Engineering with Computers (2017) 33:767–795
DOI 10.1007/s00366-016-0437-2

ORIGINAL ARTICLE

Parallel anisotropic mesh adaptation with boundary layers for automated viscous flow simulations

Onkar Sahni[1] · Aleksandr Ovcharenko[1] · Kedar C. Chitale[1] · Kenneth E. Jansen[2] · Mark S. Shephard[1]

Received: 14 January 2014 / Accepted: 22 January 2016 / Published online: 1 April 2016

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 current 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 to match/satisfy 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 is demonstrated 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

✉ Onkar Sahni
sahni@rpi.edu

Aleksandr Ovcharenko
shurik@scorec.rpi.edu

Kedar C. Chitale
chitak2@rpi.edu

Kenneth E. Jansen
kenneth.jansen@colorado.edu

Mark S. Shephard
shephard@rpi.edu

[1] Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

[2] University of Colorado at Boulder, Boulder, CO 80309, USA

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 procedures must be applied [2, 6, 24, 62]. Adaptive meshing is therefore an important component for reliable simulation of complex problems, such as for flow problems that exhibit highly anisotropic solutions which can only be located and resolved through a posteriori anisotropic adaptivity (e.g., see [9–11, 21, 49, 51, 54]). Furthermore, in a number of problem cases it is desirable to use highly anisotropic elements (e.g., with aspect ratio above 1000) in specific locations and for these elements to have a semi-structured nature which must be maintained during mesh adaptation [34, 53]. Of particular interest in this study are viscous flows with boundary layers that form near solid surfaces, e.g., in a wall-bounded flow.

The two major classes of mesh adaptation techniques are adaptive re-meshing methods and methods that use local mesh modification. Re-meshing methods [19, 21, 23, 26, 51] 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 modification retriangulate local subdomains (or cavities) until the specified mesh size field is satisfied (e.g., see [7, 37, 49]). 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 the amount of coarsening and anisotropy that can be achieved. For example, in [5, 31, 40] the coarsening or merging of child elements to recover parent elements is done by reversing the previous subdivision operations at desired locations (i.e., by applying a derefinement step). 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 (e.g., based on the initial mesh) and do not improve the geometric approximation of the curved domains as the mesh is refined. In contrast, other research work has shown that a richer set of local mesh modification operations [7, 20, 29, 37, 49] 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 a localized transfer of solution fields [5, 44] at the cavity level as the mesh is incrementally modified to attain the desired mesh.

In viscous flows with boundary layers, hybrid or semi-structured mesh generation methods have been used extensively [8, 15, 22, 25, 27, 28, 33, 41–43, 52]. For such problems, local mesh modification operations have been extended to account for mixed topology elements [30, 34, 53], wherein the semi-structured nature of the mesh is taken into consideration. Specifically the layers or stacks of wedges (triangular prisms) or hexes are modified 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 the geometric approximation of the curved domains. In [30], derefinement is also carried out for transient problems, whereas in [53] a 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 as well as pyramidal elements. The latter are used when necessary to transition between the semi-structured/layered 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 over multiple parts.

Mesh adaptation techniques must operate in parallel on distributed meshes. This is 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]), face a limitation in terms of the problem size and/or time-to-solution due to the serial mesh adaptation step. The serial mesh adaptation step may take as much time, or even more, as compared to the analysis step and thus, becomes a bottleneck. Therefore, to efficiently execute parallel adaptive simulations, both the analysis and mesh adaptation steps must be parallelized and executed 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, i.e., at the inter-part boundaries. Subdivision or refinement operations can be understood at the level of a single element and therefore, can be performed in parallel on each processor including for lower-order mesh entities that reside on inter-part boundaries. This must be followed by a communication step between processors to update the inter-part links based on new mesh entities introduced at the inter-part boundaries (e.g., see [16, 47]). In [47, 50], parallel refinement and derefinement steps are used for unsteady problems, where child elements of a given parent element always reside on the same processor. This makes the merging of child elements straightforward in which a communication step 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 contrast to a parallel scheme that is based on refinement and derefinement steps, parallel re-meshing is used in [26]. In such an approach, mesh elements marked for adaptation (based on a selection criterion) are removed from the distributed mesh leading to cavities or holes in the mesh that are re-meshed. This intermediate mesh is repartitioned with the constraint that every hole to be re-meshed resides solely on a single part or processor in the re-distributed mesh. This scheme is more flexible in terms of shape and orientation of the resulting elements, however, the overall process can be time consuming. For example, a global repartitioning of the mesh, or a re-meshing of a relatively large hole due to concentrated adaptation in a contiguous portion of the domain, leads 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 associated cavity resides solely on one part can be carried out in a similar fashion to the serial case while for a cavity which spans multiple parts a migration of associated mesh elements is needed. A naive sequence of steps, which intermingles on-part mesh modification and mesh migration steps at a low level, will be ineffective due to significant wait times between these steps. However, with a proper control of the on-part mesh modification and mesh migration steps, parallel mesh adaptation based on local mesh modifications has been shown to be efficient [3, 16]. So far such parallel mesh adaptation methods have focused on fully

unstructured/tetrahedral 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 steps are employed for mixed elements. Similar to refinement and derefinement of fully unstructured distributed meshes discussed above, the technique in [32] requires the child elements of a given parent element to always reside on the same processor such that mesh derefinement step 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 that is 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 so far there has been no study on such an approach. The current work presents such an approach based on parallelization of a richer set of local mesh modification operations for distributed boundary layer meshes, i.e., this paper presents parallel procedures for boundary layer mesh adaptation and builds on our prior work on serial boundary layer meshes [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 unstructured interior mesh. Section 4 describes the procedures that are currently used to parallelize different local mesh modification operations for hybrid or 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 of M_i^d.
$\{M_i^d\{M^D\}\}$	the set of mesh entities of dimension D adjacent to M_i^d.

2 Boundary layer mesh terminology

A common method to construct boundary layer meshes with layered elements on walls is the advancing layers method [8, 15, 22, 25, 27, 28, 33, 41–43, 52]. It inflates the unstructured surface mesh on no-slip surfaces, where the boundary layer flow forms. This inflation into the volume is performed along the local surface normals in the form of stack or layers of elements in a graded fashion up to a specified distance. Rest of the domain is filled with unstructured tetrahedral elements while pyramids are used when necessary. An example of a boundary layer mesh for a pipe geometry is shown in Fig. 1. In addition to the layers of prismatic elements and interior tetrahedral elements, this example includes a few pyramids. Note that pyramids are used to transition into 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 a difference in the number of layers in those stacks.

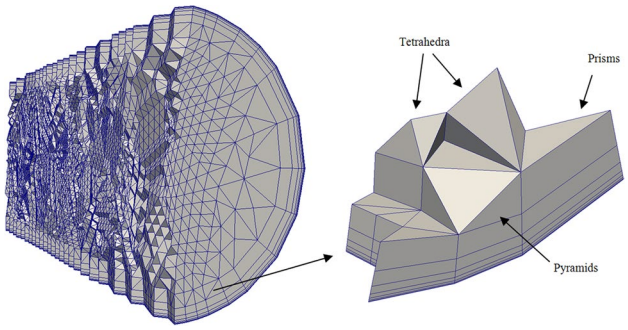

Fig. 1 Cut of the boundary layer mesh for a pipe geometry

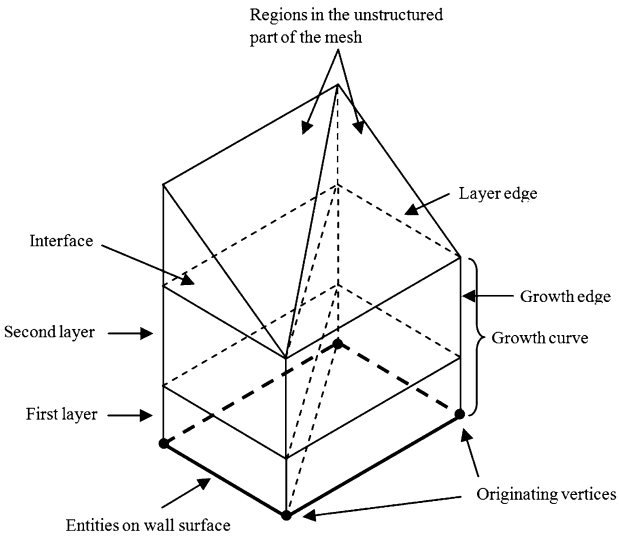

Fig. 2 Boundary layer mesh terminology

The layered portion of the boundary layer mesh has a structure that can be decomposed into a tensor product of a layer surface (2D) mesh and a thickness (1D) mesh [53]. The mesh composed of triangles located at the top or bottom end of any layer is referred to as a layer surface, while the lines normal to the wall composed of mesh edges are called growth curves, see Fig. 2. The mesh 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 two layer surfaces, one above and one below, while the in-between growth edges join these layer surfaces. The height of each layer is referred to as layer thickness whereas the collective height of all the layers is the total thickness of a layered stack of elements. 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 the originating vertices. The top most layer in a layered stack shares an interface with the unstructured interior mesh. The interior tetrahedral or pyramidal elements, sharing lower-order mesh entities with layered portion of the mesh, are referred to as the interface elements.

3 Mesh modifications in the layered portion of the mesh

The goal of mesh modification operations for boundary layer meshes is to maintain the overall layered structure in the mesh. To do this, the mesh modification operations are decomposed such that operations that affect the layer triangulation of the layers are applied consistently throughout the stack. It is also necessary to apply modification to the corresponding unstructured mesh at the top of the stack. This section describes the control of mesh modifications for the layered and unstructured portions of the mesh.

3.1 Mesh metric tensor

A mesh metric field is used to specify the anisotropic mesh size distribution over the problem domain (e.g., see [37, 49, 51]). In an adaptive process, the error estimator or indicator information is used to specify the desired mesh size or metric field. This specification at any given point P is done by a symmetric positive definite tensor $T(P)$, referred to as the mesh metric tensor. A mesh metric tensor contains the desired directional mesh resolution at a point and geometrically follows an ellipsoidal surface. Specifically, a mesh metric 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 or transformed space. Any tetrahedron that perfectly satisfies the mesh metric field should be a unit equilateral tetrahedron in the metric space as depicted in Fig. 3. However, in an unstructured mesh it is often not possible to exactly satisfy the specified mesh metric field. Therefore, mesh modification algorithms constrain edge lengths in the metric space to be within an interval around unity: $[L_{min}, L_{max}]$ (e.g., $[1/\sqrt{2},\sqrt{2}]$), while elements are desired to achieve a mean ratio in the metric space to be close to 1 (with 1 being the ideal value). Mean ratio for a

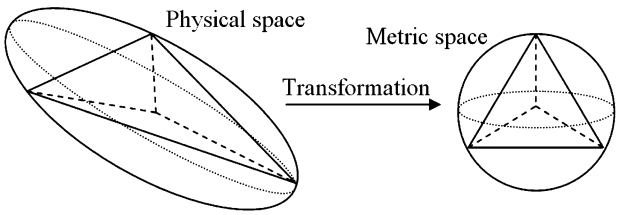

Fig. 3 Transformation of a tetrahedral element based on a mesh metric tensor

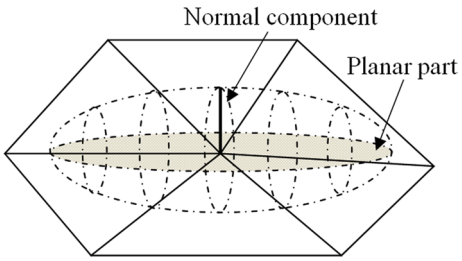

Fig. 4 Decomposition of a mesh metric tensor in layered part of the mesh

tetrahedron was defined in [39]. In the metric space, mean ratio is defined as [37]: $\eta = 12(3V_T)^{2/3}/(\sum_{i=1}^{6} l_{T,i}^2)$, where V_T is the volume of the tetrahedron and $l_{T,i}$ is the length of the ith edge of the tetrahedron in the metric space. As discussed later, we employ the cube of the mean ratio as the measure for element shape quality (e.g., see [37]).

In the layered part of the mesh, the mesh metric tensor can be decomposed into an ellipse as the planar (2D) part along a layer surface, which dictates the local in-plane mesh resolution, and a normal (1D) component that controls the 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 a 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 to match the specified size field. As discussed before, a richer set of local mesh modification operations [7, 37, 49] 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 boundary of the cavity can be shared with unchanged mesh entities outside of the cavity or in unaffected portion of the mesh (under this operation).

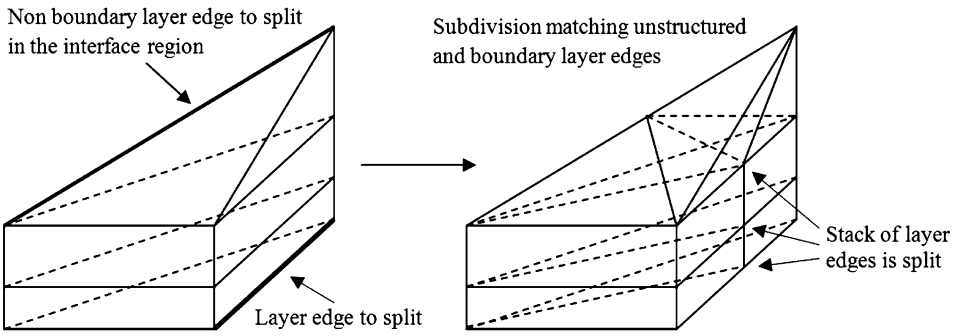

Fig. 5 Example of a layer edge split operation

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:

$$\begin{aligned}\left\{C(M_k^d)\right\} = \{&M_i^3 \bigcup \left\{\partial M_i^3\right\} | M_i^3 \\ &\text{is affected by mesh modification} \\ &\text{operation applied to } M_k^d\}.\end{aligned} \tag{1}$$

The cavity's boundary is defined as:

$$\begin{aligned}\left\{\partial C(M_k^d)\right\} = \{&M_j^2 \bigcup \left\{\partial M_j^2\right\} \in C(M_k^d) | \\ &M_j^2 \notin \left\{\partial M_i^3\right\} \bigcap \left\{\partial M_j^3\right\} \\ &\forall M_i^3, M_j^3 \in \left\{C(M_k^d)\right\}\}.\end{aligned} \tag{2}$$

Equation 2 states that the cavity's boundary contains the set of lower-order entities $\left\{M_j^d\right\}$ $(d = 0, 1, 2)$ that are located on the outer boundary of the closed set of regions comprising cavity $\left\{C(M_k^d)\right\}$. This way the cavity's boundary is shared with adjacent mesh regions that are outside and thus, not part of the cavity.

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

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

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

There can be situations when an entity M_k^d which requests modification is (only) repositioned within the cavity with no change in local mesh topology (for example, in case of a vertex motion as considered in Sect. 3.3). In this scenario, Eq. 3 will have the equality.

3.3 Boundary layer stack modification

To preserve the layered nature of the boundary layer stacks, the mesh adaptation process for layer surfaces utilizes layer edge split, collapse and swap operations [53], while adjustment of layer thicknesses and movement of newly created originating vertices to the curved domain boundary are accomplished through vertex movement (that may involve direct repositioning or local mesh modification operations as discussed later).

