numerical Methods in Fluids **25**, 475–491 (1997)
12. Chand, K.K., Diachin, L.F., Li, X., Ollivier-Gooch, C., Seol, E.S., Shephard, M.S., Tautges, T., Trease, H.: Toward interoperable mesh, geometry and field components for pde simulation development. Engineering with Computers **24**(2), 165–182 (2008). DOI 10.1007/s00366-007-0080-z
13. Chandra, S., Li, X., Saif, T., Parashar, M.: Enabling scalable parallel implementations of structured adaptive mesh refinement applications. The Journal of Supercomputing **39**(2), 177–203 (2007). DOI 10.1007/s11227-007-0110-z
14. Chitale, K., Sahni, O., Tendulkar, S., Nastasia, R., Shephard, M., Jansen, K.: Boundary layer adaptivity for transonic turbulent flows. AIAA Paper 13-2445 (2013). DOI 10.2514/6.2013-2445
15. Connell, S.D., Braaten, M.E.: Semistructured mesh generation for three-dimensional Navier-Stokes calculations. AIAA Journal **33**(6), 1017–1024 (1995)
16. de Cougny, H.L., Shephard, M.S.: Parallel refinement and coarsening of tetrahedral meshes. Computer Methods in Applied Mechanics and Engineering **46**, 1101–1125 (1999)
17. de Cougny, H.L., Shephard, M.S., Georges, M.K.: Explicit node point mesh smoothing within the octree mesh generator. Tech. Rep. 1990-10, Rensselaer Polytechnic Institute, Troy, NY (1990)
18. Cray: Cray XE6. [Online]. Available: `http://www.cray.com/Products/XE/CrayXE6System.aspx`, [Accessed Sep. 19, 2012]
19. Foster, T.M., Mohamed, M.S., Trevelyan, J., Coates, G.: Rapid re-meshing and re-solution of three-dimensional boundary element problems for interactive stress analysis. Engineering Analysis with Boundary Elements **36**(9), 1331–1343 (2012). DOI 10.1016/j.enganabound.2012.02.020
20. Freitag, L.A., Ollivier-Gooch, C.: Tetrahedral mesh improvement using swapping and smoothing. International Journal for Numerical Methods in Engineering **40**(21), 3979–4002 (1997)

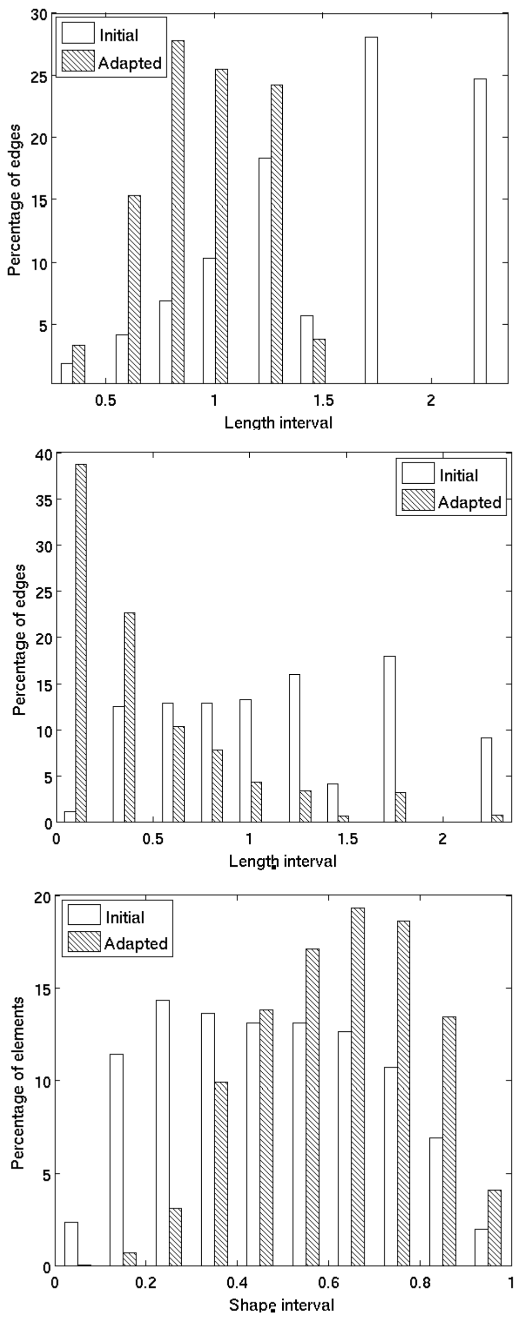


Fig. 33 Edge length (leftmost for interior edges and center for layer edges) and mean ratio (rightmost) distribution in the transformed space from the final adaptation cycle for the scramjet case.

