Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24rd International Meshing Roundtable (IMR24)

Parallel generation of large-size adapted meshes

Adrien Loseille[a,*], Victorien Menier[a], Frédéric Alauzet[a]

[a]*Gamma3 Team, INRIA Paris-Roquencourt, Le Chesnay, France*

Abstract

We devise a strategy in order to generate large-size adapted anisotropic meshes $O(10^8-10^9)$ as required in many fields of application in scientific computing. We target moderate scale parallel computational resources as typically found in R&D units where the number of cores ranges in $O(10^2 - 10^3)$. Both distributed and shared memory architectures are handled. Our strategy is based on typical domain splitting algorithm to remesh the partitions in parallel. Both the volume and the surface mesh are adapted simultaneously and the efficiency of the method is independent of the complexity of the geometry. The originality of the method relies on (i) a metric-based static load-balancing, (ii) dedicated mesh partitioning techniques to (re)split the (complex) interfaces meshes, (iii) anisotropic Delaunay cavity to define the interface meshes, (iv) a fast, robust and generic sequential cavity-based mesh modification kernel, and (v) out-of-core storing of completing parts to reduce the memory footprint. We show that we are able to generate (uniform, isotropic and anisotropic) meshes with more than 1 billion tetrahedra in less than 20 minutes on 120 cores. Examples from Computational Fluid Dynamics (CFD) simulations are also discussed.
© 2015 The Authors. Published by Elsevier Ltd.
Peer-review under responsibility of organizing committee of the 24rd International Meshing Roundtable (IMR24).

Keywords: Surface remeshing, Anisotropic Mesh Adaptation, Cavity-based primitives, Out-of-core, Parallel meshing, Domain partitioning, coarse-grained parallelization

1. Introduction

Complex numerical simulations (turbulence, noise propagation, ...) may require billions of degree of freedom to get a high-fidelity prediction of the physical phenomena. To fit this need, many numerical platforms (numerical solver, solution visualization) have been developed for parallel architectures (distributed or shared-memory). Although few simulations are performed on thousands of processors, recent studies show that many relevant R&D applications run on a daily basis on smaller architectures targeting less than 1 000 cores [1,10]. In the computational pipeline, mesh generation or adaptation is a critical point as the existence of a mesh (especially with complex geometries) is the necessary condition to start a simulation. In addition, the mesh generation CPU time should be low enough in comparison with the solver CPU time to be actually used in practice. In this paper, we aim at designing an efficient parallel adaptive mesh generation strategy. We target to generate adapted meshes composed of a billion elements in less than 20min on 120 cores. The parallelization of the meshing/remeshing step is a complex problem because it encompasses the following issues: domain partitioning, load balancing, robust surface and volume mesh adaptation.

* Corresponding author. Tel.: +33 1 39 63 55 39 ; fax: +33 1 39 63 58 82.
E-mail address: Adrien.Loseille@inria.fr

1877-7058 © 2015 The Authors. Published by Elsevier Ltd.
Peer-review under responsibility of organizing committee of the 24rd International Meshing Roundtable (IMR24).

Parallel mesh generation has been an active field of research [11,14,19,29]. Two main frames of parallelization exist: coarse-grained [8,16,19], and fined-grained [6,11,27,28]. Fine-grained parallelization requires to implement directly in parallel all the mesh modification operators at the lowest level: insertion, collapse, swap... This usually implies the use of specific data structures to handle distributed dynamic meshes, especially for adaptive procedures [2]. The second approach consists in the use of a bigger set of operators in parallel. Most of the time a complete sequential mesh generator or mesh optimizer is used. This approach was also extended to adaptive frameworks [8,16]. In this paper, we follow the coarse-grained parallelization in an adaptive context. In particular, we address the following problematics.

Surface-volume problematic. When considering the coarse-grained strategy, parallel mesh generators or parallel local remeshers generally adapt either the surface or the volume mesh. In [16,19], the fine surface mesh is unchanged during the parallel meshing process. When anisotropic meshes are used, being able to adapt the surface and the volume into a single thread is necessary to gain in robustness [23]. However, adapting both the surface and the volume meshes at the same time induces additional complexity for the load balancing as the costs of the volume or surface operators differ.

Domain partitioning. Domain partitioning is a critical task as each partition should represent an equal level of work. Graph-based techniques [15] tend to minimize the size of the cuts (or integer cost function) which is not the primary intent in remeshing. This becomes even more critical for anisotropic mesh adaptation where refinements have a large variation in the computational domain. Additional developments of graph-based methods are then necessary to work in the anisotropic framework [16]. Domain partitioning represents also one of the main parallel overhead of the method. In particular, general purpose graph-partitioners cannot take into account the different geometrical properties of the sub-domain to be partitioned. Indeed, splitting an initial domain is completely different from partitioning an interface mesh.

Partition remeshing. This is the core component of the coarse-grained parallelization. The overall efficiency of the approach is bounded by the limits of the sequential mesh generator. One limit is the speed of the sequential remesher that defines the optimal potential speed in parallel. In addition, as for as the partitioning of interfaces, meshing a partition is different from meshing a standard complete domain. Indeed, the boundary of the partition usually features non-manifold components and constrained boundary faces. In particular, it is necessary to ensure that the speed and robustness of the remesher is guaranteed on interface meshes.

Out-of-core. Out-of-core meshing was originally designed to store the parts of the mesh that were completed on disk to reduce the memory footprint [4]. Despite the high increase of memory (in term of storage and speeds with solid state drives), coupling out-of-core meshing with a parallel strategy may be advantageously used. On shared memory machines (with 100-200 cores), if the memory used by a thread is bigger that the memory of a socket, then the memory exchange between neighboring sockets implies a huge overhead of the sequential time (when running the procedure with one thread only). This phenomena is even more critical of NUMA architectures.

Our approach. Our procedure is based standard coarse-grained parallel strategies [16,19,20] where the initial domain is split into several sub-domains that are meshed in parallel. The interfaces between the partitions are constrained during the meshing phase. We define in this paper, two distinct partitioning techniques depending on the level of refinement. In particular, we take advantage of the geometry of the mesh at the interface to guarantee that the number of constrained faces are minimized at each steps. From the partitions' interfaces, a new volume mesh is deduced and split again until convergence. In comparison with standard approaches, the volume and the surface meshes are adapted at the same time. To handle non uniform refinements (in term of sizes and directions), a metric-based static load balancing formula is used to *a priori* equilibrate the work on each sub-domain. Once the remeshing of a sub-domain is completed, two additional sub-domains are created. The first one represents an interface mesh composed of elements that need additional refinement. The second one is the completed part that is stored to disk. To define the interface mesh, mesh modification operators (insertion/collapse) are simulated in order to enlarge the initial interface mesh to perform a quality remeshing in the subsequent iterations. Current state-of-art parallel mesh generation approaches [8,18] for unstructured (and adapted) meshes require thousands of cores (4092-200 000 cores)

to generate meshes with a billion elements. Our scope is to make this size of meshes affordable on cheaper parallel architectures (120-480 cores) with an acceptable runtime for a design process (less than 20 min).