The layer edge split operation splits edges in the boundary layer stack and applies the appropriate subdivisions to the unstructured interior mesh at the interface. 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 a stack of N layer edges, then the cavity associated with it consists of a set of unique regions $\left\{C\left\{M_i^1\right\}\right\} = \left\{\bigcup_{i=1}^N \left\{M_i^1\left\{M^3\right\}\right\}\right\}$. 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 all short edges in the metric space. It is carried out under this condition to avoid any 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 operation. Let $\left\{(M_i^1, M_i^0)\right\}$ (with $i \in [1..N]$) be N pairs of layer edges (to be collapsed) and their corresponding vertices (to be deleted) in a stack. Then the cavity associated with the layer edge collapse operator is $\left\{C\left\{M_i^0\right\}\right\} = \left\{\bigcup_{i=1}^N \left\{M_i^0\left\{M^3\right\}\right\}\right\}$ in which regions $\left\{\bigcup_{i=1}^N \left\{M_i^1\left\{M^3\right\}\right\}\right\}$ are deleted. Figure 6 shows the 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 to an edge swap operation for tetrahedra, which are reconfigured based on the equatorial plane, there is only one other possible configuration for layer faces in case of a layer edge swap operation [53]. If $\left\{M_i^1\right\}$ (with $i \in [1..N]$) are the layer edges to be swapped, then the layer edge swap operation retriangulates the cavity $\left\{C\left\{M_i^1\right\}\right\} = \left\{\bigcup_{i=1}^N \left\{M_i^1\left\{M^3\right\}\right\}\right\}$ and new layer edges are introduced inside the cavity while

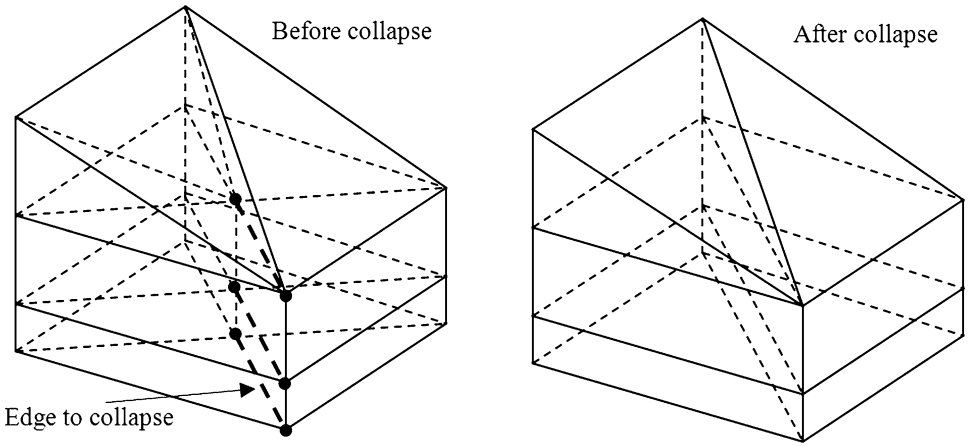

Fig. 6 Example of a layer edge collapse operation

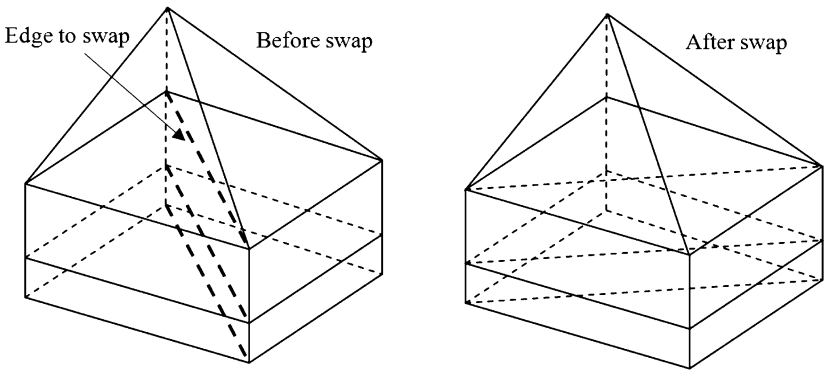

Fig. 7 Example of a layer edge swap operation

old layer edges $\{M_i^l\}$ are deleted. The unstructured interior regions at the interface are also retriangulated and in general are not guaranteed to be in a valid configuration after the swap operation. Thus, appropriate checks are required to ensure that the layer edge swap operation results in a valid mesh after its completion. Figure 7 gives an example of the layer edge swap operation.

When edge split operations are applied to layer edges on curved wall surfaces, the newly introduced originating vertices must be moved to the curved boundary to maintain the proper geometric approximation. All the vertices on a growth curve should gradually be moved to follow the originating vertex, i.e., based on a movement vector [53]. The movement vector is determined for each stack of vertices based on the target location of the originating vertex on the curved domain boundary. This is given as: $\mathbf{m} = \mathbf{v_0^t} - \mathbf{v_0}$, where $\mathbf{v}_0^t$ is the target location of the new originating vertex on the curved boundary (superscript t is used to denote target) and $\mathbf{v}_0$ is the location of the new originating vertex on the original layer edge. The movement vector $\mathbf{m}$ is then applied to all the vertices on that growth curve. The procedure first evaluates the target location of every vertex on all new 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 or original location of the ith vertex on a growth curve and $\mathbf{v}_i^t$ is its target location. The procedure then moves the vertices to their computed target locations. An example is depicted in Fig. 8. Similar to the repositioning of a newly created vertex in the unstructured mesh to the curved boundary, direct repositioning of a new originating vertex is not always possible without additional local mesh modification. Specifically for the top most vertices, as it may introduce inverted elements. In this case local mesh modification operations are applied to the unstructured interior mesh for it to allow in a successful repositioning of the top most vertex. This step is followed by repositioning 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 on serial boundary layer meshes [53]. This is done to achieve more flexibility in parallel mesh adaptation of boundary layer meshes. Pyramids are subdivided during the refinement of the unstructured part of the mesh.

There are three ways the quadrilateral face of a pyramid, which caps an exposed side of a stack, can be subdivided, see Fig. 9. While a refinement in the lateral direction splits

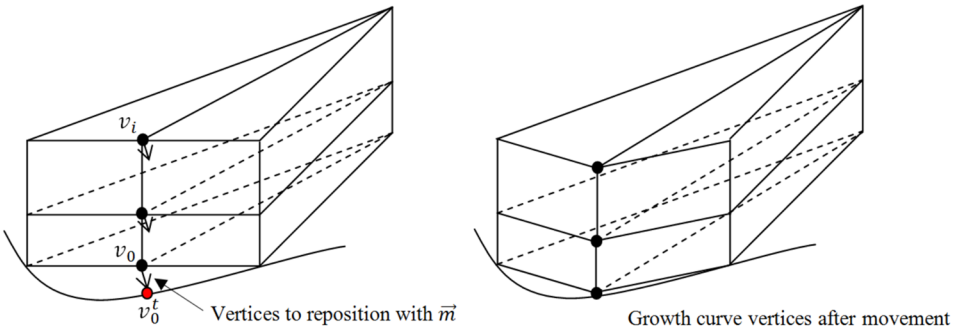

Fig. 8 Repositioning of boundary layer vertices due to movement of the newly created originating vertex to the curved domain boundary

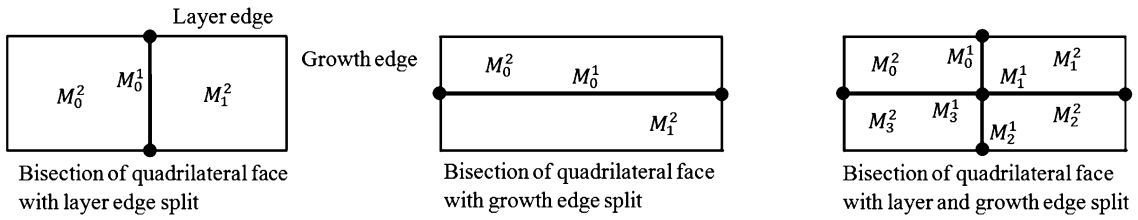

Fig. 9 Subdivision of a quadrilateral face based on different edge splits

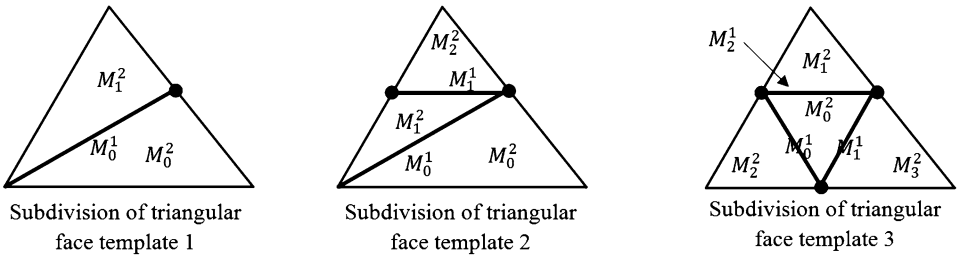

Fig. 10 Subdivision templates for a triangular face

the quadrilateral face using layer edges only (see the left image in Fig. 9), the request to change the number of layers or the resolution along the normal or thickness direction can be achieved with bisection of the quadrilateral face along the growth edges (see the middle image in Fig. 9). Additionally, subdivision can be performed in both directions with the split of both layer and growth edges (see the right image in Fig. 9).

In this 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 problem cases considered in this work.

The rest of triangular faces of a pyramid is subdivided by counting the number of layer edges that are marked or tagged to be split, see Fig. 10. The triangular face templates are the same for the subdivision of layer faces and unstructured interior faces, and thus, do not introduce any additional ambiguity associated with the refinement of triangular faces.

The templates for a pyramid depend on the number of marked edges and the selection of the diagonal edge on a triangular face when two of its edges are marked [37]. This leads to a total of twenty-five subdivision templates for a pyramid. The most frequently used templates for subdividing pyramids are shown in Fig. 11.

3.5 Unstructured decomposition of boundary layers

Figure 12 shows a close-up of an adapted mesh on a pipe geometry without unstructured decomposition of prisms (on the left) and with prisms divided into tetrahedra and pyramids (on the right). One can observe that on the left part of the figure there are boundary layer regions with low aspect ratio. It includes some layered regions that have growth edges that are longer than the layer edges which is counterintuitive in a boundary layer mesh and are not desirable. To satisfy the desired mesh resolution in such cases the corresponding regions in top portion of the stacks are converted into unstructured part of the mesh

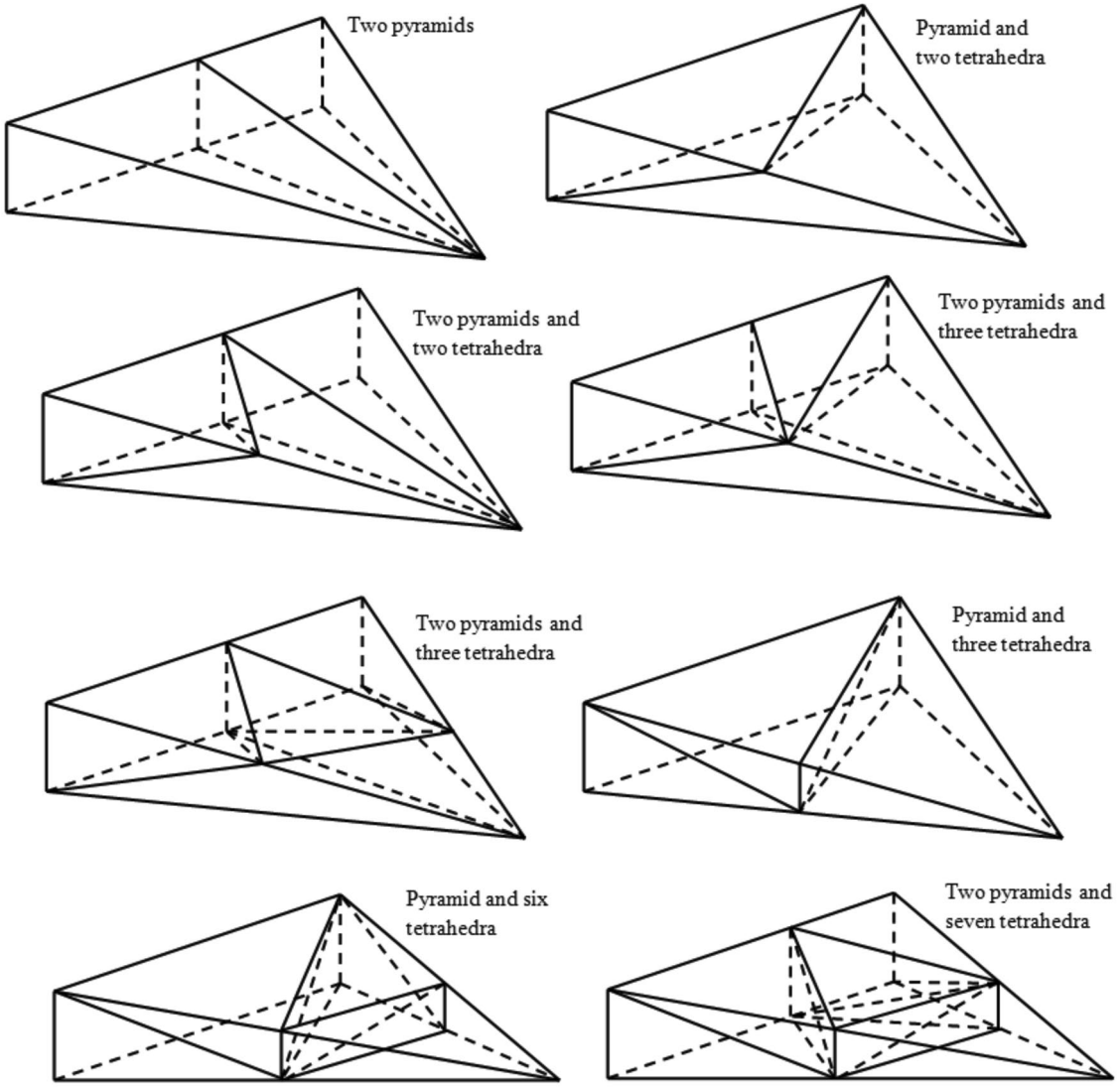

Fig. 11 Subdivision templates for a pyramid

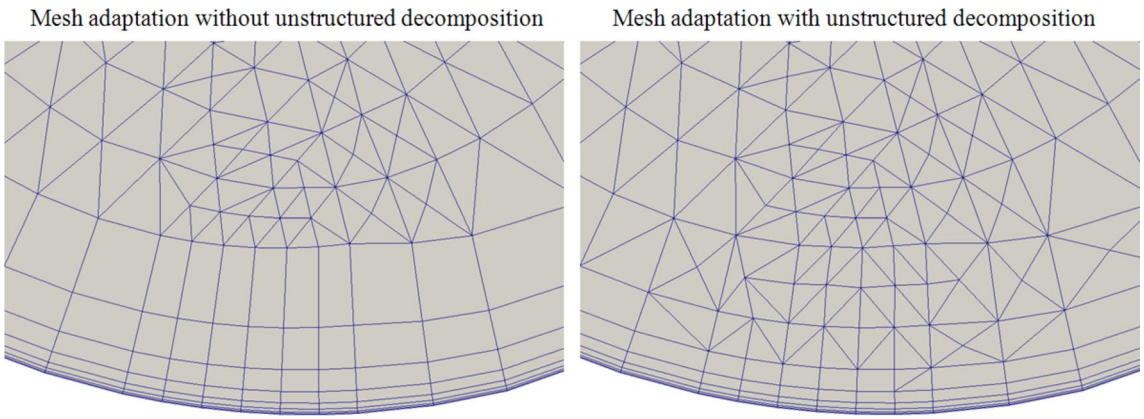

Fig. 12 Mesh adaptation without and with unstructured decomposition of regions with relatively low aspect ratio in the *top* portion of the stacks

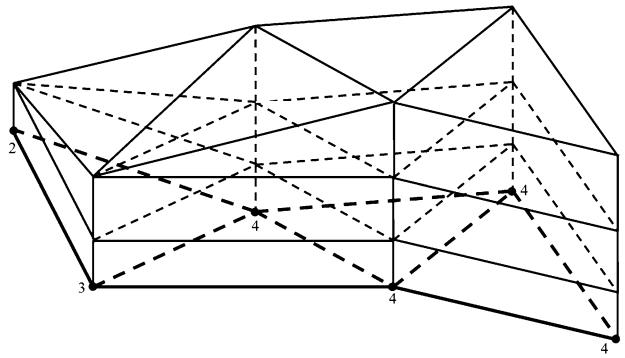

Fig. 13 Different number of layers in neighboring boundary layer stacks. The number of vertices on growth curves is indicated by the count shown at originating vertices. Bold solid and dashed lines at the bottom represent edges on the wall

and subsequently more flexible unstructured mesh modification operations can be applied to achieve the appropriate level of mesh resolution and anisotropy. This is an additional feature to our previous work on serial boundary layer meshes [53].