21. Frey, P.J., Alauzet, F.: Anisotropic mesh adaptation for CFD computations. Computer Methods in Applied Mechanics and Engineering **194**(48-49), 5068–5082 (2005). DOI 10.1016/j.cma.2004.11.025
22. Garimella, R.V., Shephard, M.S.: Boundary layer mesh generation for viscous flow simulations. International Journal for Numerical Methods in Engineering **49**, 193–218 (2000)
23. George, P., Borouchaki, H., Laug, P.: An efficient algorithm for 3D adaptive meshing. Advances in Engineering Software **33**(7), 377–387 (2002)
24. Giles, M.B., Süli, E.: Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality. Acta Numerica **11**(1), 145–236 (2002)
25. Hassan, O., Morgan, K., Probert, E.J., Peraire, J.: Unstructured tetrahedral mesh generation for three-dimensional viscous flows. International Journal for Numerical Methods in Engineering **39**, 549–567 (1996)
26. Hassan, O., Morgan, K., Weatherill, N.: Unstructured mesh methods for the solution of the unsteady compressible flow equations with moving boundary components. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **365**(1859), 2531–2552 (2007)
27. Ito, Y., Murayama, M., Yamamoto, K., Shih, A.M., Soni, B.K.: Efficient hybrid surface/volume mesh generation using suppressed marching-direction method. AIAA Journal **51**(6), 1450–1461 (2013)
28. Ito, Y., Nakahashi, K.: Unstructured mesh generation for viscous flow computations. In: Proceedings of the 11th International Meshing Roundtable, pp. 367–377 (2002)
29. Joe, B.: Construction of three-dimensional improved-quality triangulations using local transformations. SIAM Journal on Scientific Computing **16**(6), 1292–1307 (1995)
30. Kallinderis, Y., Kavouklis, C.: A dynamic adaptation scheme for general 3-D hybrid meshes. Computer Methods in Applied Mechanics and Engineering **194**(48-49), 5019–5050 (2005). DOI 10.1016/j.cma.2004.11.023
31. Kallinderis, Y., Vijayan, P.: Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. AIAA journal **31**(8), 1440–1447 (1993)
32. Kavouklis, C., Kallinderis, Y.: Parallel adaptation of general three-dimensional hybrid meshes. Journal of Computational Physics **229**(9), 3454–3473 (2010). DOI 10.1016/j.jcp.2010.01.011
33. Khawaja, A., Kallinderis, Y.: Hybrid grid generation for turbomachinery and aerospace applications. International Journal for Numerical Methods in Engineering **49**(1-2), 145–166 (2000)
34. Khawaja, A., Minyard, T., Kallinderis, Y.: Adaptive hybrid grid methods. Computer Methods in Applied Mechanics and Engineering **189**(4), 1231–1245 (2000). DOI 10.1016/S0045-7825(99)00375-8
35. Li, X.: Mesh modification procedures for general 3d non-manifold domains. Ph.D. dissertation, Depart-