The paper is organized as follows. In Section 2, we describe the domain partitioning methods and the load balancing. In Section 3, the properties of the local remeshing algorithm are described. Finally, we give numerical examples.

2. Domain partitioning

In the context of parallel remeshing, the domain partitioning method must be fast, low memory, able to handle domain with many connected components and effective to balance the remeshing work. Moreover, we should have efficient partitioning method for several level of partitions. More precisely, we first - level 1 - split the domain volume. Level 2, we split the interface of the partitions of level 1; the interface domain being formed by all the elements having at least one vertex sharing several sub-domains. Level 3, we split the interface of the partitions of level 2, and so on. The different levels for the decomposition of a cubic domain into 32 partitions is shown in Figure 1. We observe that the domain topology varies drastically with the level.

2.1. Element work evaluation

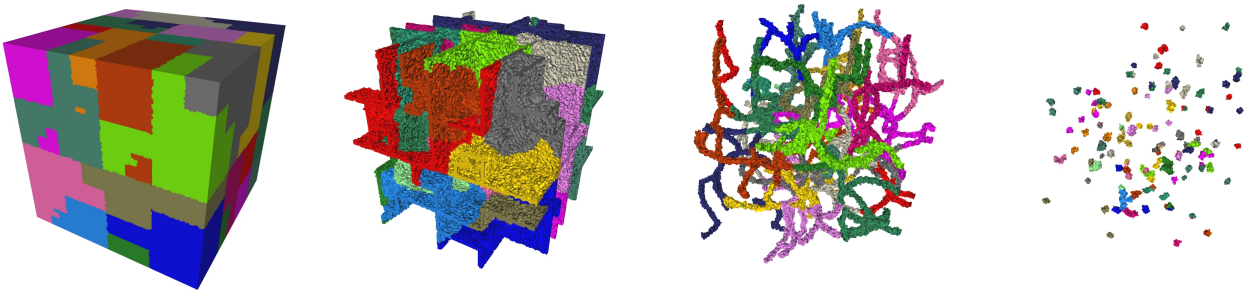
An effective domain partitioning strategy should balance the work which is going to be done by the local remesher on each partition, knowing that each partition is meshed independently, *i.e.*, there is no communication and the partition interfaces are constrained. The work to be performed depends on the used mesh operations (insertion, collapse, swap, smoothing), the given metric field $\mathcal{M}$ and, also, on the initial mesh $\mathcal{H}$ natural metric field $\mathcal{M}_{\mathcal{H}}$. Indeed, if the initial mesh already satisfies the metric then nothing has to be done. We recall that the natural metric of an element K is the unique metric tensor $\mathcal{M}_K$ such that all edges of K are of length 1 for $\mathcal{M}_K$ which is obtained by solving a simple linear system [22]. And, metric field $\mathcal{M}_{\mathcal{H}}$ is the union of the element metrics $\mathcal{M}_K$.

Isotropic case. Assuming the initial mesh is too coarse and is going to be only refined, the work per element is:

$$wrk_{vol}(K) = r_n \left(\frac{d_{\mathcal{M}_K}}{d_{\mathcal{M}}} - 1 \right),$$

where r_n is a constant defining the cost of the vertex insertion operator in dimension n and $d_{\mathcal{M}} = |K| \sqrt{\det \mathcal{M}}$ is the metric density. For an isotropic metric, metric $\mathcal{M}$ reduces to $h_{\mathcal{M}}^{-2} \mathcal{I}_n$ with $h_{\mathcal{M}}$ the local mesh size, and $d_{\mathcal{M}} = |K| h_{\mathcal{M}}^{-n}$. Thus, we get

$$wrk_{vol}(K) = r_n \left(\frac{h_{\mathcal{M}}^{-n}}{h_{\mathcal{M}_K}^{-n}} - 1 \right).$$



Level 1 Level 2 Level 3 Level 4

Fig. 1. Recursive partitioning into 32 sub-domains of a cubic domain for a constant work per element. From left to right, level 1, 2, 3 and 4 of partitioning. We observe that the domain topology varies drastically with the level.

For instance, if element K has a constant size h and we seek for a final mesh of size $h/2$ then the work is

$$wrk_{vol}(K) = r_n\,(2^n - 1)\,.$$

But, this formula is not valid for coarsening. The opposite is required. Hence,

$$wrk_{vol}(K) = c_n\,\left(\frac{h_{\mathcal{M}_K}^{-n}}{h_{\mathcal{M}}^{-n}} - 1\right).$$

where c_n is a constant defining the cost of the vertex collapse operator in dimension n. In our case, the local remeshing strategy uses a unique cavity operator for all mesh modifications (see Section 3), therefore all mesh modifications have exactly the same cost. We thus set: $r_n = c_n = 1$. Finally, the work per element is evaluated as

$$wrk_{vol}(K) = \max\left(\frac{h_{\mathcal{M}_K}}{h_{\mathcal{M}}}\,,\ \frac{h_{\mathcal{M}}}{h_{\mathcal{M}_K}}\right)^n - 1\,.$$

Anisotropic case. We cannot directly use the density (or sizes) of the anisotropic metric to define the work per element because the direction associated with each size must be taken into account. Indeed, two metrics may have the same density but opposite directions hence in one direction we should refine the mesh and in the other direction we should coarsen the mesh. To take into account these variations, we consider the intersected metric $\mathcal{M}_I = \mathcal{M} \cap \mathcal{M}_K$ based on the simultaneous reduction of both metrics ($\mathcal{M}$ and $\mathcal{M}_K$) [12]. $\mathcal{M}_I$ represents the common part of both metrics. The volume included between the unit-ball of $\mathcal{M}_I$ and $\mathcal{M}_K$ must be removed. Then, an estimate of the numer of collapse is then given by $(d_{\mathcal{M}_I} - d_{\mathcal{M}_K})$. Similarly, the volume difference between the unit ball of $\mathcal{M}_I$ and $\mathcal{M}$ represents the number of point to be added. In term of density, an estimate of the number of insertions is provided by $(d_{\mathcal{M}_I} - d_{\mathcal{M}})$. If we assume that the serial remesher has a linear complexity with respect to the work (number of insertion and collapse), the estimate of the work per element is sought as a linear function of :

$$wrk_{vol}(K) = \alpha\, d_{\mathcal{M}} + \beta\, d_{\mathcal{M}_I} + \gamma\, d_{\mathcal{M}_K},$$

where constants α, γ and β depends on the properties of the remesher. In our case, a linear regression on a set of test cases leads to :

$$wrk_{vol}(K) = 3(d_{\mathcal{M}} + d_{\mathcal{M}_K}) - 2\, d_{\mathcal{M}_I}. \tag{1}$$