The process of unstructured decomposition of prisms or reducing the numbers of layers in selected stacks, also referred to as trimming, leads to introduction of additional quadrilateral faces that must be capped with pyramids so that the mesh can be transitioned into tetrahedral elements in the unstructured interior mesh. This step can be reversed when a thicker boundary layer or more layers are desired. It can be done by splitting growth edges that will reintroduce more layers (locally). However, as mentioned before, split of growth edges is not considered currently since it was not necessary for the problems cases considered in this work.

The current algorithm for stack decomposition relies on adjacent stacks of regions that share a face to differ in the number of layers (or prisms) by only one, in which case the top prism on the higher stack must be connected to a pyramid. This is done to transition between different number of layers in face-neighbor stacks of regions. This condition can be enforced by controlling the number of vertices in neighboring growth curves in a preprocessing step before executing the trimming step.

Figure 13 shows boundary layer stacks with different number of layers where any two face-neighbor stacks 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 the quadrilateral faces adjacent to it. If vertices next to each other are of the same level, the priority is granted to the one having the smallest local vertex identifier (ID) which eliminates any possible ambiguity in selecting the diagonal edge of the quadrilateral face.

The application of this restriction during trimming of the stacks yields a favorable situation in which all prisms subdivided into tetrahedra can be triangulated without the introduction of an interior vertex [60]. Avoiding an interior vertex eliminates certain algorithmic complexities and typically results in a better control of the element shape quality [35].

Algorithm 1 Pseudo code for the unstructured decomposition of boundary layers.

```
for each boundary layer stack BL_i do
  for each region in the stack M_j^3 ∈ BL_i from bottom to top do
    assign: AR_j = aspect ratio of M_j^3
    if AR_j < AR_min (where AR_min is the prescribed limit for aspect ratio) then
      tag BL_i for unstructured decomposition
      determine and assign the new number of layers to BL_i
      assign the new number of layers to corresponding originating vertices (i.e., maximum of surrounding stacks)
      break
    end if
  end for
end for
for each level starting from 0 do
  if number of layers assigned to any two face-neighbor stacks differ by more than 1 then
    reduce the number of layers assigned to the higher stack BL_j to achieve a difference of 1
    tag BL_j for unstructured decomposition
    assign the new number of layers to BL_j
  end if
end for
bisect quad faces of each stack BL_i tagged for unstructured decomposition
subdivide regions adjacent to bisected quad faces
tetrahedralize any two pyramids sharing a quad face
```

A prism can be subdivided into a pyramid and a tetrahedron when two of its quadrilateral faces are split and the two diagonal edges share a common vertex, or into three tetrahedra when all quadrilateral faces are subdivided and there is at least a common vertex between any two diagonal edges. This is depicted in Fig. 14. This logic eliminates the possibility for a prism being subdivided to have only one quadrilateral face that is split, or that there is no common vertex between diagonal edges.

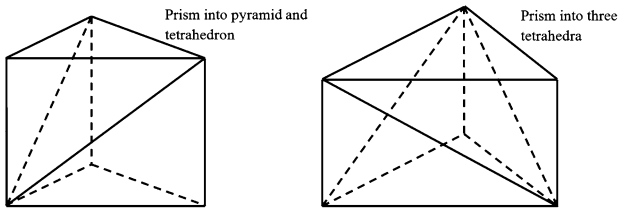

Fig. 14 Subdivision of a prism based on bisection of quadrilateral faces with diagonal edges sharing a common vertex

The algorithm for unstructured decomposition of the appropriate portions of a boundary layer mesh consists of three parts. The first part is responsible for determining the layers in each stack that need to be decomposed based on the allowed minimum value of the aspect ratio. The second part adjusts the number of layers for any two face-neighbor stacks of regions such that they have difference of no more than one after the unstructured decomposition is accomplished. The third step assigns each quadrilateral face with an appropriate diagonal edge using the rules described above. Algorithm 1 presents the overall process of the unstructured decomposition of boundary layers.

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 improvement [53]. The first two stages are controlled by analysis of mesh edge length in the metric or transformed space, whereas the third stage is dictated by both mesh edge length and element shape or quality control. The layered part of the mesh is given a 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 (e.g., layer edge split is applied throughout the stack) and thus, resolved first.

The overall procedure is given in Algorithm 2. It starts with the coarsening stage. The coarsening stage applies local mesh modification operations to eliminate the majority of edges shorter than 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 during the adaptation process. 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 boundary layer mesh adaptation.

```
get mesh metric field from the application
apply metric decomposition in layered part of the mesh
coarsen short layer edges in the metric space
coarsen short interior edges in the metric space
adjust thickness for each growth curve
while mesh resolution is not satisfied do
    tag layer edges that are long in the metric space
    split tagged layer edges, their adjacent faces and regions except for the interior ones
    tag interior edges that are long in the metric space
    split tagged interior edges, their adjacent faces and regions
    move new originating vertices at curved boundaries onto solid surface
    move new unstructured vertices at curved boundaries onto solid surface
    coarsen short layer edges introduced in this refinement iteration
    apply unstructured decomposition to top portion of stacks with relatively low aspect ratio
    coarsen short interior edges introduced in this refinement iteration
end while
eliminate poorly shaped layer faces by layer edge swap or collapse operators
eliminate poorly shaped interior regions by swap or compound operators
```

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 $\sqrt{2}$, respectively [37]. The procedure ensures that the refinement is applied to stacks 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 in this stage, any elements in the stack that have relatively low aspect are tetrahedralized and made part of the unstructured interior mesh. Thus, removing them from the stack by applying unstructured decomposition of boundary layers. The refinement iterations terminate when no long layer or interior edges are left.

The third stage applies shape improvement operations to improve the quality of poorly shaped entities in the transformed space. We use mean ratio [39] as the measure for element shape quality, specifically the cube of the mean ratio in the transformed space [37, 53]. This is done for

both layer faces (in layered part of the mesh) and tetrahedra (in unstructured interior part of the mesh).

Poorly shaped entities (i.e., those below a certain value of quality measure) are modified using sets of swap and compound operators [37, 53] to obtain the best possible element quality while preserving the desired edge length in the transformed space [4, 17, 39]. Again, the shape correction operations are first carried out in the layered part of the mesh and then followed by mesh optimization in the unstructured interior part [53].

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 a set of mesh entities and is treated as a partitioned mesh with the addition of inter-part boundaries within the mesh. A partitioned mesh is managed by the parallel mesh database that tracks the mesh entities residing on inter-part boundaries.

The application of a local mesh modification in parallel involves a cavity of mesh entities that are all on one part or are on multiple parts. In the case where the entities associated with a cavity are on a single part, the mesh modification can be carried out on that part and thus making the parallel execution straightforward. However, in cases where the entities in the cavity are distributed on multiple parts, some form of inter-part or inter-processor coordination is needed.

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 [57] employed maintains the information on mesh entities on inter-part boundaries such that all parts sharing an entity maintain an on-part copy as well as remote copies corresponding to other parts. It supports the ability to update mesh entities on inter-part boundaries if they are modified (e.g., due to an edge split). It also supports the movement or migration of mesh entities from one part to another, which is referred to as a mesh migration step and in which the inter-part boundaries are automatically updated.

Mesh migration is needed to localize a cavity associated with a mesh modification operator. For certain mesh modification operations, such as collapse and swap, the direct consideration of cavities spanning multiple parts leads to a complex and expensive procedure since it requires a number of communication steps to properly carry out the mesh modification operation and update the local mesh in each part. Thus, before applying the mesh modification operation, such cavities are localized on one part or processor such that the cavity retriangulation can be carried out as in the serial case. In cavity localization, all regions and stacks of regions involved in the mesh modification operation are migrated onto a single part [3, 57].

Note that during mesh adaptation, mesh migration is also needed to control the memory usage since the adaptive mesh modification process will alter the numbers of entities on a part (e.g., due to concentrated refinement in a part) and thus the mesh must be dynamically repartitioned. The Zoltan library [56] is used to perform the dynamic repartitioning.

For a boundary layer mesh, the mesh modifications in the layered part of the mesh are applied to the entire stack, and therefore, maintaining the knowledge of the stack is critical. In parallel, managing this information would be difficult if the mesh regions in a stack were distributed over multiple parts. Thus, the implementation of parallel boundary layer mesh adaptation requires all the mesh regions in a boundary layer stack to be placed in a mesh set [64] and each such mesh set is required to reside on a single part along with the ability to migrate such a set to another part.

Additionally, mesh modification and migration involve many irregular or unstructured messages of relatively small sizes. Thus, the parallel efficiency and scalability depend 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 the inter-processor communications by exploiting mesh neighborhood of a given part and by packing small messages into larger messages.

Algorithm 3 Pseudo code for refinement of the parallel boundary layer mesh.

```
1: for all edges M_i^1 (and faces M_i^2) which are tagged for refinement do
2:    split M_i^1 (and M_i^2)
3:    if the entity being split resides on inter-part boundary then
4:       attach the list of newly created entities NewEntList to corresponding M_i^1 (and M_i^2)
5:       add M_i^1 (and M_i^2) to the list on update of inter-part links UpdateLinksList
6:    end if
7: end for
8: for all M_i^1 (and M_i^2) in UpdateLinksList do
9:    send the attached NewEntList to remote copies of M_i^1 (and M_i^2)
10:   receive and set up links between NewEntList of M_i^1 (and M_i^2) on local part with ones from remote parts
11: end for
12: subdivide layered and unstructured regions
```

4.2 Refinement and boundary vertex repositioning

Subdivision of mesh edges and their adjacent mesh faces on inter-part boundaries happens the same way as it is done in serial [3, 16]. The duplicate faces on inter-part boundaries maintain the bounding edges and vertices in the same order on each part to ensure the triangulations are consistent across face neighbors. Note that triangular faces can be split using any combination of edges tagged for refinement whereas quadrilateral faces (which are part of the boundary layer stacks) are subdivided using opposite edges, see Figs. 9 and 10. When a quadrilateral face is bisected for unstructured decomposition of boundary layers (see Fig. 14), then the procedure ensures that the diagonal edge of the face is created in the same way on both parts sharing the face. This way no invalidity is introduced during mesh triangulation and matching of the newly created entities on inter-part boundaries.

The inter-part links between newly created mesh entities are updated across the parts in a communication step such that the distributed mesh is correctly connected. In the execution of the refinement step, the corresponding old-to-new entity mapping is formed such that the links for newly created entities can be effectively set during the communication step. The communication step is carried out after all tagged edges and faces have been split. The mesh regions are subdivided using the same templates as in serial without any communication (i.e., only lower-order entities on inter-part boundaries require a communication step). The pseudo code of the parallel refinement algorithm is given in Algorithm 3.

The algorithm for updating the inter-part links involves the same logic for both layered and unstructured parts of the mesh. The only difference for the layered part is that the update for any stack involves the entire subdivided portion of that stack.

Figure 15 demonstrates an example of the parallel refinement procedure. The initial distributed mesh is depicted in Fig. 15a, where thick lines indicate edges and adjacent 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 (Fig. 15a). Consider top view of the stack (Fig. 15b, c) showing refinement of edges M_0^1 and M_1^1 on each part. Figure 15b shows the introduction of the vertex M_0^0 splitting the edge M_0^1, and Fig. 15c shows M_1^0 splitting the edge M_1^1. On each part, the newly created vertex and two edges are the child entities and attached to the parent edge on inter-part boundary, namely: $M_0^1 \rightarrow \{M_0^0, M_2^1, M_3^1\}$ on P_0 and $M_1^1 \rightarrow \{M_1^0, M_4^1, M_5^1\}$ on P_1. To set up the correct inter-part links between the new entities, a communication step is carried out for 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. On P_0, M_0^1 has a link to M_1^1 and using this link it sends to P_1 the list of child entities, whereas on P_1, M_1^1 has a link to M_0^1 and using this link it sends the list of child entities to P_0. This way P_1 receives the message for M_1^1 containing the list of remote copies of its children on P_0 and vice-versa. Then on P_1, the old-to-new mapping is used to update 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. Same is done on P_0. This is depicted in Fig. 15c. After all edges and faces are split on the inter-part boundaries and the links are updated, the regions are subdivided with no further communication. The resulting mesh is depicted in Fig. 15d.

In the process of refinement, each part maintains a list of new mesh vertices that reside on curved domain boundary and need to be projected [3]. For the layered part of the mesh, the newly created originating vertices are projected onto to the curved surfaces with the help of the movement vector as described in Sect. 3.3. In cases where a direct repositioning will introduce invalidities in the unstructured part of mesh (i.e., at the top of the stack), a more extensive set of local mesh modification operations, which includes collapses, swaps and/or splits, is used. This is done in parallel which involves mesh migration (as discussed above). Algorithm 4 describes vertex repositioning procedure for the parallel boundary layer mesh.

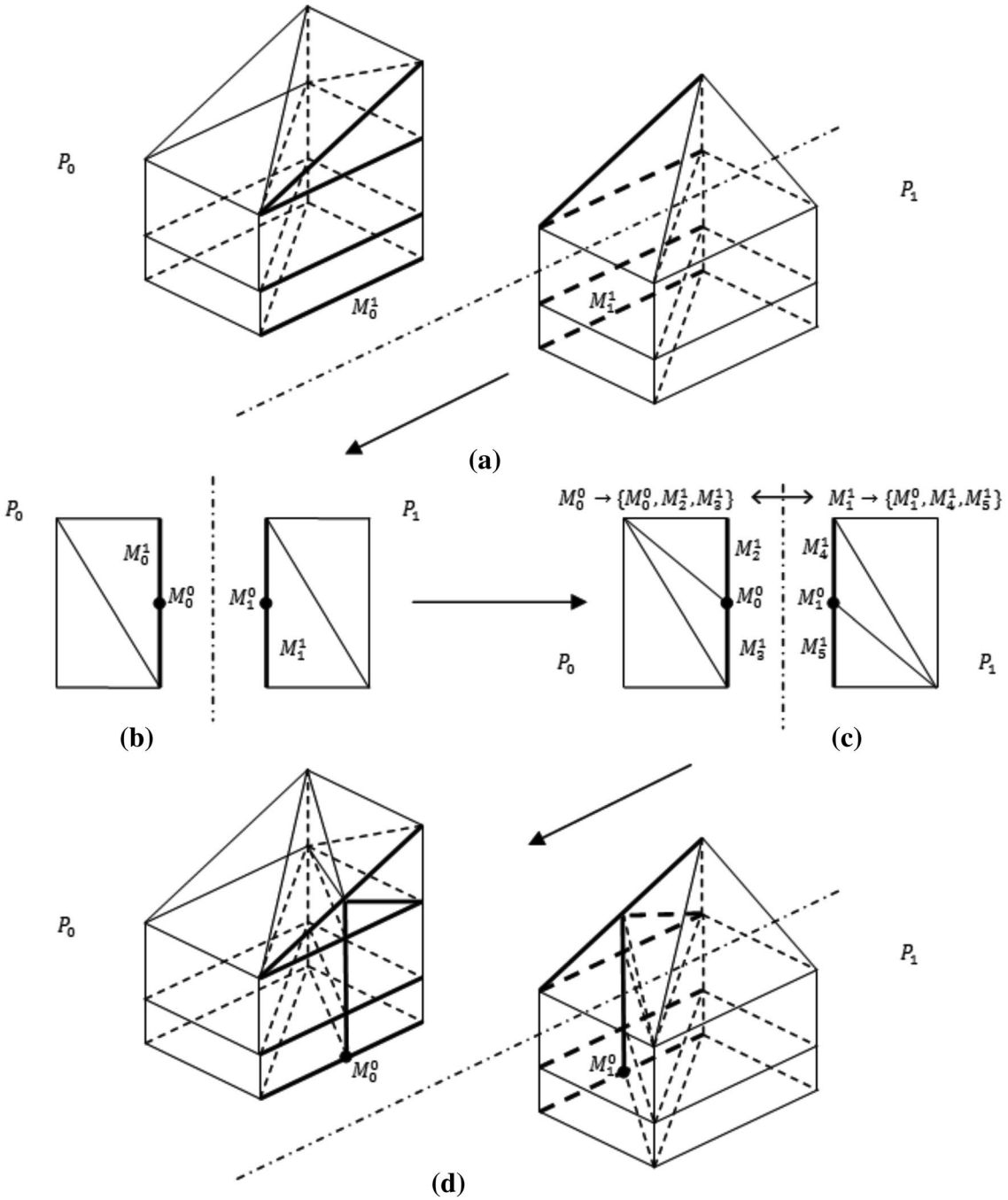

Fig. 15 Example of layer edge split in parallel

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

for all new originating vertices M_i^0 on model boundary with target locations $\mathbf{x}^t_{M^0_{i_{orgn}}}$ **do**
  calculate target locations for each vertex on corresponding growth curve GC_j (i.e., $M_i^0 \in GC_j$)
  add the top most vertex $M^0_{i_{top}} \in GC_j$ with its corresponding target location $\mathbf{x}^t_{M^0_{i_{top}}}$ to $VtxListToMove$
end for
while $VtxListToMove$ is not empty in all parts **do**
  for all $M_i^0 \in VtxListToMove$ **do**
    move M_i^0 to its target location
    if movement introduces flat or inverted regions **then**
      if M_i^0 is on inter-part boundary **then**
        add M_i^0 to $VtxAdjRgnMigrate$
      else
        if possible then apply local mesh modification to ensure the movement of M_i^0 to its target location
      end if
    else
      remove M_i^0 from $VtxListToMove$
    end if
  end for
  send each vertex $M_i^0 \in VtxAdjRgnMigrate$ to its remote copies
  move vertices which receive messages from remote copies (based on $VtxAdjRgnMigrate$) to their original location
  perform mesh migration based on $VtxAdjRgnMigrate$
  update $VtxListToMove$ on each part after migration
end while
for all growth curves GC_j in which repositioning of the top most vertex was performed **do**
  move each vertex in the stack to its target
end for