ment of Mechanical Engineering, Rensselaer Polytechnic Institute, Troy, NY (2003)
36. Li, X., Remacle, J.F., Chevaugeon, N., Shephard, M.S.: Anisotropic mesh gradation control. In: Proceedings of the 13th International Meshing Roundtable, pp. 401–412 (2004)
37. Li, X., Shephard, M., Beall, M.: 3D anisotropic mesh adaptation by mesh modifications. Computer Methods in Applied Mechanics and Engineering **194**(48-49), 4915–4950 (2005). DOI 10.1016/j.cma.2004.11.019
38. Li, X., Shephard, M.S., Beall, M.W.: Accounting for curved domains in mesh adaptation. International Journal for Numerical Methods in Engineering **58**(2), 247–276 (2003). DOI 10.1002/nme.772
39. Liu, A., Joe, B.: On the shape of tetrahedra from bisection. Mathematics of Computation **63**(207), 141–154 (1994)
40. Löhner, R., Baum, J.D.: Adaptive h-refinement on 3D unstructured grids for transient problems. International Journal for Numerical Methods in Fluids **14**(12), 1407–1419 (1992)
41. Lohner, R., Cebral, J.: Generation of non-isotropic unstructured grids via directional enrichment. International Journal for Numerical Methods in Engineering **49**(1-2), 219–232 (2000)
42. Loseille, A., Löhner, R.: Robust boundary layer mesh generation. In: Proceedings of the 21st International Meshing Roundtable, pp. 493–511 (2013)
43. Marcum, D.L.: Generation of unstructured grids for viscous flow applications. AIAA Paper 95-0212 (1995)
44. Muller, J., Sahni, O., Li, X., Jansen, K.E., Shephard, M.S., Taylor, C.A.: Anisotropic adaptive finite element method for modeling blood flow. Computer Methods in Biomechanics and Biomedical Engineering **8**(5), 295–305 (2005). DOI 10.1080/10255840500264742
45. NASA: CIAM Axisymmetric Scramjet. [Online]. Available: `http://hapb-www.larc.nasa.gov/Public/Engines/Ciam/Ciam.html`, [Accessed Sep. 19, 2012]
46. NASA: FUN3D online manual. [Online]. Available: `http://fun3d.larc.nasa.gov/`, [Accessed Sep. 19, 2012]
47. Oliker, L., Biswas, R., Gabow, H.N.: Parallel tetrahedral mesh adaptation with dynamic load balancing. Parallel Computing **26**(12), 1583–1608 (2000)
48. Ovcharenko, A., Ibanez, D., Delalondre, F., Sahni, O., Jansen, K.E., Carothers, C.D., Shephard, M.S.: Neighborhood communication paradigm to increase scalability in large-scale dynamic scientific applications. Parallel Computing **38**(3), 140–156 (2012). DOI 10.1016/j.parco.2011.10.013
49. Pain, C.C., Umpleby, A.P., de Oliveira, C.R.E., Goddard, A.J.H.: Tetrahedral mesh optimization and adaptivity for steady-state and transient finite element calculations. Computer Methods in Applied Mechanics and Engineering **190**(29-30), 3771–3796 (2001). DOI 10.1016/S0045-7825(00)00294-2
50. Park, Y.M., Kwon, O.J.: A parallel unstructured dynamic mesh adaptation algorithm for 3-D unsteady flows. International journal for numerical methods in fluids **48**(6), 671–690 (2005)
51. Peraire, J., Peiro, J., Morgan, K.: Adaptive remeshing for three-dimensional compressible flow computation. Journal of Computational Physics **103**(2), 269–285 (1992). DOI 10.1016/0021-9991(92)90401-J
52. Pirzadeh, S.: Unstructured viscous grid generation by the advancing-layers method. AIAA Journal **32**(8), 1735–1737 (1994)
53. Sahni, O., Jansen, K.E., Shephard, M.S., Taylor, C.A., Beall, M.W.: Adaptive boundary layer meshing for viscous flow simulations. Engineering with Computers **24**(3), 267–285 (2008). DOI 10.1007/s00366-008-0095-0
54. Sahni, O., Muller, J., Jansen, K.E., Shephard, M.S., Taylor, C.A.: Efficient anisotropic adaptive discretization of the cardiovascular system. Computer Methods in Applied Mechanics and Engineering **195**(41-43), 5634–5655 (2006). DOI 10.1016/j.cma.2005.10.018
55. Sahni, O., Zhou, M., Shephard, M.S., Jansen, K.E.: Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores. In: Proceedigs of the 2009 ACM/IEEE Conference on High Performance Computing (2009)
56. Sandia National Laboratories: Zoltan unstructured communication utilities. [Online]. Available: `http://www.cs.sandia.gov/Zoltan/ug_html/ug_util_comm.html`, [Accessed Sep. 19, 2012]
57. Seol, E.S., Shephard, M.S.: Efficient distributed mesh data structure for parallel automated adaptive analysis. Engineering with Computers **22**(3-4), 197–213 (2006)
58. Shephard, M.S., Beall, M.W., O'Bara, R.M., Webster, B.E.: Toward simulation-based design. Finite Elements in Analysis and Design **40**(12), 1575–1598 (2004). DOI 10.1016/j.finel.2003.11.004
59. Stogner, R.H., Carey, G.F., Murray, B.T.: Approximation of Cahn-Hilliard diffuse interface models using parallel adaptive mesh refinement and coarsening with C1 elements. International Journal for Numerical Methods in Engineering **76**(5), 636–661

(2008). DOI 10.1002/nme.2337

60. Toussaint, G.T., Verbrugge, C., Wang, C., Zhu, B.: Tetrahedralization of simple and non-simple polyhedra. In: Proceedings of the 5th Canadian Conference on Computational Geometry, pp. 24–29 (1993)
61. V. Schmitt and F. Charpin: Pressure distribution on the ONERA-M6-Wing at transonic mach numbers. In: Report of the Fluid Dynamics Panel Working Group 04, vol. 138. AGARD (May 1979)
62. Verfürth, R.: A Review of Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques. Teubner-Wiley, Stuttgart (1996)
63. Whiting, C.H., Jansen, K.E.: A stabilized finite element method for the incompressible Navier–Stokes equations using a hierarchical basis. International Journal for Numerical Methods in Fluids **35**(1), 93–116 (2001)
64. Xie, T., Seol, E.S., Shephard, M.S.: Generic components for petascale automated adaptive simulations. Engineering with Computers (2012). Accepted for publication
65. Zhang, L., Chang, X., Duan, X., Zhao, Z., He, X.: Applications of dynamic hybrid grid method for three-dimensional moving/deforming boundary problems. Computers and Fluids **62**, 45–63 (2012). DOI 10.1016/j.compfluid.2012.03.008
66. Zhou, M., Sahni, O., Kim, H.J., Figueroa, C.A., Taylor, C.A., Shephard, M.S., Jansen, K.E.: Cardiovascular flow simulation at extreme scale. Computational Mechanics **46**(1), 71–82 (2010). DOI 10.1007/s00466-009-0450-z