We find in this formula the linear costs in term of the number of collapse and insertion but also an additional cost proportional to $d_{\mathcal{M}}$ that represents the final optimisation (smoothing, swap of edge-face) on the complete mesh, and $d_{\mathcal{M}_K}$ that represents an optimizations (swap of edge-face) on the set of removed elements. Similar formula is used to estimate the work per triangle by taking into account the surface metric. In Table 1, we consider the adaptation (at one time step) for a blast problem from an initial uniform mesh. The initial mesh is composed of 821 373 vertices and 4 767 431 while the adapted mesh is composed of 82 418 vertices and 511 998 tetrahedra, see Figure 3. The structure of the shocks waves implies that a large number of insertion and collapse are needed while being non uniformly distributed in the domain. Using (1) to balance the work leads to a quasi-uniform CPU times for each of the 8 partitions. On the contrary, considering only the final metric density to balance the partitions leads a completely non uniform CPU times and a large overhead in waiting the end of all the remeshing of the partitions.

2.2. Partitioning methods

Before using any of the partitioning methods presented below, the mesh vertices are first renumbered using a Hilbert space filling curve based reordering [3]. A Hilbert index (the position on the curve) is associated with each vertex according to its position in space. This operation has a linear complexity and is straightforward to parallelize as there is no dependency. Then, the renumbering is deduced from the vertices Hilbert indices. Vertices are sorted using the standard C-library `quicksort`.

The domain partitioning problem can be viewed as a renumbering problem of the elements. In that case, the first partition is composed of the elements from 1 to N_1 such that the sum of these elements work is equal to the total mesh

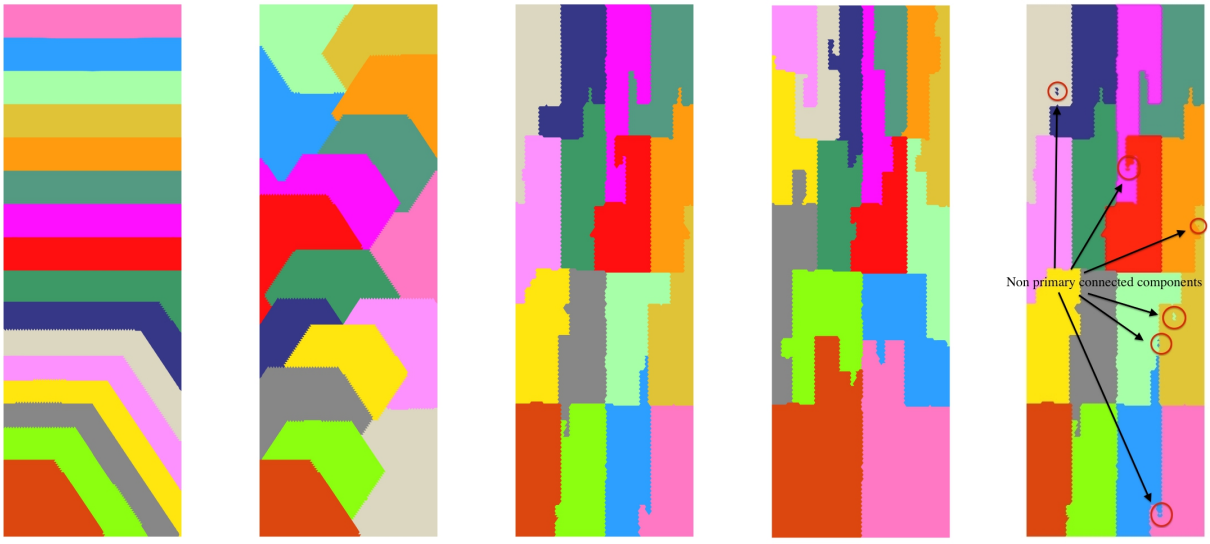
anisotropic work	statistics for each partition								waiting time
Cpu time (sec)	6.1	6.2	6.2	6.9	7.2	7.6	7.3	7.3	1.5 s
number of collapse	98 704	53 525	64 008	95 276	95 403	95 607	92 716	93 778	
number of insertion	0	38 039	30 402	0	0	0	0		
density-based work	statistics for each partition								waiting time
Cpu time (sec)	13.7	1.0	1.1	1.4	15.8	1.2	1.24	17.6	16.6 s
number of collapse	223 774	98	6	3 122	232 065	531	2 379	260 836	
number of insertion	8 494	8 385	8 246	8 340	8 117	8 601	8 508	8 402	

Table 1. Statistics of 8 partitions for a blast problem comparing the final remeshing CPU time when the partitions are balanced according to the anisotropic work estimate and to the metric density alone.

work divided by the total number of partitions. Then, the second partition is composed of the elements from $N_1 + 1$ to N_2 such that that the sum of these elements work is equal to the total mesh work divided by the total number of partitions. And so on. The difference between all strategies lies on the choice of the renumbering. Note that, for efficiency purposes, the elements are not explicitly reordered but they are only assigned an index or a partition index on the fly.

Now, assuming the vertices have been renumbered, we propose three methods to split the mesh: Hilbert based, breadth-first search (BFS) or frontal approach, and BFS with restart.

Hilbert partitioning. It consists in ordering the elements list according to the element minimal vertex index. In other words, we first list the elements sharing vertex 1 (the elements ball of vertex 1), then we list the elements sharing vertex 2 (the elements ball of vertex 2 not already assigned), etc. This splitting of the domain is based on the Hilbert renumbering of the vertices. For level 1 domain (initial domain splitting), it results in block partitions with uniform size interface (see Figure 2 (c)) but it may leads to partitions with several connected components on complex geometry due to domain holes not seen by the Hilbert curve. For level 2 or more domains, it is not effective because



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

Fig. 2. Partitioning into 16 sub-domains of a - level 1 - rectangular domain for a constant work per element with the BFS (a), BFS with restart (b) and Hilbert-based (c) methods. Picture (d) shows the Hilbert-based partitioning with a linear work function (the work per element increase with y) which has to be compare with picture (c) for a constant work per element. Picture (e) shows the Hilbert-based partitioning before the connected components correction. Several isolated connected components appear. The result after the correction is shown in picture (c).