4.3 Coarsening and swapping

Layer edge collapse operation is always performed on a localized or on-part cavity [3, 16]. For applying a layer edge collapse operation on a growth curve, the boundary layer stacks adjacent to the growth curve are checked. In this step, all the layer edges that are adjacent to the vertices on the growth curve, starting at the originating vertex $M^0_{j_{orgn}}$ and ending at the top most vertex $M^0_{j_{top}}$, are considered. If no surrounding stack of short layer edges can be collapsed locally, then the boundary layer coarsening procedure migrates all the layered and interface regions adjacent to growth-curve vertices (from $M^0_{j_{orgn}}$ to $M^0_{j_{top}}$) onto a single part. The procedure then checks for the validity of the layer edge collapse operation for the given growth curve and proceeds with it.

Figure 16 shows the example of an layer edge collapse operation requiring migration. It can be seen from the figure that growth-curve vertices $[M^0_{j_{orgn}}, \ldots, M^0_{j_{top}}]$ reside on the inter-part boundary and the collapse operation cannot be carried out. Thus, all the adjacent layered and interface regions are migrated to one part P_2 to perform the layer edge collapse operation.

Algorithm 5 Pseudo code for coarsening of the parallel boundary layer mesh.

Create and fill the list of originating vertices $VtxListToClps$ connected to atleast a stack of layer edges with all short edges in the metric space
while $VtxListToClps$ is not empty in all parts **do**
  for all originating vertices $M^0_{j_{orgn}}$ in the list of vertices $VtxListToClps$ **do**
    find stack of shortest adjacent edges corresponding to vertices M_j^0 on growth curve GC_j
    if adjacent layered and interface regions are on one part **then**
      check for validity of the layer edge collapse operation and apply it
      update $VtxListToClps$
    else
      add vertices on inter-part boundary $[M^0_{j_{orgn}}, \ldots, M^0_{j_{top}}] \in GC_j$ into the list $VtxToMigrate$
    end if
  end for
  perform mesh migration based on the list $VtxToMigrate$
  update $VtxListToClps$ based on migration
end while

Algorithm 5 presents the procedure for coarsening of the parallel boundary layer mesh. It starts with a (dynamic) list on each part which consists of originating vertices that are connected to atleast a stack of layer edges with all short edges in the metric space and must be collapsed. This list is repeatedly traversed until it is empty.

In each traversal, the adjacent boundary layer stacks are checked for the layer edge collapse operation based on the shortest adjacent edges to a given growth curve. If all layered and interface regions are on one part, then the collapse operation is checked for validity and applied as in the serial case and the dynamic list of originating vertices to

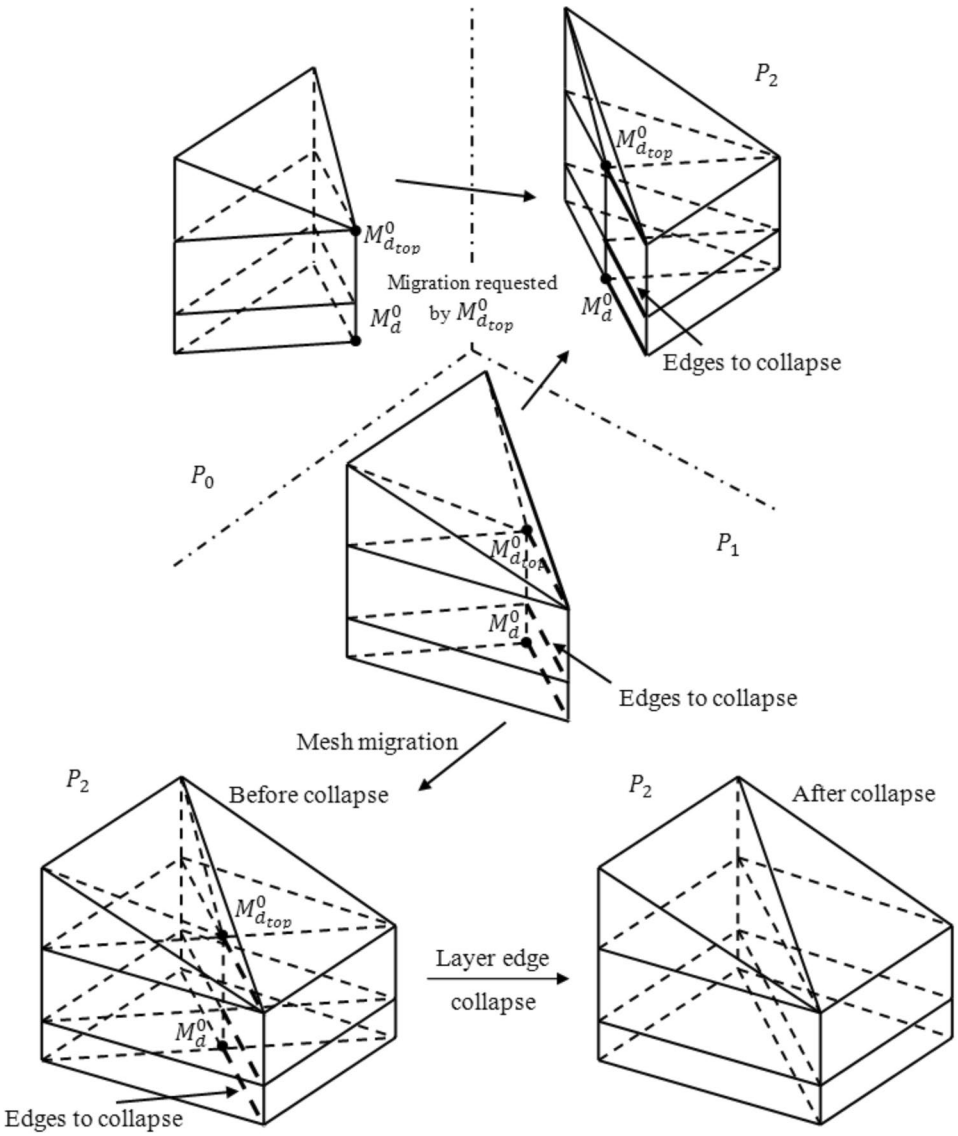

Fig. 16 Example of layer edge collapse in parallel involving mesh migration

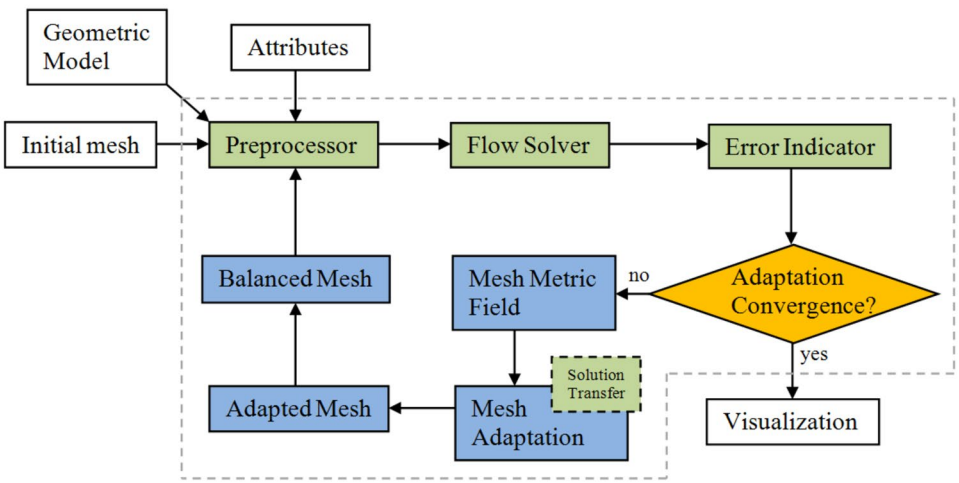

Fig. 17 A schematic of the adaptive loop with different simulation components

be collapsed is updated. Otherwise, the corresponding layer vertices on inter-part boundaries are added to the list of vertices to be migrated. After each traversal, the migration list is used to request migration of surrounding layered and interior regions. These requests then drive the mesh migration step and also updating of the on-part list of originating vertices to be collapsed.

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

```
determine the stacks with poorly shaped layer faces and collect the corresponding zero-level layer faces in list FaceListToOpt
while FaceListToOpt is not empty in all parts do
  for all faces M_k^2 in FaceListToOpt do
    check for the layer edge swap operation to apply on bounding edges
    if layered and interface regions associated with the swap operation are on one part then
      apply the operation and update FaceListToOpt
    else
      obtain the top most layer face M_{k_top}^2 (corresponding to M_k^2)
      add the top most vertices M_{j_top}^0 ∈ {∂M_{k_top}^2} into the list VtxToMigrate
    end if
  end for
  perform mesh migration based on the list VtxToMigrate
  update FaceListToOpt based on migration
end while
```

Parallelization of the layer edge swap operation follows the same overall logic as the layer edge collapse operation. Layer edge swaps are applied at the end of the boundary layer mesh adaptation procedure, i.e., within the mesh optimization step for the layered part of the mesh [53]. The only difference in such an optimization step is that it is driven by a traversal process focused on improving the shape quality of the layer faces. Algorithm 6 describes the surface optimization procedure for the parallel boundary layer mesh.

5 Results and discussions

5.1 Adaptive loops and applications

An adaptive loop is constructed using a set of interoperable components that includes analysis code along with libraries for geometry-based problem specification, automatic mesh generation, error estimation and generalized mesh modification (e.g., see [12, 58]). The adaptive loop links the analysis and adaptation components needed for the successful simulation of the problem on the domain of interest. The solution obtained by the analysis code is evaluated to provide information for mesh adaptation, which in-turn results in an adapted mesh that enriches the solution approximation. In this step, the error distribution is determined on the current mesh and is converted into a mesh metric or size field that is used to drive the adaptation procedure. In the adaptation procedure as the mesh is locally modified, the necessary solution fields are transferred onto the modified mesh. The resulting adapted mesh and associated solution fields are sent back to the analysis code to perform the next step of analysis and adaptation within the adaptive loop. The overall structure of the adaptive loop is shown in Fig. 17.

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

The parallel boundary layer mesh adaptation procedure for these cases has been executed on Hopper Cray XE6 [18] at the National Energy Research Scientific Computing Center. It is configured with 2 twelve-core AMD 2.1 GHz processors per node, with separate L3 caches and memory controllers, 32 GB or 64 GB DDR3 SDRAM per node. Hopper has a Gemini interconnect with a 3D torus topology. Note that all available processors or cores on a node were used in this work.

In this work, both strong and weak scalings are studied. The strong scaling (for a fixed size mesh in an aggregate sense) is computed based on the execution time on base processors and is defined as:

$$S_i^s = (np_{base} \times t_{base})/(np_i \times t_i), \tag{5}$$

where np_{base} is the base number of processors or cores, t_{base} is the execution time on base processors, np_i is the number of processors on which strong scaling is tested, and t_i is the execution time on test processors. A scaling factor of 1 indicates a perfect linear scaling (i.e., 100 % parallel efficiency) and a value below or above 1 denotes a sub-linear scaling (or below 100 % parallel efficiency) or super-linear scaling (or above 100 % parallel efficiency), respectively.

On the other hand, the weak scaling (with a fixed load per part) is computed using a correction factor. This is done to account for (slightly) different loads in an aggregate and average sense between different meshes considered under weak scaling. It is computed as:

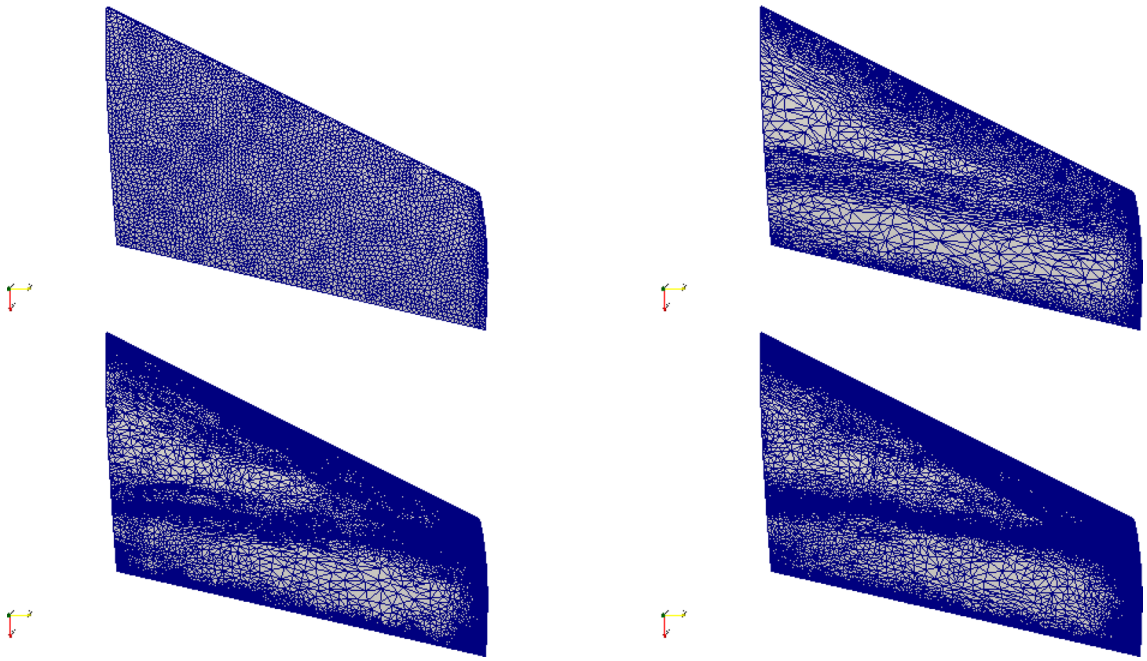

Fig. 18 ONERA M6 wing: initial (*top left*) and three adapted meshes (first and third adapted meshes are shown in the *right column*)

$$f = M_{incr-i}/M_{incr-base}, \tag{6}$$

$$S_i^w = f \times t_{base}/t_i, \tag{7}$$

where M_{incr} is the mesh increase factor defined by the ratio of the number of regions from input to adapted meshes for a test case and M_{incr-i} and $M_{incr-base}$ are the mesh increase factors for the test and base cases, respectively. f is a factor which is the ratio of the mesh increase factors of the test and base cases.