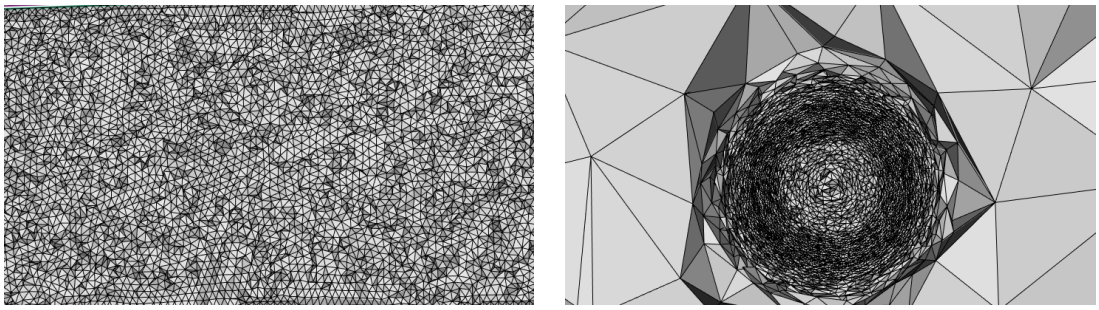


Fig. 3. Blast example to asses partitioning techniques and anisotropic work prediction: initial uniform mesh (left) and final adapted mesh (right). The serial adaptation takes 36 s.

Method	# of interface faces								Total
Hilbert	30 513	26 326	33 845	39 877	33 922	49 828	2 9011	43 720	143 518
BFS	19 584	46 126	55 682	53 256	46 120	43 557	41 664	20 111	163 050
BFS restart	19 569	25 962	29 162	40 649	28 417	24 945	41 045	23 745	116 744

Table 2. Statistics of 8 partitions for the blast problem. For each techniques, we report the number of interface faces. The relative misbalance of work per partition is around 10^{-2}. We observe a 16% maximal variation for Hilbert-based, 10% BFS and 18% for BFS restart and more than 30% variation on the number of total interface faces between each technique.

it will reproduce the previous level result and thus it will not gather the interfaces of different sub-domains. Hilbert partitioning provides a mean results in term of number of interface faces, see Table 2.

Breadth-first search (BFS) partitioning. Here, we start from an element root - generally, element 1 - and we add the neighbor elements of the root first. Then, we move to the next level of neighbors, in other words, we add the neighbor of the neighbors not already assigned. And so on. This splitting of the domain progresses by front. Indeed, each time an element is assigned, its non-assigned neighbors are added to a stack. The elements in this stack represent the current front. For level 1 domain, it results in layered partitions which contains only one connected component (see Figure 2 (a)) except the last one(s) which could be multi-connected. For level 2 or more domains, this method is able to gather the interfaces of different sub-domains but, as the stack is always growing, the number of connected components grows each time a bifurcation is encountered (see Figure 4 (a)). This leads to unbalance sub-domains after the correction presented in Section 2.3. BFS techniques provides the worst result as it tends to maximize the number of interface faces 2.

Breadth-first search (BFS) with restart partitioning. In the previous BFS algorithm, the splitting progresses by front, and generally this front grows until it reaches the diameter of the domain. During the splitting of interface domains (level 2 or more), this is problematic because the resulting partitions are multi-connected, cf. Figure 4 (a). One easy way to solve this issue is to reset the stack each time we deal with a new partition. The root of the new partition is the first element of the present stack, all the other elements are removed from the stack. For level 1 domain, it results in more circular (spherical) partitions (see Figure 2 (b)). For level 2 or more domains, this method is able to gather the interfaces of different sub-domains and also to obtain one connected component for each partition expect the last one(s), see Figure 4 (c). We observe in Figure 1 that the size of the partitions interface mesh reduces at each level. The BFS with restart provides the optimal results in term of number of interfaces, see Table 2, by reducing to 30% the number of interface faces from the two previous approaches. This techniques is then used in the sequel for every level of the remeshing phase.

2.3. Connected components correction

As the interface are constrained and not remeshed, the number of connected components per sub-domain should be minimized to maximized the work done by the remeshing strategy. In other words, each partition should have only

one connected component if it is possible. All elements of the same connected component are linked by at least a neighboring face.

After the domain splitting, a correction is applied to merge isolated connected components, see Figure 2 (e). First, for each sub-domain, the number of connected components is computed and the primary connected component (the one with the most work) of each partition is flagged. Second, we compute the neighboring connected components of each non-primary connected component. Then, iteratively, we merge each non-primary connected component with a neighboring primary connected component. If several choices occur, we pick the primary connected component with the smallest work. The impact of this correction is illustrated in Figure 2 from (e) to (c).

Remark: *We may end-up with non-manifold (but connected) partitions,* i.e.*, elements are linked by a vertex or an edge. As the local remeshing strategy is able to take care of such configurations, no correction is applied. Otherwise, such configurations should be detected and corrected.*

2.4. Efficiency of the method

The presented domain partitioning methods minimize the memory requirement as the data structures they use are only : the elements list, the elements' neighbors list, the elements' partitions indices list and a stack.

They are efficient in CPU because the elements assignment to a sub-domain is done in one loop over the elements. Then, the connected components correction requires only a few loops over the partitions. For instance, let us consider the domain partitioning of a cubic domain composed of 10 million tetrahedra into 64 sub-domains. In serial on a Intel Core i7 at 2.7Ghz, it takes 0.52, 0.24 and 0.24 seconds for the partitioning of the level 1, 2 and 3 domains, respectively, where the Hilbert-based partitioning has been use for level 1 domain and the BFS with restart partitioning has been used for the level 2 and 3 domains.

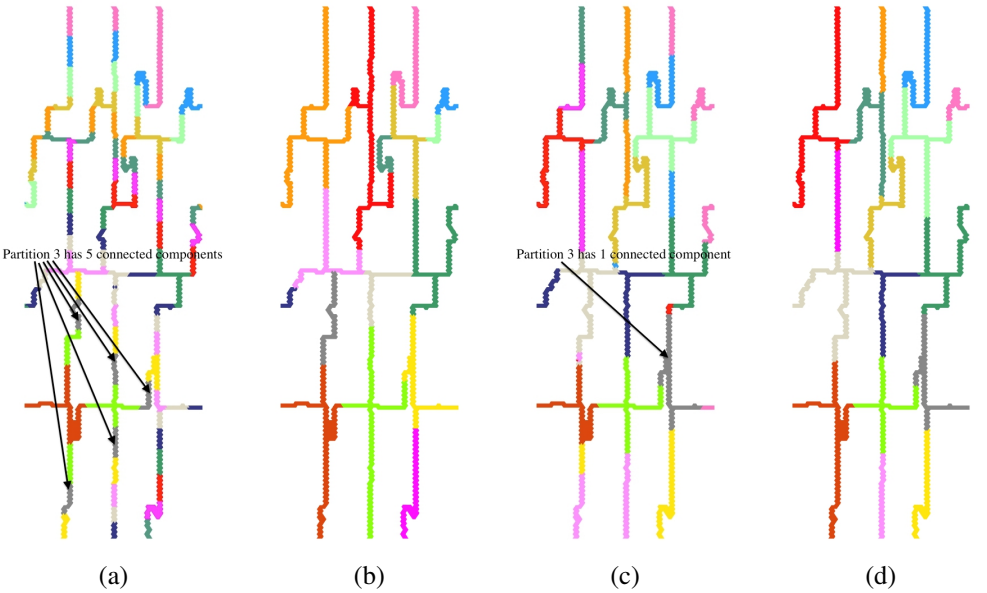


Fig. 4. Partitioning into 16 sub-domains of a - level 2 - interface mesh of a rectangular domain for a constant work per element. The interface mesh results from the Hilbert-based partitioning of the level 1 domain. Partitions obtained with the BFS method before and after correction are shown in pictures (a) and (b), respectively. Many connected components are created for each partition (a) due to the bifurcations resulting in an unbalance domain decomposition after correction (b). Partitions obtained with the BFS method before and after correction are shown in pictures (c) and (d), respectively. Just a few isolated small connected components are created leading to a balance domain decomposition after correction.

3. Sequential mesh generator

We give a brief overview of the AMG meshing algorithm that is used as the sequential mesh modification operator. For a complete description, we refer to [21,25]. it natively allows us to take into account constrained boundary faces (defining the interface) and can handle non manifold geometries. In addition, the volume and the surface meshes are adapted simultaneously in order to keep a valid 3D mesh throughout the entire process. This guarantees the robustness of the complete remeshing step. Note there is no specific modification for the parallel version from the sequential one except that the complexity of all loops is accurately verified. We list some of theses verifications below. Finally, the same operators (insertion/collapses) are simulated for points on the interface in order to define automatically the next level interface.

3.1. Metric-based and unit-mesh concept

AMG is a generic purpose adaptive mesh generator dealing with 2D, 3D and surface mesh generation. AMG belongs to the class of metric-based mesh generator [7,9,17,24,26] which aims at generating a unit mesh with respect to a prescribed metric field $\mathcal{M}$. A mesh is said to be unit when composed of almost unit-length edges and unit-volume element. The length of an edge AB in $\mathcal{M}$ is evaluated with:

$$\ell_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^tAB\,\mathcal{M}((1-t)A+tB)\,AB}\,\mathrm{d}t,$$

while the volume is given by $|K|_{\mathcal{M}} = \sqrt{\det \mathcal{M}}|K|$, where $|K|$ is the Eucldidean volume of K. From a practical point of view, the volume and length requirements are combined into a quality function defined by :

$$Q_{\mathcal{M}}(K) = \frac{36}{3^{\frac{1}{3}}} \frac{\sum_{i=1}^{6} \ell^2_{\mathcal{M}}(\mathbf{e}_i)}{|K|^{\frac{2}{3}}_{\mathcal{M}}} \in [1,\infty],$$

where $\{\mathbf{e}_i\}_{i=1,6}$ are the edges of element K. A perfect element has a quality of 1.

3.2. Cavity-based operators

A complete mesh generation or mesh adaptation process usually requires a large number of operators: Delaunay insertion, edge-face-element point insertion, edge collapse, point smoothing, face/edge swaps, etc. Independently of the complexity of the geometry, the more operators are involved in a remeshing process, the less robust the process may become. Consequently, the multiplication of operators implies additional difficulties in maintaining, improving and parallelizing a code. In [25], a unique cavity-based operator has been introduced which embeds all the aforementioned operators. This unique operator is used at each step of the process for surface and volume remeshing.

The cavity-based operator is inspired from incremental Delaunay methods [5,13,30] where the current mesh $\mathcal{H}_k$ is modified iteratively through sequences of point insertion. The insertion of a point P can be written:

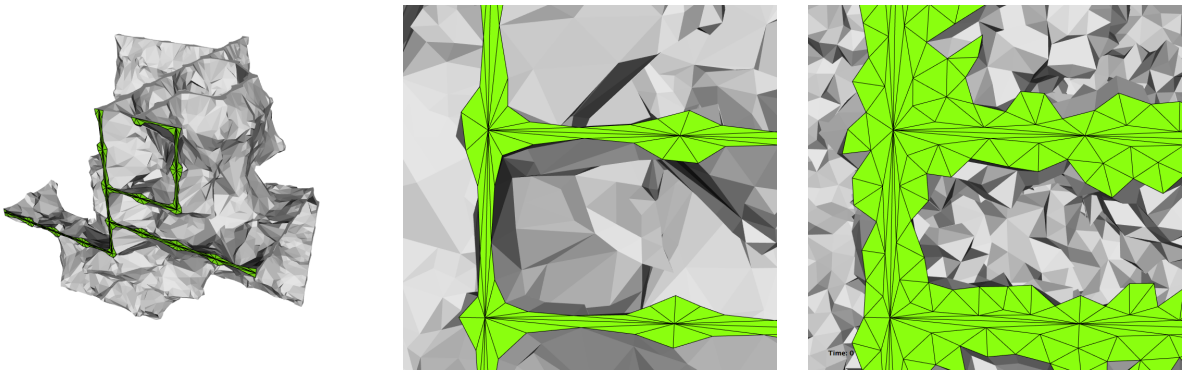
$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C}_P + \mathcal{B}_P, \tag{2}$$

where, for the Delaunay insertion, the cavity $\mathcal{C}_P$ is the set of elements of $\mathcal{H}_k$ such that P is contained in their circumsphere and $\mathcal{B}_P$ is the ball of P, i.e., the set of new elements having P as vertex. These elements are created by connecting P to the set of the boundary faces of $\mathcal{C}_P$.

In [25], each meshing operator is equivalent to a node (re)insertion inside a cavity. For each operator, we just have to define judiciously which node P to (re)insert and which set of volume and surface elements will form the cavity $\mathcal{C}$ where point P will be reconnected:

$$\mathcal{H}_{k+1} = \mathcal{H}_k - \mathcal{C} + \mathcal{R}_P. \tag{3}$$

Note that if $\mathcal{H}_k$ is a valid mesh (only composed of elements of positive volume) then $\mathcal{H}_{k+1}$ will be valid if and only if $\mathcal{C}$ is connected (through internal faces of tetrahedron) and $\mathcal{R}_P$ generates only valid elements.