5.2 ONERA M6 wing

The ONERA M6 wing is a classic validation case [61]. Air enters the wind tunnel at transonic speed and is accelerated over the wing to a supersonic speed causing a shock to appear on the wing. The free-stream Mach number is 0.84 and the angle of attack is 3.06º. The free-stream pressure and temperature are 42.89 psi and 255.5 K, respectively. The Reynolds number is 11.72 million based on the mean aerodynamic chord. This flow marks a strong need for mesh adaptivity since the location and structure of the complex lambda shock is unknown a priori. The reference experimental data are from Schmitt and Charpin in [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 with pre-defined boundary layers (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 shows the surface mesh on the upper side of the wing for the initial and three adapt meshes. The imprint of the 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 Fig. 18) show that the mesh is refined in the shock region and the shape of the lambda shock is clearly captured. The mesh away from the shock is coarsened, due to a low variation in pressure in those regions. 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.

To perform a more quantitative comparison, we look at the 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 are also included [61]. These plots show that as the mesh is adapted, the pressure coefficient becomes more accurate. To establish this aspect further we look at a zoomed view near the suction peak in Fig. 21. The zoomed view clearly shows that the agreement between experimental and numerical results are improved as the mesh is adapted further. The finest or third adapted mesh shows 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

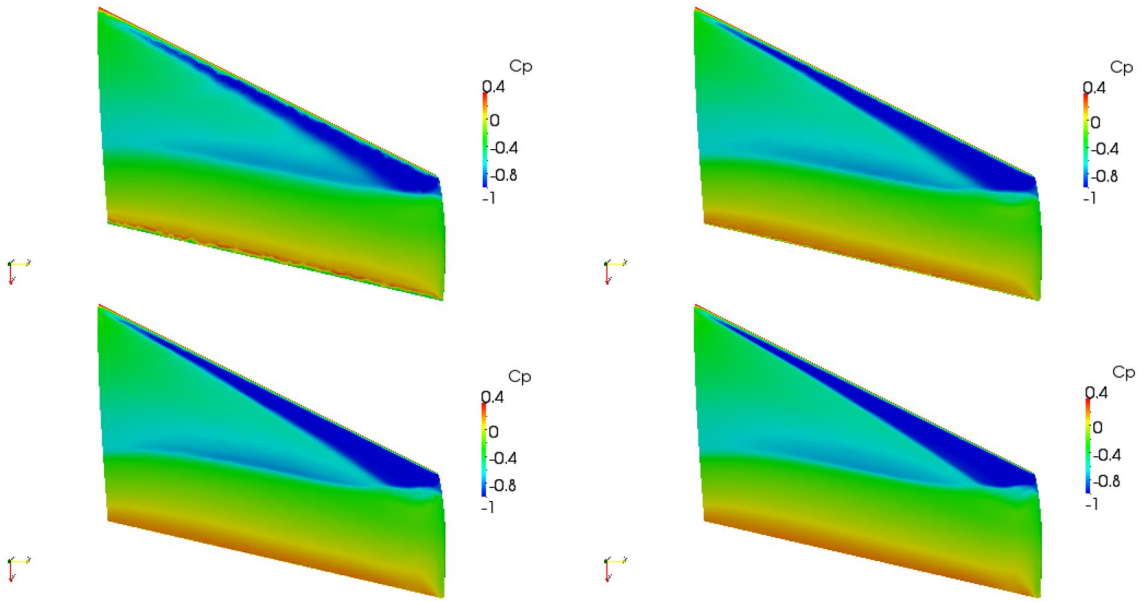

Fig. 19 ONERA M6 wing: pressure on initial (*top left*) and three adapted meshes (first and third adapted meshes are shown in the *right column*)

on the finest adapted mesh as compared to other adapted meshes. Results on the initial mesh are the least accurate among all meshes.

To evaluate the parallel performance of boundary mesh adaptivity, a strong scaling study was conducted. In this study we consider a refined mesh of the third adapted mesh resulting in 160M regions. Strong scaling study was executed on cores ranging from 512 (base) to 8192, which spans 4 doublings in core counts.

Table 1 shows that the execution time for mesh adaptation procedure decreases with an increase in the number of processors. As the given mesh is distributed to more processors, there is little computation performed during mesh modifications relative to the substantial increase in communications, and thus, the scaling decreases on high core counts.

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 simulations for this case were performed using the incompressible Reynolds-averaged Navier-Stokes (RANS) simulations with the Spalart-Allmaras turbulence model. 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 homogeneous natural pressure was prescribed at the outlet. In this case, the Hessian-based error indicator used the static pressure combined with a scaled dynamic pressure. This was defined as: $p + \alpha\rho u^2/2$, where the factor $\alpha = 0.2$ was chosen to attain an appropriate balance of static and dynamic pressure.

Two iterations or cycles of the adaptive loop (which consists of a flow solve and mesh adaptation within each cycle) were carried out, and at each cycle, flow solver was started from the solution of the previous cycle. The initial computation used a mesh of 3M elements with pre-defined boundary layers. The first adapted mesh had 16M regions and the second adapted boundary layer mesh had 81M regions. The initial mesh along with the first and second adapted meshes are shown in Fig. 22.

For this case we provide a qualitative assessment of the numerical results due to the lack of experimental or any reference data for comparison. The pressure distribution near the inlet pipe is provided in Fig. 23, whereas near the outlet pipe is presented in Fig. 24. The initial mesh is too coarse and these figures demonstrate its inability to capture the dominant flow features. Critical flow locations, including stagnation and turns around fillets of the pipes, get significantly refined. For example, see the smoother solution obtained on adapted meshes. The walls of the manifold, especially the wall closest to the inlet pipe, get refined to a higher degree. The fillets of the outlet pipes also get more refinement. The central part of the flat manifold gets relatively lesser refinement because of a relatively small variation in the solution. Moreover, away from flow regions with stagnation and turns, highly anisotropic mesh elements are created to effectively capture the anisotropy present in the flow. This results in significant computational savings over isotropic meshes of equivalent resolution.

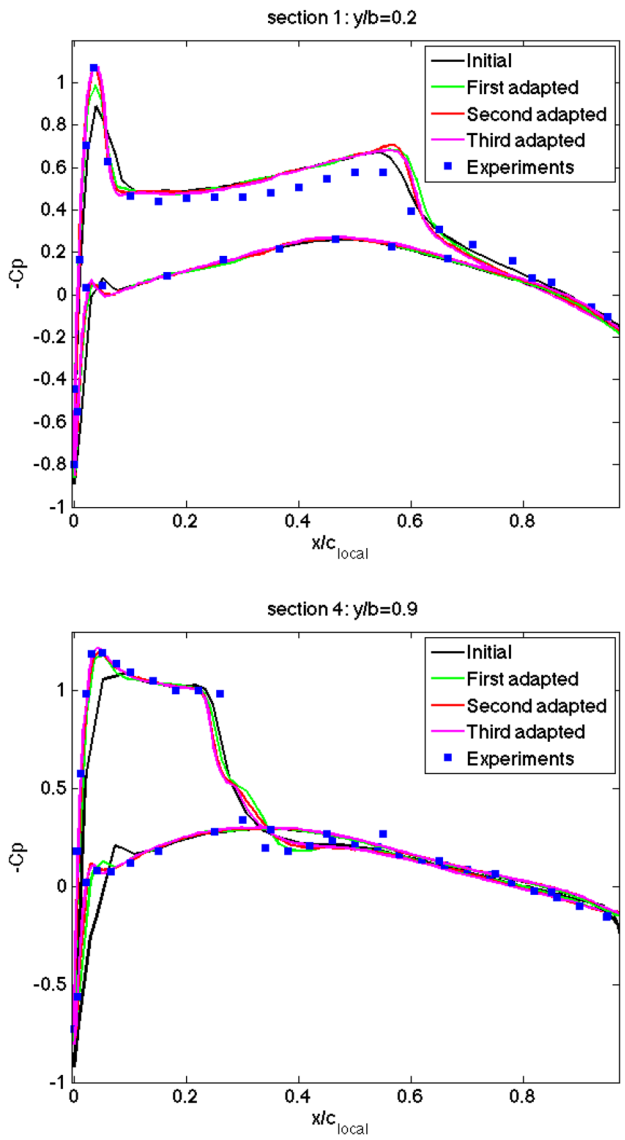

Fig. 20 Pressure coefficient profiles along the local chord on initial and three adapted meshes at two spanwise locations

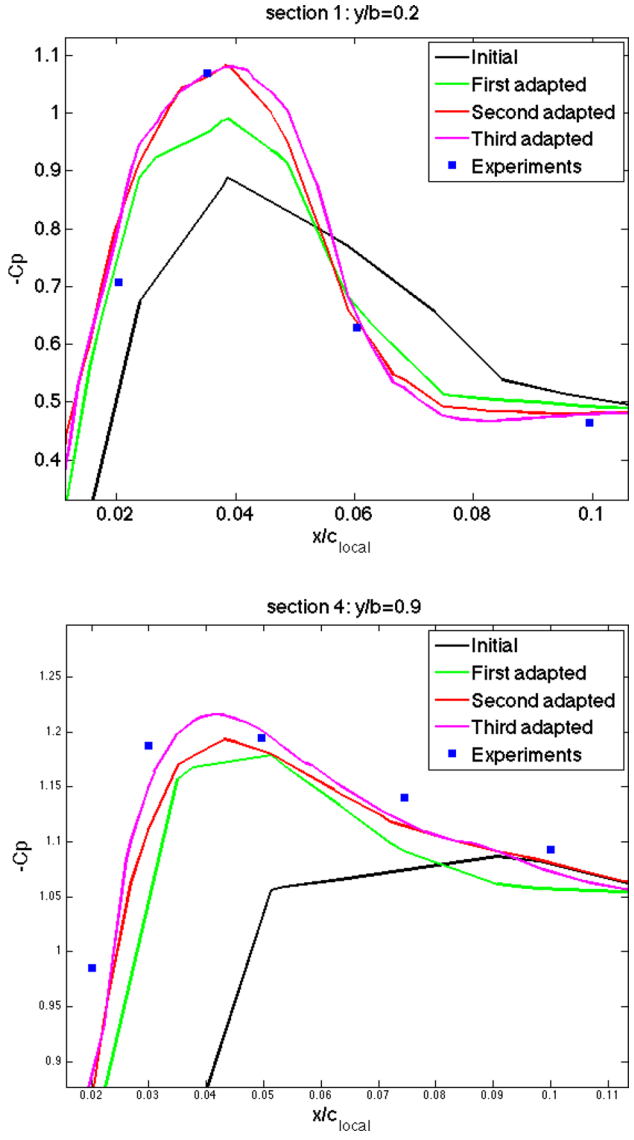

Fig. 21 A zoomed view of the pressure coefficient profiles (near the suction peak) on initial and three adapted meshes at two spanwise locations

Figure 25 shows the magnitude of wall shear stress. With adaptivity, smoothness of the wall shear stress field improves and its details are captured better. It has been shown in [53] that a wall shear stress field computed on an adapted boundary layer mesh is superior to that computed on a fully unstructured adapted mesh of a similar resolution.

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 further details on such mesh statistics for fully unstructured, anisotropically adapted meshes see [37, 49]). Currently mesh statistics were collected for the final adaptation cycle, where it was done for both the input mesh and the resulting adapted mesh. Figure 26 shows these statistics. It can be seen that the input mesh has a 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 within this interval indicating the satisfaction of the specified mesh metric field. Note that a large number of layer edges in the adapted mesh have a length (in the metric space) close to 0.5. This is due to the conservative nature of the split scheme used for layer edges (i.e., edge split of a single layer edge results in the split of all layer edges in that stack). Similarly, mean ratio plot shows that the shape quality measure of the elements in the adapted mesh is higher and respects the specified mesh metric field.

Table 1 Execution time and strong scaling of mesh adaptation for the ONERA M6 case

Num. cores (*np*)	512 (base)	1024	2048	4096	8192
Time (*t* in s)	1212.83	812.68	507.36	322.82	241.94
Scaling factor (S^s)	1	0.75	0.60	0.47	0.31

As before, a strong scaling study was conducted to evaluate the parallel performance of boundary mesh adaptation procedure on this test case. In this case, mesh adaptation in the second cycle of the adaptive loop was executed on a range of processors from 256 (base) to 4096, which spans 4 doublings in processor counts. Table 2 gives the scaling results for the input mesh with 16M regions and the final

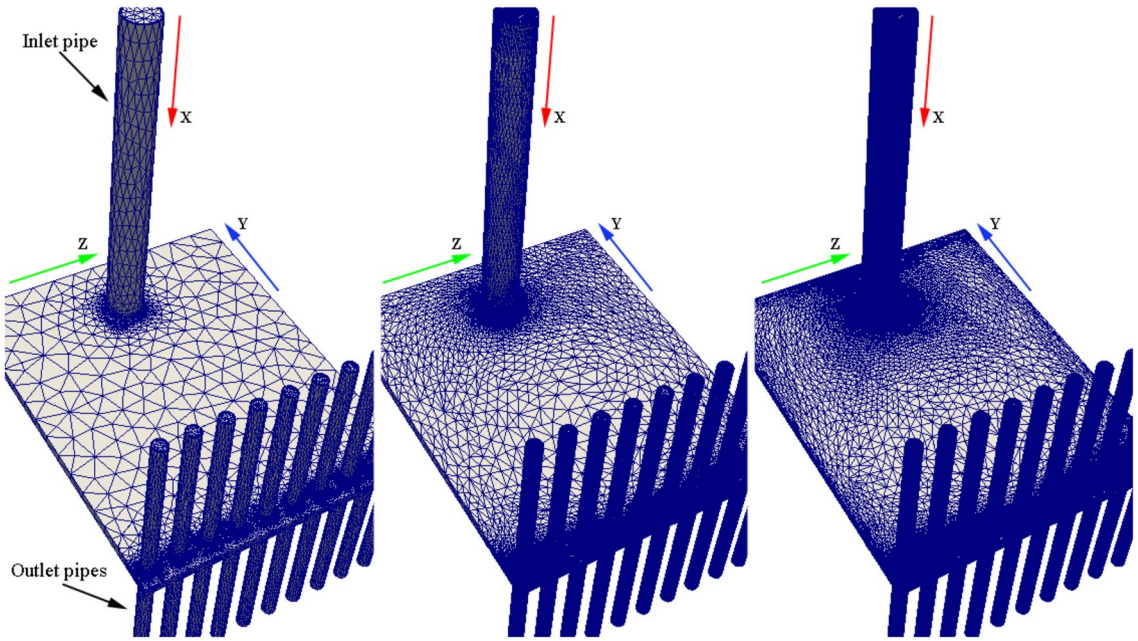

Fig. 22 Heat transfer manifold: initial (*left*), first adapted (*middle*) and second adapted (*right*) meshes

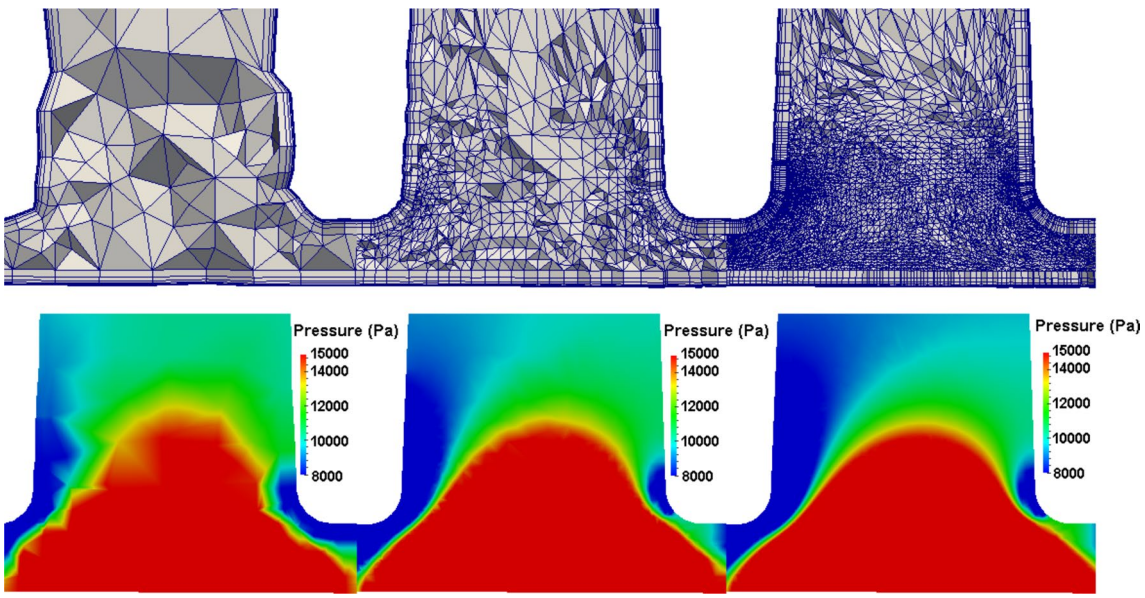

Fig. 23 Initial (*left*), first adapted (*middle*) and second adapted (*right*) meshes (in *top row*) and pressure distribution (in *bottom row*) for the heat transfer manifold test case. The cut is applied at the end of the inlet pipe near the flat section

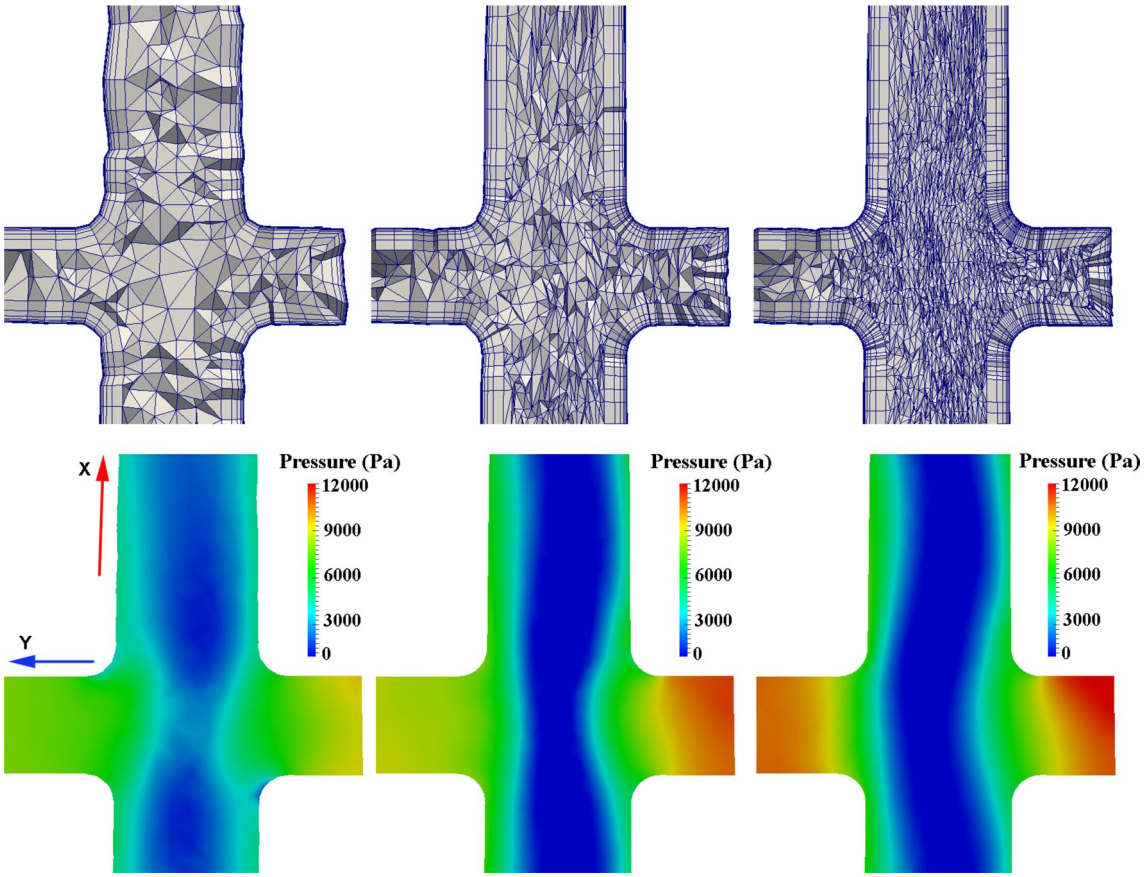

Fig. 24 Initial (*left*), first adapted (*middle*) and second adapted (*right*) meshes (in *top row*) and pressure distribution (in *bottom row*) for the heat transfer manifold test case. The cut is applied at an outlet pipe

adapted mesh with 81M regions. In Table 2 the execution time for mesh adaptation decreases with an increase in the number of cores. Again, on high core counts there is little computation performed during mesh modifications relative to the substantial increase in communications, and thus, the scaling decreases on high core counts.

The flow solver has been shown to strongly scale [55, 66] on a large number of cores, i.e., for a fixed size problem in an aggregate sense. It is the analysis part of a simulation which defines the number of processors on which the particular problem is being executed. The idea is to efficiently execute the mesh adaptation procedure on the same number of cores since repartitioning and migrating the mesh to a smaller number of cores for adaptation, and then again to a larger number after adaptation, will introduce a substantial amount of additional work and data movement.

In this case, the mesh adaptation procedure took 0.7 % of the total simulation time on 256 cores and 3.2 % on 4096 cores. In either case, mesh adaptation cost do not dominate as compared to the cost of the analysis step. Thus, even with some loss of strong scaling in the mesh adaptation step, it is reasonable to execute it on the same number of processors together with the flow solver.

A weak scaling study was also performed for this case on three uniformly refined meshes starting with the first mesh on 256 (base) cores. The second mesh was obtained by uniform refinement of the first mesh and the third mesh by uniform refinement of the second. The mesh metric field for the second and third meshes was constructed from that on the first mesh by multiplying it uniformly by a factor of 1 / 2 and 1 / 4, respectively. Due to the existence of the layered elements in the mesh the second and third 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, respectively. This roughly spans a factor of 36 in the number of mesh regions and core counts. Considering the heuristics employed in the mesh adaptation procedure and the fact that the ratio of regions in the test and base meshes is not precisely six, the weak scaling factor is calculated using a correction factor as discussed above. The weak scaling results are presented in Table 3.

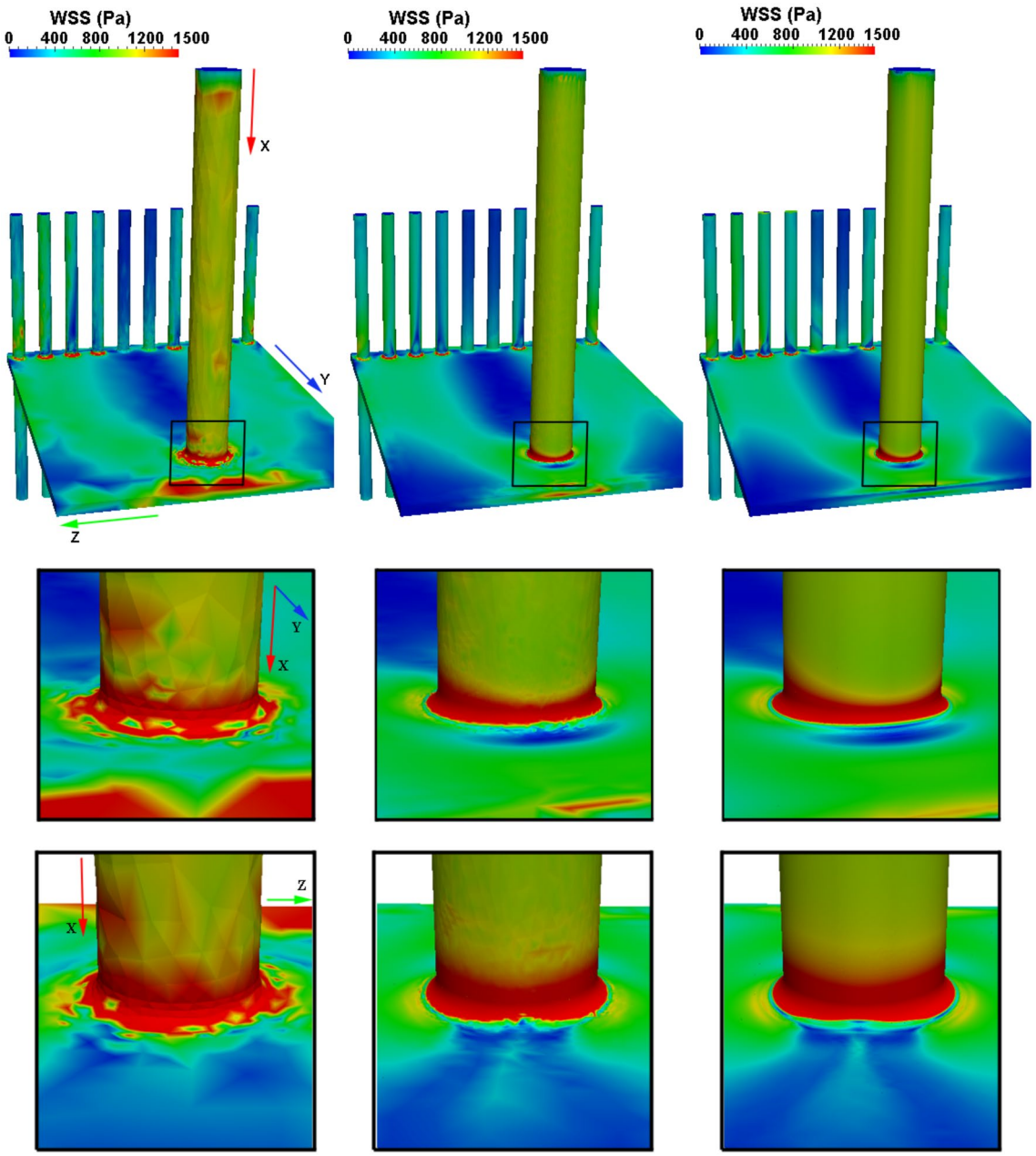

Fig. 25 Wall shear stress on the initial (*left*), first adapted (*middle*) and second adapted (*right*) meshes

Table 3 shows that the weak scaling does not degrade substantially with the increase in the number of cores while having relatively the same amount of workload in each test. The scaling is affected not only by the growing data exchange between cores at larger core counts, but also by the asynchronous application of mesh modification operations. Note that although the average workload is approximately the same per part, it might be very different for each specific part depending on the amount of different types of mesh modification operations applied locally on any given part and its neighboring parts sharing inter-part boundaries.

5.4 Scramjet engine

The NASA CIAM scramjet case [45] was setup with a free-stream Mach number of 6.2 and temperature of 203.5 K. The initial mesh had 2.86M regions. A cut of it is shown in Fig. 27 along with a zoomed view near the tip of the nose cone. Hessian calculations were based on the Mach number to compute the mesh size field.

Two adaptation cycles were carried out for this case. The first adapted mesh had 7.2M regions and the second adapted mesh consisted of 16M regions. Figure 28 presents

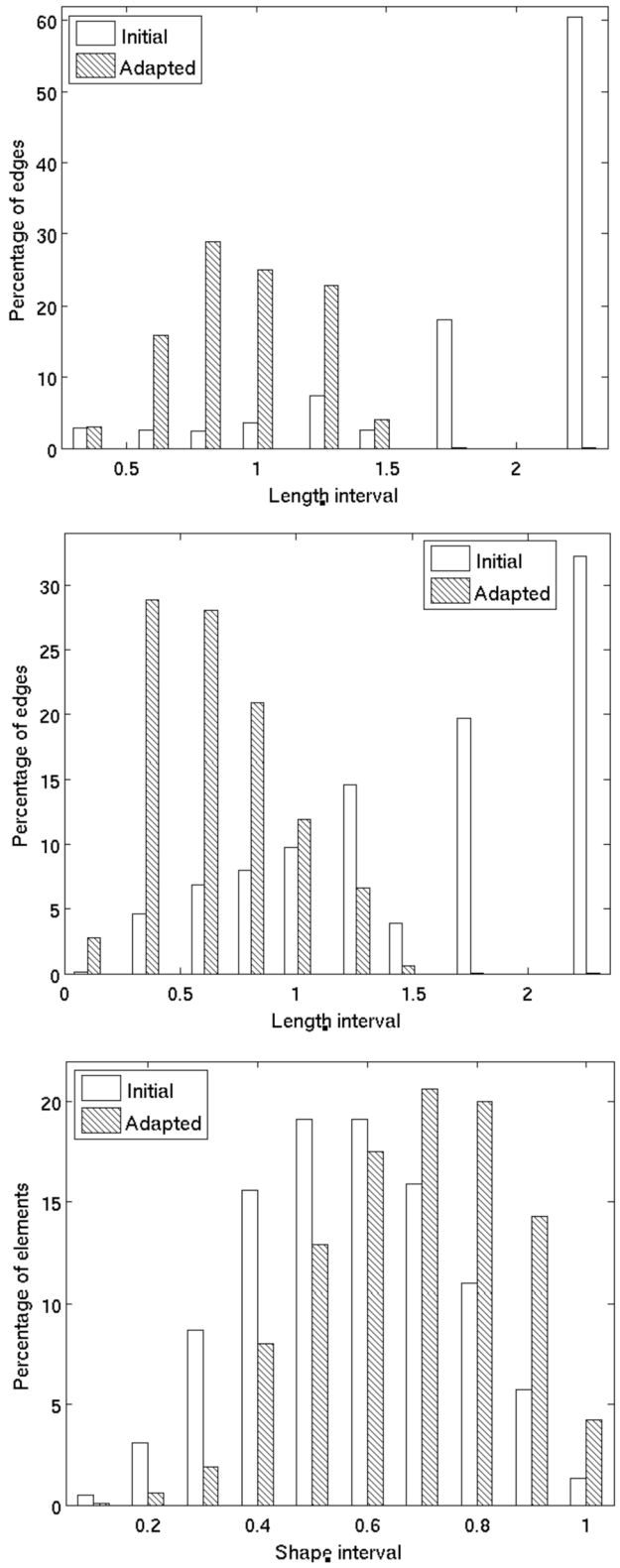

Fig. 26 Distribution of edge length (*left* plot for interior edges and *middle* plot for layer edges) and mean ratio (*right* plot) in the transformed space from the final adaptation cycle of the heat transfer manifold case

Table 2 Execution time and strong scaling of mesh adaptation for the heat transfer manifold case

Num. cores (*np*)	256 (base)	512	1024	2048	4096
Time (t in s)	1194.34	785.44	514.45	421.09	339.38
Scaling factor (S^s)	1	0.76	0.58	0.35	0.22

a cut of the first and second adapted meshes, whereas Fig. 29 shows a zoomed view near the nose cone. For this case also we provide a qualitative assessment of the numerical results. Figure 30 presents the Mach number contour plots on the initial mesh and the first and second adapted meshes. In addition, Fig. 31 shows adapted meshes and contours of the computed Mach number near the cowl lip region of the inlet to the combustor region.

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

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

In this case, an anisotropic mesh gradation procedure [36] is also used to reduce high variations in the mesh size around the tip of the nose cone. Figure 32 illustrates the impact of anisotropic gradation near the tip of the nose cone. The requested sizes at the wall surface are over an order of magnitude smaller than those in the unstructured region directly at the top of the boundary layer stacks. As a result, the boundary layer stack at the tip is much more refined than in the adjacent unstructured interior mesh, leading to a so-called "spider web" behavior and poorly shaped elements locally. As shown in Fig. 32, gradation of the mesh size field alleviates this issue.