Interface No correction Cavity-based correction

Fig. 5. Definition of the interface mesh on a cube example. Global view of interface geometry (left), interface defined by the balls of the vertices belonging to the interface (middle), and interface mesh defined by predicting the set of elements needed to perform the remeshing operation (insertion or collapse).

3.3. Features of the serial remesher

The use of the previous cavity-based operators allows us to design a remeshing algorithm that has a linear complexity in time with respect to the required work (sum of the number of collapses and insertions). On a typical laptop computer Intel Core I7 at 2.7 GHz, the speed for the (cavity-based) collapse is around 20 000 points removed per second and the speed for the insertion is also around 20 000 points or equivalently 120 000 elements inserted per second. Both estimates hold in an anisotropic context [21]. In addition, the complexity to compress the mesh to remove destroyed entities (points, elements) depends on the number of destroyed entities (and not on the number of current entities). A rule of thumb is then to make sure that each loop used in the remeshing phase has a complexity proportional to the required work rather than a complexity proportional to the size of the current mesh. In many cases, it is sufficient to replace complete loop over the entities with a loop on a front of entities; This front being updated dynamically during the underlying process. If these modifications have a little impact on medium size meshes, they appear to be a drastic bottleneck for very large meshes or when the process is run in parallel with a *a piori* metric-based static load balacing.

3.4. Definition of the interface mesh

During the remeshing phase, the set of elements that surrounds the constrained faces (defining the boundary of the current partition) are not adapted. It is then necessary to define a set of elements that needs to be adapted at the next iteration (or level). An initial choice consists in introducing all the elements having at least one node on the boundary of the interface. This choice is illustrated in Figure 5 (middle). Despite its simplicity, this choice is appropriate only when the size of the elements of the interface is of the same order as the size imposed elsewhere. However, when large size variation occurs, additional elements need to be part of the new interface volume. Optimally, a sufficient number of elements needs to be added to make sure that underlying local modification will be possible at the next level. An automatic way to find this elements is to add the relevant set of elements of the cavity [21] for each operator. Two situations occur. When an edge of the interface is too short, a collapse will be needed at the next level. Consequently, for all interfaces sharing this edge, the ball of the two end-points edges is added. When an edge is too long, a point will be inserted at the next level, consequently, the Delaunay cavity of the mid-point edge is added. Note that these modifications are done parallel at the end of the remeshing step, thus limiting the overhead of this corection. The modification of the set of elements defining the interface is illustrated in Figure 5 where a cube domain is refined from a size h to $h/4$. If we select only the balls of the interface vertices, then the remeshing process is much more constrained, see Figure 5 (middle). Including additional elements based on the cavity defining the relevant mesh modification operator (collapse or insertion) gives additional room to the mesh generator to perform a quality modification 5 (right).

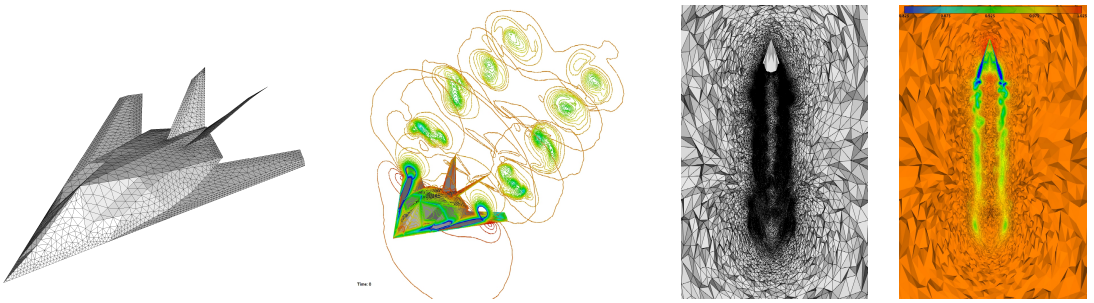


Fig. 6. F117 test case. From left to right, geometry of the f117 aircraft, representation of the vortical flow, top view of the mesh adapted to the local Mach number and local Mach number iso-values.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores	elt/sec	elt/sec/core
1	84%	69 195 431	433 495 495	180.8	120	$2.4\,10^6$	19 980
2	96%	1 692 739	502 706 732	95.0	120	$7.2\,10^5$	6 071
3	99%	1 231 868	518 850 149	35.9	91	$4.6\,10^5$	5 068
4	99%	6459	520 067 586	7.5	7	$1.6\,10^5$	2 318
5	100%	0	520 073 940	1.7	1	$3.7\,10^3$	3 737

Table 3. F117 test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores	elt/sec	elt/sec/core
1	76%	109 269 782	389 476 861	109.9	480	$3.5\,10^6$	7 383
2	91%	42 836 303	486 695 293	67.0	480	$1.4\,10^6$	1 440
3	98%	5 567 744	525 073 846	28.1	228	$1.3\,10^6$	6 011
4	99%	32292	530 573 260	8.9	30	$6.1\,10^5$	20 597
5	100%	0	530 605 308	2.3	1	$1.4\,10^4$	13 933

Table 4. F117 test case on 480 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

4. Numerical Results

Several examples are illustrated in this section. For each case, the parallel mesh generation converges in 5 iterations. The number of core is chosen to ensure that at least 100 000 tetrahedra per core will be inserted. Consequently, the number of cores is reduced when the remaining work decreases. All the examples are run on a cluster composed of 40 nodes with 48Gb of memory, composed of two-chip Intel Xeon X56650 with 12 cores. A high-speed internal network InfiniBand (40Gb/s) connects these nodes. For each example, we report the complete CPU time including the IOs, the initial partitioning and gathering along with the parallel remeshing time.

Vortical flows on the F117 geometry. This case is part of an unsteady adaptive simulation to accurately capture vortices generated by the delta-shaped wings of the F117 geometry, see Figure 6. The final adapted mesh of the simulation is depicted in Figure 6. The final adapted mesh is composed of 83 752 358 vertices, 539 658 triangles and 520 073 940 tetrahedra. The initial background mesh is 1 619 947 vertices, 45740 triangles and 9 710 771 tetrahedra. The complete CPU time (including initial domain partitioning and final gathering) is 12 min on 120 cores. The parallel mesh adaptation of the process takes 8 min 50 s. The parallel procedure inserts 10^6 vertices/min or equivalently $6\,.\,10^6$ tetrahedra/min, see Table 3. The maximal memory used per core is 1.25 Gb. The same example on 480 cores is reported in Table 4, the CPU for the parallel mesh generation part is 3 min 36 s while the maximal memory used per core is 0.6Gb. The speed up from 120 to 480 cores is limited to 1.5 (4 optimally), this is due to the large increase of the interfaces in the mesh, see Table 5 (left). For a partition, the typical time to create its interface mesh using the anisotropic Delaunay cavity is less than 10% of the meshing time.