Mesh statistics were collected for this case too. The same three quantities were collected in the final adaptation cycle. Figure 33 shows these statistics. It can be seen

Table 3 Execution time and weak scaling of mesh adaptation for the heat transfer manifold case

Num. cores (*np*)	256 (base)	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})	4.96	5.58	6.12
Time (t in s)	1194.34	1381.55	1716.23
Scaling factor (S^w)	1	0.97	0.86

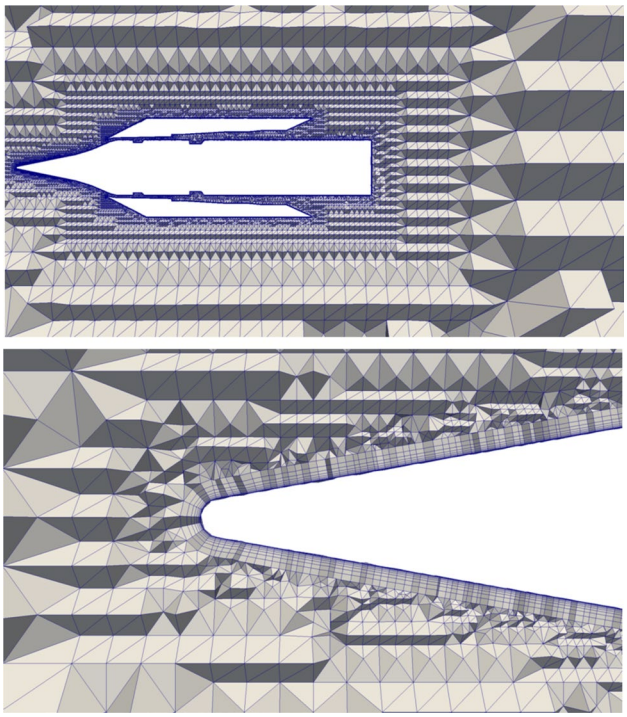

Fig. 27 Cut of the initial mesh for the scramjet case: whole body (*top*) and a zoomed view near the tip of the nose cone (*bottom*)

that the input mesh has a 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 in the metric space is higher in this case because the computed mesh size field has more variation along the thickness of the layered mesh (e.g., in stacks near the tip region of the nose cone). As before, mean ratio plot shows that the shape quality measure of the elements in the adapted mesh is higher and respects the specified mesh metric field.

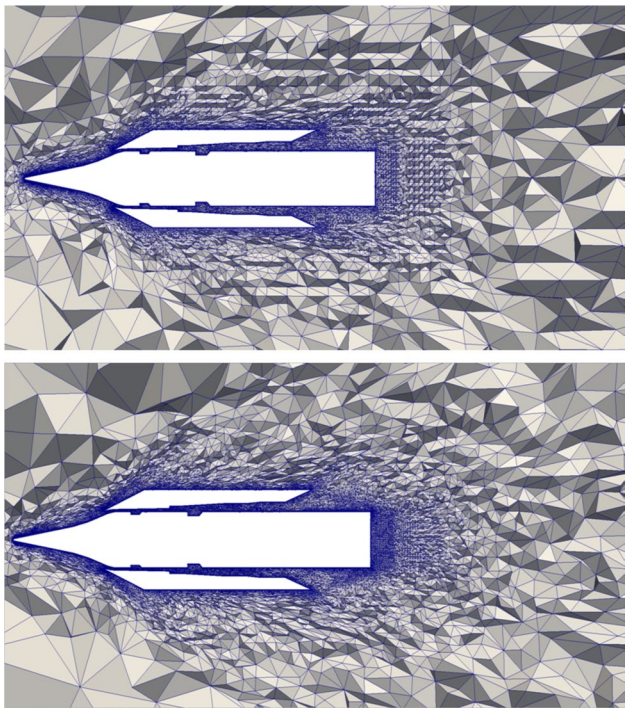

Fig. 28 Cut of the first (*top*) and second (*bottom*) adapted meshes for the scramjet case

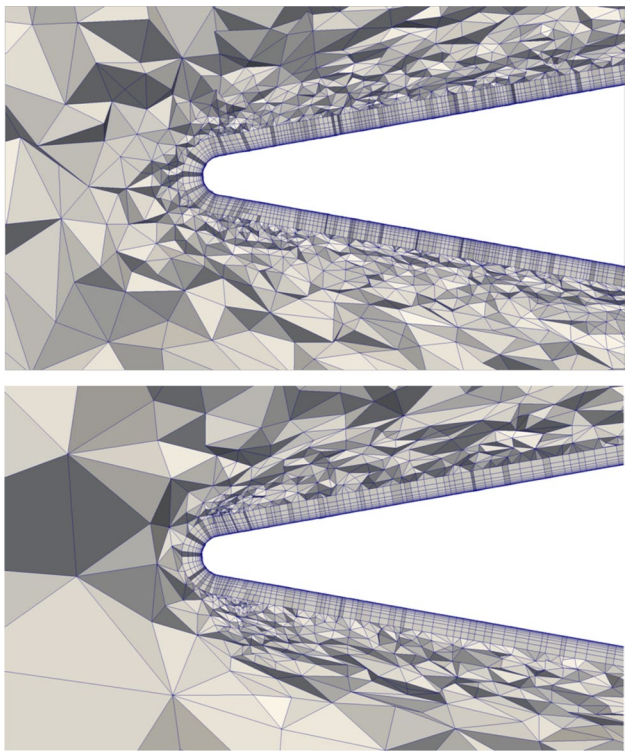

Fig. 29 Cut of the first (*top*) and second (*bottom*) meshes for the scramjet case near the nose

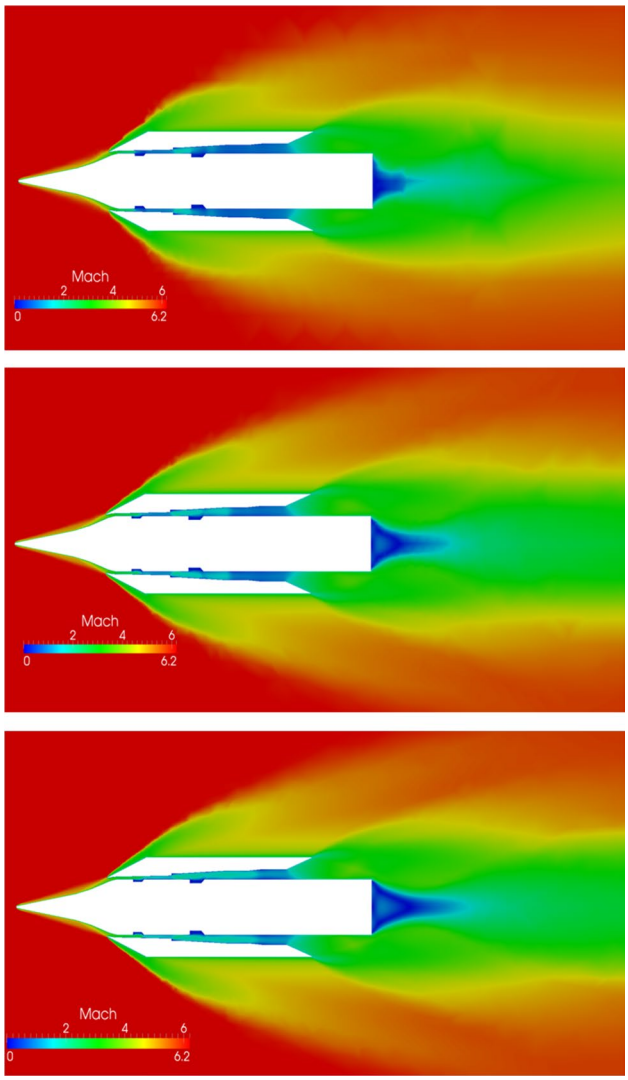

Fig. 30 Mach contours on the initial (*top*), first adapted (*middle*) and second adapted (*bottom*) meshes

Table 4 provides execution time and scalability of the mesh adaptation procedure based on the second adapted mesh. The strong scaling studies were performed on cores ranging from 128 (base) to 4096, which spans 5 doublings in core counts.

Table 4 shows that the execution time for mesh adaptation procedure is decreased on larger number of cores. Even though the mesh is relatively smaller for this case as compared to the heat transfer manifold case, the parallel scalability is better. Note that the mesh adaptation procedure took 1.2 % of the total simulation time on 128 cores and 3.6 % on 4096 cores, which is not significant when compared to the analysis time. As in the previous cases, even for a fixed size problem, mesh adaptation procedure is able to perform effectively on high core counts.

Table 5 gives weak scaling results for the scramjet engine case, where the scaling factor is calculated the same way it was done for the heat transfer manifold case. It can be seen in Table 5, with a relatively equal amount of workload per part in each test, the drop in weak scaling is modest as the core count is increased. In contrast to the strong scaling results, the weak scaling is better for the heat transfer manifold case as compared to the scramjet case. Note that the mesh increase factors are less in the scramjet case by roughly 2× as compared to the heat transfer manifold case. As noted earlier, the parallel performance of the mesh adaptation procedure is specific to the types of mesh modification operations carried out on different parts and on the communication-to-computation ratio.

6 Closing remarks

In this paper, a parallel adaptive boundary layer meshing procedure is presented. The approach successfully works on distributed meshes and effectively supports layered structure in the mesh. It is based on local mesh modification operations which are carried out in parallel and dictated by the specified mesh size field. The current parallelization paradigm allows the adaptation procedure to be applied on large and complex problem cases (e.g., on meshes with billions of regions).

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

The parallel performance of the mesh adaptation procedure for these problems showed that the execution time decreases with an increase in the number of cores for a fixed size problem or under strong scaling (e.g., with 5 doublings in core counts). Weak scaling was also presented showing that the procedure is capable to scale for larger meshes on high core counts (e.g., spanning a factor of 36 in the number of mesh regions and core counts). With mesh adaptation taking a small fraction of the total simulation time within the adaptive loop, parallel boundary layer mesh adaptation can be effectively integrated into workflows to support large-scale automated flow simulations of complex problems on high core counts. In the future, we plan to include cases where change in number of layers is required

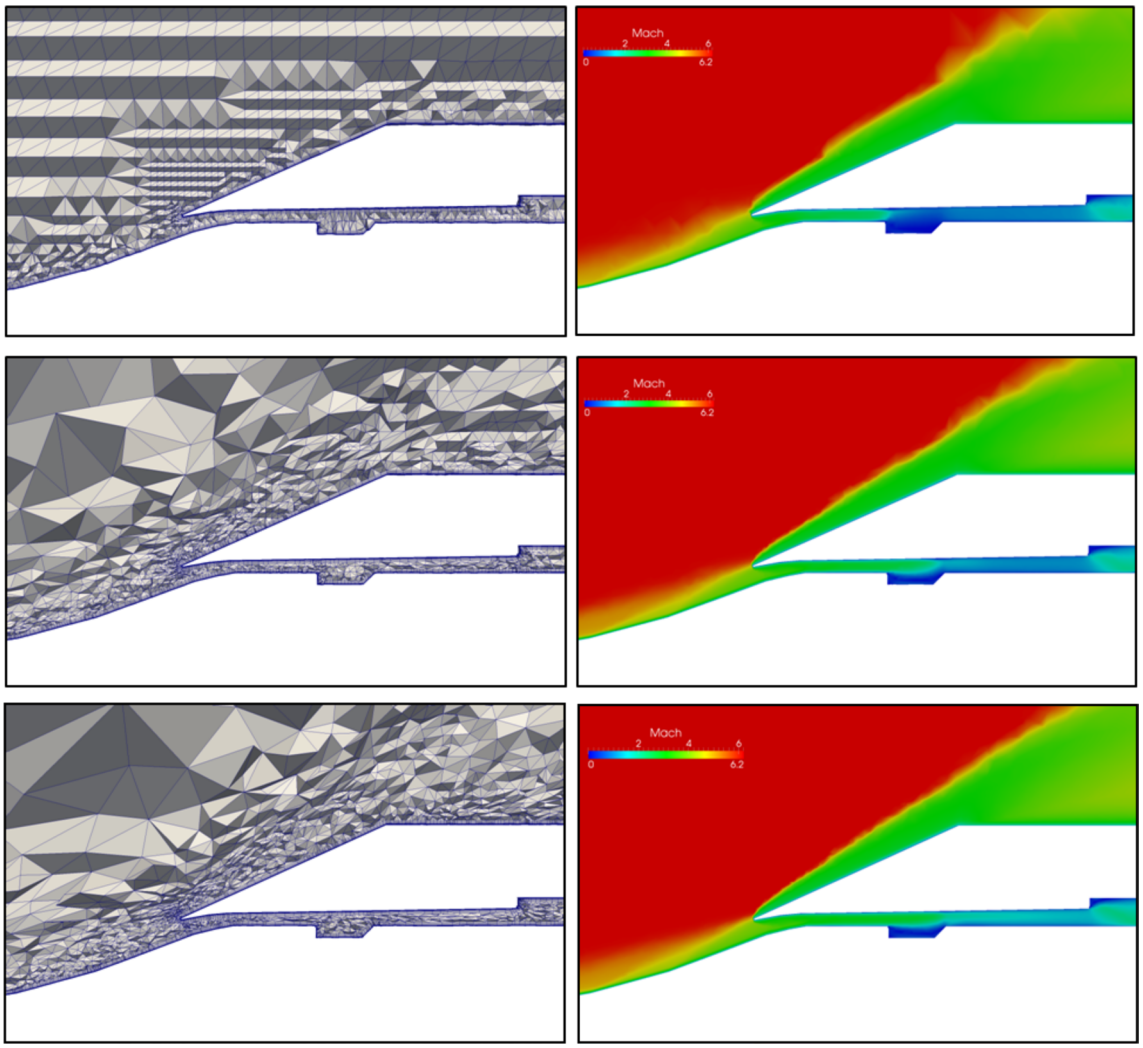

Fig. 31 Initial (*top*), first adapted (*middle*) and second adapted (*bottom*) meshes (*left column*) and Mach number contours (*right column*) for the scramjet case near the cowl lip and at the inlet to the combustor region

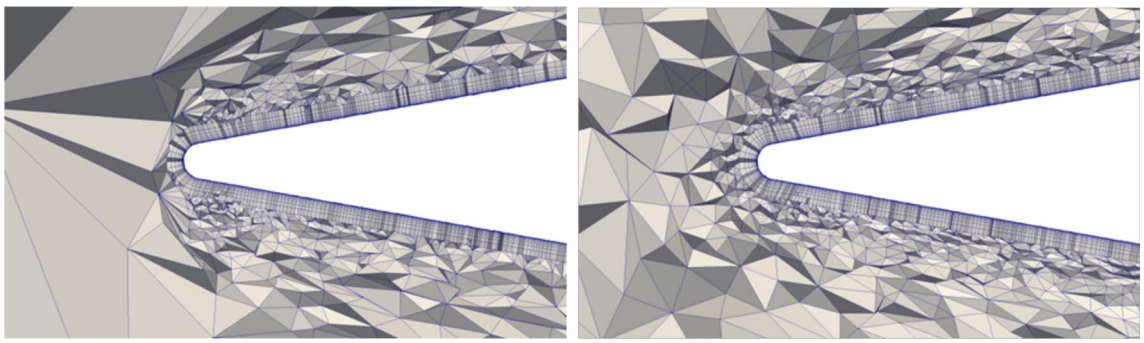

Fig. 32 Anisotropic gradation near the nose cone for the scramjet case: without (*left*) and with (*right*) gradation

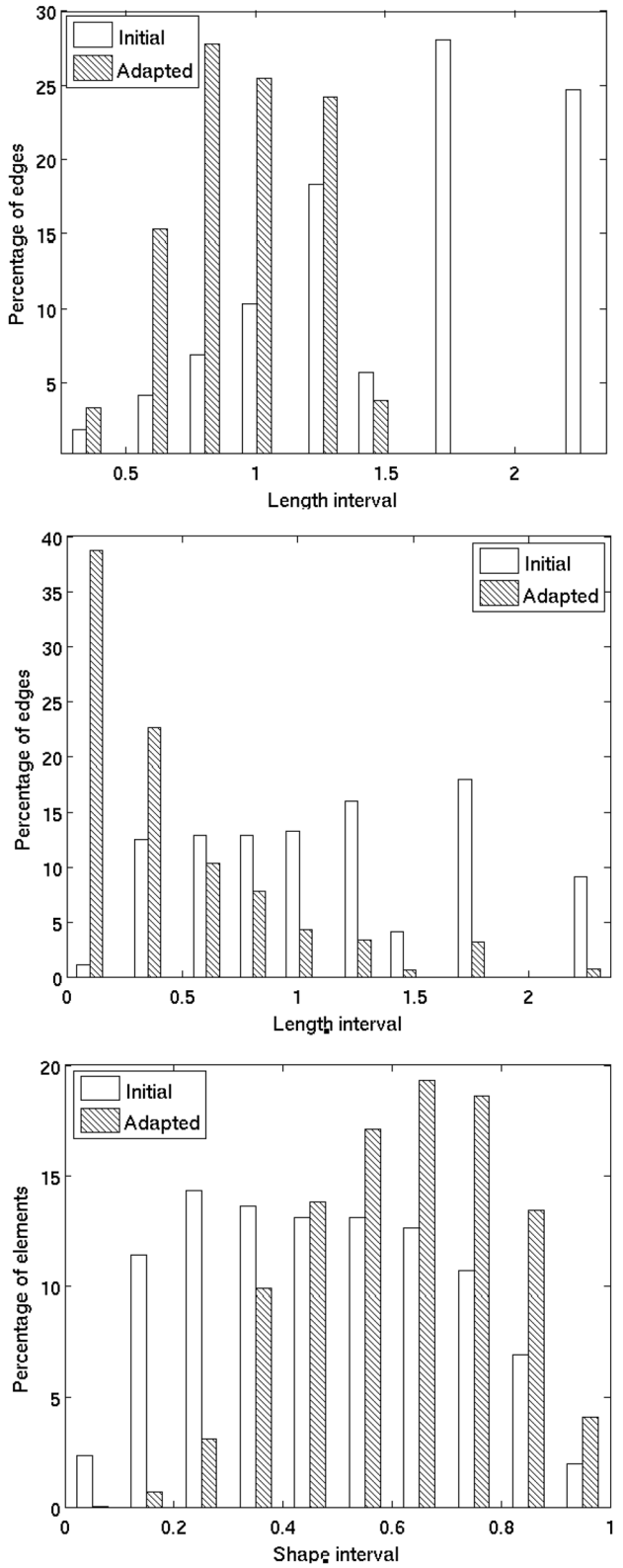

Fig. 33 Distribution of edge length (*left* plot for interior edges and *middle* plot for layer edges) and mean ratio (*right* plot) in the transformed space from the final adaptation cycle of the scramjet case

Table 4 Execution time and strong scaling of mesh adaptation for the scramjet case

Num. cores (*np*)	128 (base)	256	512	1024	2048	4096
Time (*t* in s)	957.80	532.29	339.39	202.17	136.83	90.85
Scaling factor (S^s)	1	0.90	0.71	0.59	0.44	0.33

Table 5 Execution time and weak scaling of mesh adaptation for the scramjet case

Num. cores (*np*)	256 (base)	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 (*t* in s)	532.29	606.12	734.82
Scaling factor (S^w)	1	0.95	0.81

and far-field solution features (e.g., far-field shocks) play an important role.

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 the National Energy Research Scientific Computing Center for granting access to the Hopper Cray XE6 supercomputer. Resources at the Center for Computational Innovations (CCI) at Rensselaer were also used for testing and development. The authors would like to acknowledge the help of Dr. L. Fovargue on the ONERA M6 case and F. Nihan Cayan and O. Breslouer for help with the scramjet case.

References

1. Gropp W, Lusk E, Skjellum A (2014) Using MPI: portable parallel programming with the message-passing interface. MIT Press, Cambridge. https://mitpress.mit.edu/using-MPI-3ed
2. Ainsworth M, Oden JT (2000) A posteriori error estimation in finite element analysis. Wiley, New York
3. Alauzet F, Li X, Seol ES, Shephard MS (2006) Parallel anisotropic 3D mesh adaptation by mesh modification. Eng Comput 21(3):247–258. doi:10.1007/s00366-005-0009-3
4. Au P, Dompierre J, Labbe P, Guibault F, Camarero R (1998) Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization. In: Proceedings of the 7th International Meshing Roundtable, pp 459–478
5. Bänsch E (1991) Local mesh refinement in 2 and 3 dimensions. IMPACT Comput Sci Eng 3(3):181–191
6. Becker R, Rannacher R (2001) An optimal control approach to a posteriori error estimation in finite element methods. Acta Numer 10(1):1–102

7. Botasso CL (2004) Anisotropic mesh adaption by metric-driven optimization. Int J Numer Methods Eng 60(3):597–639. doi:10.1002/nme.977
8. Bottasso CL, Detomi D (2002) A procedure for tetrahedral boundary layer mesh generation. Eng Comput 18(1):66–79. doi:10.1007/s003660200006
9. Bourgault Y, Picasso M, Alauzet F, Loseille A (2009) On the use of anisotropic a posteriori error estimators for the adaptative solution of 3D inviscid compressible flows. Int J Numer Methods Fluids 59(1):47–74. doi:10.1002/fld.1797
10. Buscaglia GC, Dari EA (1997) Anisotropic mesh optimization and its application in adaptivity. Int J Numer Methods Eng 40:4119–4136
11. Castro-Diáz MJ, Hecht F, Mohammadi B, Pironneau O (1997) Anisotropic unstructured mesh adaption for flow simulations. Int J Numer Methods Fluids 25:475–491
12. Chand KK, Diachin LF, Li X, Ollivier-Gooch C, Seol ES, Shephard MS, Tautges T, Trease H (2008) Toward interoperable mesh, geometry and field components for pde simulation development. Eng Comput 24(2):165–182. doi:10.1007/s00366-007-0080-z
13. Chandra S, Li X, Saif T, Parashar M (2007) Enabling scalable parallel implementations of structured adaptive mesh refinement applications. J Supercomput 39(2):177–203. doi:10.1007/s11227-007-0110-z
14. Chitale K, Sahni O, Tendulkar S, Nastasia R, Shephard M, Jansen K (2013) Boundary layer adaptivity for transonic turbulent flows. AIAA Paper 13-2445. doi:10.2514/6.2013-2445
15. Connell SD, Braaten ME (1995) Semistructured mesh generation for three-dimensional Navier–Stokes calculations. AIAA J 33(6):1017–1024
16. de Cougny HL, Shephard MS (1999) Parallel refinement and coarsening of tetrahedral meshes. Comput Methods Appl Mech Eng 46:1101–1125
17. de Cougny HL, Shephard MS, Georges MK (1990) Explicit node point mesh smoothing within the octree mesh generator. Tech. Rep. 1990-10, Rensselaer Polytechnic Institute, Troy
18. Cray: Cray XE6. [Online]. http://www.cray.com/Products/XE/CrayXE6System.aspx. Accessed 19 Sep 2012
19. Foster TM, Mohamed MS, Trevelyan J, Coates G (2012) Rapid re-meshing and re-solution of three-dimensional boundary element problems for interactive stress analysis. Eng Anal Bound Elem 36(9):1331–1343. doi:10.1016/j.enganabound.2012.02.020
20. Freitag LA, Ollivier-Gooch C (1997) Tetrahedral mesh improvement using swapping and smoothing. Int J Numer Methods Eng 40(21):3979–4002
21. Frey PJ, Alauzet F (2005) Anisotropic mesh adaptation for CFD computations. Comput Methods Appl Mech Eng 194(48–49):5068–5082. doi:10.1016/j.cma.2004.11.025
22. Garimella RV, Shephard MS (2000) Boundary layer mesh generation for viscous flow simulations. Int J Numer Methods Eng 49:193–218
23. George P, Borouchaki H, Laug P (2002) An efficient algorithm for 3D adaptive meshing. Adv Eng Softw 33(7):377–387
24. Giles MB, Süli E (2002) Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality. Acta Numer 11(1):145–236
25. Hassan O, Morgan K, Probert EJ, Peraire J (1996) Unstructured tetrahedral mesh generation for three-dimensional viscous flows. Int J Numer Methods Eng 39:549–567
26. Hassan O, Morgan K, Weatherill N (2007) Unstructured mesh methods for the solution of the unsteady compressible flow equations with moving boundary components. Philos Trans R S A Math Phys Eng Sci 365(1859):2531–2552
27. Ito Y, Murayama M, Yamamoto K, Shih AM, Soni BK (2013) Efficient hybrid surface/volume mesh generation using suppressed marching-direction method. AIAA J 51(6):1450–1461
28. Ito Y, Nakahashi K (2002) Unstructured mesh generation for viscous flow computations. In: Proceedings of the 11th International Meshing Roundtable, pp 367–377
29. Joe B (1995) Construction of three-dimensional improved-quality triangulations using local transformations. SIAM J Sci Comput 16(6):1292–1307
30. Kallinderis Y, Kavouklis C (2005) A dynamic adaptation scheme for general 3-D hybrid meshes. Comput Methods Appl Mech Eng 194(48–49):5019–5050. doi:10.1016/j.cma.2004.11.023
31. Kallinderis Y, Vijayan P (1993) Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. AIAA J 31(8):1440–1447
32. Kavouklis C, Kallinderis Y (2010) Parallel adaptation of general three-dimensional hybrid meshes. J Comput Phys 229(9):3454–3473. doi:10.1016/j.jcp.2010.01.011
33. Khawaja A, Kallinderis Y (2000) Hybrid grid generation for turbomachinery and aerospace applications. Int J Numer Methods Eng 49(1–2):145–166
34. Khawaja A, Minyard T, Kallinderis Y (2000) Adaptive hybrid grid methods. Comput Methods Appl Mech Eng 189(4):1231–1245. doi:10.1016/S0045-7825(99)00375-8
35. Li X (2003) Mesh modification procedures for general 3D non-manifold domains. Ph.D. Dissertation, Department of Mechanical Engineering, Rensselaer Polytechnic Institute, Troy
36. Li X, Remacle JF, Chevaugeon N, Shephard MS (2004) Anisotropic mesh gradation control. In: Proceedings of the 13th International Meshing Roundtable, pp 401–412
37. Li X, Shephard M, Beall M (2005) 3D anisotropic mesh adaptation by mesh modifications. Comput Methods Appl Mech Eng 194(48–49):4915–4950. doi:10.1016/j.cma.2004.11.019
38. Li X, Shephard MS, Beall MW (2003) Accounting for curved domains in mesh adaptation. Int J Numer Methods Eng 58(2):247–276. doi:10.1002/nme.772
39. Liu A, Joe B (1994) On the shape of tetrahedra from bisection. Math Comput 63(207):141–154
40. Löhner R, Baum JD (1992) Adaptive h-refinement on 3D unstructured grids for transient problems. Int J Numer Methods Fluids 14(12):1407–1419
41. Lohner R, Cebral J (2000) Generation of non-isotropic unstructured grids via directional enrichment. Int J Numer Methods Eng 49(1–2):219–232
42. Loseille A, Löhner R (2013) Robust boundary layer mesh generation. In: Proceedings of the 21st International Meshing Roundtable, pp 493–511
43. Marcum DL (1995) Generation of unstructured grids for viscous flow applications. AIAA Paper 95-0212
44. Muller J, Sahni O, Li X, Jansen KE, Shephard MS, Taylor CA (2005) Anisotropic adaptive finite element method for modeling blood flow. Comput Methods Biomech Biomed Eng 8(5):295–305. doi:10.1080/10255840500264742
45. NASA: CIAM Axisymmetric Scramjet. [Online]. http://hapb-www.larc.nasa.gov/Public/Engines/Ciam/Ciam.html. Accessed 19 Sep 2012
46. NASA: FUN3D online manual. [Online]. http://fun3d.larc.nasa.gov/. Accessed 19 Sep 2012
47. Oliker L, Biswas R, Gabow HN (2000) Parallel tetrahedral mesh adaptation with dynamic load balancing. Parallel Comput 26(12):1583–1608
48. Ovcharenko A, Ibanez D, Delalondre F, Sahni O, Jansen KE, Carothers CD, Shephard MS (2012) Neighborhood communication paradigm to increase scalability in large-scale dynamic scientific applications. Parallel Comput 38(3):140–156. doi:10.1016/j.parco.2011.10.013
49. Pain CC, Umpleby AP, de Oliveira CRE, Goddard AJH (2001) Tetrahedral mesh optimization and adaptivity for steady-state and transient finite element calculations. Comput

Methods Appl Mech Eng 190(29–30):3771–3796. doi:10.1016/S0045-7825(00)00294-2
50. Park YM, Kwon OJ (2005) A parallel unstructured dynamic mesh adaptation algorithm for 3-D unsteady flows. Int J Numer Methods Fluids 48(6):671–690
51. Peraire J, Peiro J, Morgan K (1992) Adaptive remeshing for three-dimensional compressible flow computation. J Comput Phys 103(2):269–285. doi:10.1016/0021-9991(92)90401-J
52. Pirzadeh S (1994) Unstructured viscous grid generation by the advancing-layers method. AIAA J 32(8):1735–1737
53. Sahni O, Jansen KE, Shephard MS, Taylor CA, Beall MW (2008) Adaptive boundary layer meshing for viscous flow simulations. Eng Comput 24(3):267–285. doi:10.1007/s00366-008-0095-0
54. Sahni O, Muller J, Jansen KE, Shephard MS, Taylor CA (2006) Efficient anisotropic adaptive discretization of the cardiovascular system. Comput Methods Appl Mech Eng 195(41–43):5634–5655. doi:10.1016/j.cma.2005.10.018
55. Sahni O, Zhou M, Shephard MS, Jansen KE (2009) 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
56. Sandia National Laboratories: Zoltan unstructured communication utilities. [Online]. http://www.cs.sandia.gov/Zoltan/ug_html/ug_util_comm.html. Accessed 19 Sep 2012
57. Seol ES, Shephard MS (2006) Efficient distributed mesh data structure for parallel automated adaptive analysis. Eng Comput 22(3–4):197–213
58. Shephard MS, Beall MW, O'Bara RM, Webster BE (2004) Toward simulation-based design. Finite Elem Anal Design 40(12):1575–1598. doi:10.1016/j.finel.2003.11.004
59. Stogner RH, Carey GF, Murray BT (2008) Approximation of Cahn–Hilliard diffuse interface models using parallel adaptive mesh refinement and coarsening with C1 elements. Int J Numer Methods Eng 76(5):636–661. doi:10.1002/nme.2337
60. Toussaint GT, Verbrugge C, Wang C, Zhu B (1993) Tetrahedralization of simple and non-simple polyhedra. In: Proceedings of the 5th Canadian Conference on Computational Geometry, pp 24–29
61. Schmitt V, Charpin F (1979) Pressure distribution on the ONERA-M6-Wing at transonic mach numbers. In: Report of the Fluid Dynamics Panel Working Group 04, vol 138. AGARD
62. Verfürth R (1996) A review of posteriori error estimation and adaptive mesh-refinement techniques. Teubner-Wiley, Stuttgart
63. Whiting CH, Jansen KE (2001) A stabilized finite element method for the incompressible Navier–Stokes equations using a hierarchical basis. Int J Numer Methods Fluids 35(1):93–116
64. Xie T, Seol ES, Shephard MS (2012) Generic components for petascale automated adaptive simulations. Eng Comput **(Accepted for publication)**
65. Zhang L, Chang X, Duan X, Zhao Z, He X (2012) Applications of dynamic hybrid grid method for three-dimensional moving/deforming boundary problems. Comput Fluids 62:45–63. doi:10.1016/j.compfluid.2012.03.008
66. Zhou M, Sahni O, Kim HJ, Figueroa CA, Taylor CA, Shephard MS, Jansen KE (2010) Cardiovascular flow simulation at extreme scale. Comput Mechan 46(1):71–82. doi:10.1007/s00466-009-0450-z