Blast simulation on the tower bridge. The example consists in computing a blast propagation on the London Tower Bridge. The geometry is the 23rd IMR meshing contest geometry. The initial mesh is composed of 3 837 269 ver-

Iteration	120 cores	480 cores
1	590 038	954 166
2	1 711 512	4 306 256
3	130 262	589 532
4	869	4 018
5	0	0

Iteration	120 cores	480 cores
1	1 081 246	1 627 846
2	2 416 840	5 265 939
3	132 659	451 355
4	488	3 230
5	0	0

Table 5. Number of boundary faces at the interfaces at each iteration when running on 120 and 480 cores for the F117 (left) and the tower-bridge (right) test cases.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores	elt/sec	elt/sec/core
1	84%	89 577 773	919 345 377	577.3	120	$1.5\,10^6$	13 277
2	95%	14 290 245	1 062 994 802	280.7	120	$5.1\,10^5$	4 264
3	97%	1 290 855	1 089 035 610	56.3	120	$4.6\,10^5$	3 854
4	97%	3636	1 090 321 352	8.0	7	$1.6\,10^5$	22 959
5	100 %	0	1 090 324 952	2.1	1	$1.7\,10^3$	1 714

Table 6. Tower-bridge test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores	elt/sec	elt/sec/core
1	79%	193 529 057	922 145 088	255.8	480	$3.6\,10^6$	7 510
2	93 %	52 837 674	1 115 428 211	106.7	379	$1.8\,10^6$	4 779
3	96%	4 258 411	1 165 096 167	34.6	282	$1.4\,10^6$	5 090
4	97%	27 095	1 169 283 585	23.0	23	$1.8\,10^5$	7 915
5	100%	0	1 169 310 260	3.9	1	$6.8\,10^3$	6 839

Table 7. Tower-bridge test case on 480 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

Fig. 7. Tower-bridge test case. Initial mesh and geometry (left) and density iso-values of the the blast on an adapted mesh (right).

tices 477 852 triangles and 22 782 603 tetrahedra while the final mesh is composed of 174 628 779 vertices 4 860 384 triangles and 1 090 324 952 tetrahedra. From the previous example, the surface geometry and mesh adaptation is much more complex as many shock waves impact the bridge. The time to generate the adapted mesh on 120 cores is 22 min 30 s and 28 min for the total CPU time including the initial splitting, final gathering and IOs. On 480 cores, the time to generate the mesh reduces to 16 min 30 s. The maximal memory used on 120 cores is 1.8Gb and reduces to 1Gb on 480 cores. We report in Tables 5 (right), 6 and 7, the convergence of the process. This example exemplifies the robustness of this approach with complex geometries.

Landing gear geometry mesh refinement. This geometry is designed for the study of the propagation of the noise generated by a landing gear. This simulation requires large isotropic surface and volume meshes to capture the complex flow field which is used for aeroacoustic analysis. The initial background mesh is composed of 2 658 753 vertices 844 768 and 14 731 068 tetrahedra while the adapted mesh is composed of 184 608 096 vertices 14 431 356 triangles

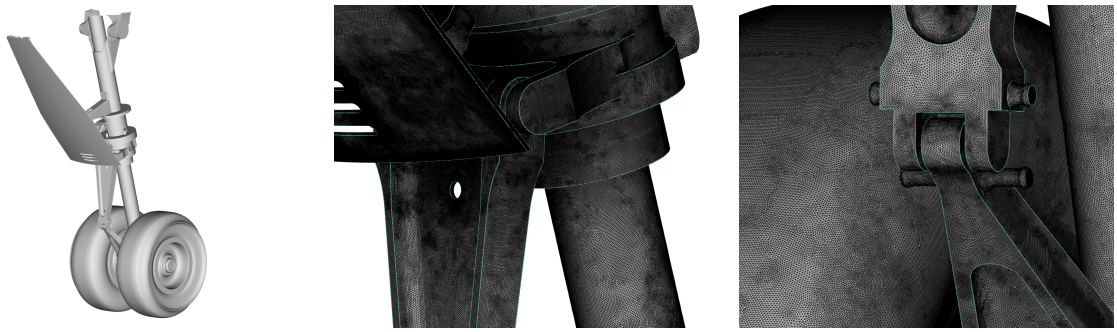


Fig. 8. Landing gear test case. Geometry of the landing gear (left) and closer view of the surface mesh around some geometrical details (middle and right).

Iteration	% done	# of tets in interface	# of tets inserted	CPU time (sec.)	# of cores	elt/sec	elt/sec/core
1	84 %	89 718 245	1 009 783 723	487.5	120	$2.0\,10^6$	17 261
2	91 %	16 368 313	1 107 015 758	126.7	120	$7.6\,10^5$	6 395
3	92 %	645 035	1 122 857 778	36.6	87	$4.3\,10^5$	4 975
4	97%	2 351	1 123 488 597	5.6	4	$1.1\,10^5$	28 161
5	100%	0	1 123 490 929	1.7	1	$1.3\,10^3$	1 371

Table 8. Landing gear test case on 120 cores. Table gathering the size of the interface, the number of inserted tetrahedra and the CPU time for each iteration.

and 1 123 490 929 tetrahedra. The parallel remeshing time is 15 min 18 s and the total CPU time is 24 min 57 s (with the initial splitting and the final gathering). This example illustrates the stability of this strategy when the surface mesh contains most of the refinement. Indeed, the surface mesh is composed of more than 7.2 million vertices and 14.4 million triangles. Table 8 gathers all the data per iteration on this case. The geometry and closer view on the surface mesh are depicted in Figure 8.

5. Conclusion and future works

An efficient coarse-grained parallel strategy is proposed to generate large-size adaptive meshes. Both uniform, isotropic and anisotropic refinements are handled. The volume and the surface meshes are adapted simultaneously and a valid mesh is kept throughout the process. The parallel resources are used to remove the memory impediment of the serial meshing software. Even if the remeshing is the only part of the process completely done in parallel, we still achieve reasonable CPU times. The CPU time for the meshing part ranges from 15 min to 30 min to generate 1 billion tetrahedra adapted meshes. The key components of the process are:

- a fast sequential cavity-based remesher that can handle constrained surface and non-manifold geometries during the remeshing,
- specific splitting of the interface mesh ensuring that the number of faces defining the interfaces tends to zero,
- a cavity-based correction of the interface mesh to ensure that enough elements are included in order to favor the success of the needed mesh modification operator at the next iteration.

Additional developments are needed to still reduce the total CPU time. The current work is directed at recovering the IOs with the remeshing. Indeed, as we use an out-of-core strategy, the final gathering can be partially done at the same time. Then, the partitioning techniques of the interfaces is also currently extending to work efficiently as well in a parallel environment.

References

[1] The ubercloud hpc experiment: Compendium of case studies, 2013.

[2] F. Alauzet, X. Li, E. Seegyoung Seol, and M.S. Shephard. Parallel anisotropic 3D mesh adaptation by mesh modification. *Engineering with Computers*, 21(3):247–258, 2006.

[3] F. Alauzet and A. Loseille. On the use of space filling curves for parallel anisotropic mesh adaptation. In *Proceedings of the 18th International Meshing Roundtable*, pages 337–357. Springer, 2009.

[4] A. Alleaume, L. Francez, M. Loriot, and N. Maman. Automatic tetrahedral out-of-core meshing. In MichaelL. Brewer and David Marcum, editors, *Proceedings of the 16th International Meshing Roundtable*, pages 461–476. Springer Berlin Heidelberg, 2008.

[5] A. Bowyer. Computing dirichlet tessellations. *Comput. J.*, 24(2):162–166, 1981.

[6] A.N. Chernikov and N.P. Chrisochoides. A template for developing next generation parallel delaunay refinement methods. *Finite Elements in Analysis and Design*, 46(12):96 – 113, 2010. Mesh Generation - Applications and Adaptation.

[7] T. Coupez. Génération de maillages et adaptation de maillage par optimisation locale. *Revue Européenne des Éléments Finis*, 9:403–423, 2000.

[8] Hugues Digonnet, Luisa Silva, and Thierry Coupez. Massively parallel computation on anisotropic meshes. In *6th International Conference on Adaptive Modeling and Simulation*, pages 199–211, Lisbon, Portugal, June 2013. International Center for Numerical Methods in Engineering.

[9] C. Dobrzynski and P.J. Frey. Anisotropic Delaunay mesh adaptation for unsteady simulations. In *Proceedings of the 17th International Meshing Roundtable*, pages 177–194. Springer, 2008.

[10] J. Dongarra. Toward a new metric for ranking high performance computing systems. *Sandia Report*, 2013.

[11] P. Foteinos and N.P. Chrisochoides. Dynamic parallel 3d delaunay triangulation. In WilliamRoshan Quadros, editor, *Proceedings of the 20th International Meshing Roundtable*, pages 3–20. Springer Berlin Heidelberg, 2012.

[12] P.J. Frey and F. Alauzet. Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.*, 194(48-49):5068–5082, 2005.

[13] F. Hermeline. Triangulation automatique d'un polyèdre en dimension *n. RAIRO - Analyse numérique*, 16(3):211–242, 1982.

[14] Y. Ito, A.M. Shih, A.K. Erukala, B.K. Soni, A.N. Chernikov, N.P.Chrisochoides, and K. Nakahashi. Parallel unstructured mesh generation by an advancing front method. *Math. Comput. Simul.*, 75(5-6):200–209, September 2007.

[15] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[16] C. Lachat, C. Dobrzynski, and F. Pellegrini. Parallel mesh adaptation using parallel graph partitioning. In *5th European Conference on Computational Mechanics (ECCM V)*, volume 3 of *Minisymposia in the frame of ECCM V*, pages 2612–2623, Barcelone, Spain, July 2014. IACM & ECCOMAS, CIMNE - International Center for Numerical Methods in Engineering. ISBN 978-84-942844-7-2.

[17] X. Li, M.S. Shephard, and M.W. Beal. 3D anisotropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.*, 194(48-49):4915–4950, 2005.

[18] A. Lintermann, S. Schlimpert, J.H. Grimmen, C. Gnther, M. Meinke, and W. Schrder. Massively parallel grid generation on {HPC} systems. *Computer Methods in Applied Mechanics and Engineering*, 277(0):131 – 153, 2014.

[19] R. Löhner. A 2nd generation parallel advancing front grid generator. In Xiangmin Jiao and Jean-Christophe Weill, editors, *Proceedings of the 21st International Meshing Roundtable*, pages 457–474. Springer Berlin Heidelberg, 2013.

[20] R. Löhner, J. Camberos, and M. Merriam. Parallel unstructured grid generation. *Computer Methods in Applied Mechanics and Engineering*, 95(3):343 – 357, 1992.

[21] A. Loseille. Metric-orthogonal anisotropic mesh generation. *Proceedings of the 23th International Meshing Roundtable, Procedia Engineering*, 82:403–415, 2014.

[22] A. Loseille and F. Alauzet. Continuous mesh framework. Part I: well-posed continuous interpolation error. *SIAM J. Numer. Anal.*, 49(1):38–60, 2011.

[23] A. Loseille and R. Löhner. On 3D anisotropic local remeshing for surface, volume and boundary layers. In *Proceedings of the 18th International Meshing Roundtable*, pages 611–630. Springer, 2009.

[24] A. Loseille and R. Löhner. Adaptive anisotropic simulations in aerodynamics. In *48th AIAA Aerospace Sciences Meeting*, AIAA Paper 2010-169, Orlando, FL, USA, Jan 2010.

[25] A. Loseille and V. Menier. Serial and parallel mesh modification through a unique cavity-based primitive. In *Proceedings of the 22th International Meshing Roundtable*, pages 541–558. Springer, 2013.

[26] T. Michal and J. Krakos. Anisotropic mesh adaptation through edge primitive operations. *50th AIAA Aerospace Sciences Meeting*, Jan 2012.

[27] C. Özturan, H.L. deCougny, M.S. Shephard, and J.E. Flaherty. Parallel adaptive mesh refinement and redistribution on distributed memory computers. *Computer Methods in Applied Mechanics and Engineering*, 119(12):123 – 137, 1994.

[28] M.S. Shephard, C. Smith, and J.E. Kolb. Bringing hpc to engineering innovation. *Computing in Science Engineering*, 15(1):16–25, Jan 2013.

[29] U. Tremel, K.A. Sørensen, S. Hitzel, H. Rieger, O. Hassan, and N.P. Weatherill. Parallel remeshing of unstructured volume grids for cfd applications. *International Journal for Numerical Methods in Fluids*, 53(8):1361–1379, 2007.

[30] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to voronoi polytopes. *Comput. J.*, 24(2):167–172, 1981.