Parallel unstructured mesh adaptation using iterative remeshing and repartitioning

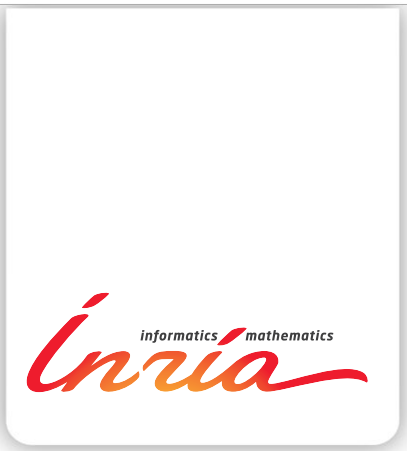
Luca Cirrottola, Algiane Froehly

Parallel unstructured mesh adaptation using iterative remeshing and repartitioning

Luca Cirrottola, Algiane Froehly

RESEARCH REPORT
N° 9307
November 2019
Project-Team Cardamom

ISSN 0249-6399 ISRN INRIA/RR--9307--FR+ENG



Parallel unstructured mesh adaptation using iterative remeshing and repartitioning

Luca Cirrottola*, Algiane Froehly†

Project-Team Cardamom

Research Report n° 9307 — November 2019 — 24 pages

Abstract: Mesh adaptation has proven to be a powerful tool for increasing the accuracy of numerical simulations whenever the solution exhibits strong non-uniform features over the computational domain. Sequential remeshing currently represents a bottleneck for parallel solvers. To this aim, we present a parallel remeshing algorithm that enables the reuse of existing sequential remeshing libraries, a non-intrusive linkage with third-party solvers, and an efficient exploitation of distributed parallel environments. The numerical procedure is implemented in the open source `ParMmg` software package.

Key-words: parallel mesh adaptation, unstructured meshes, mesh migration, interface displacement, distributed memory environments.

* Postdoctoral researcher, Inria Bordeaux - Sud-Ouest, Institut de Mathématiques de Bordeaux (IMB)
† Research engineer, Inria - Diréction générale déléguée à l'innovation (DGD-I)

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Adaptation parallèle de maillages non structurés avec remaillage et repartitionnement itératif

Résumé : L'adaptation de maillage a été provée comme un outil puissant pour améliorer la précision des simulations numériques dans tous les cas où la solution présente des characteristiques non uniformes sur le domain de calcul. Le remaillage séquentiel représente un cou de bouteille pour les solveurs parallèles. Pour cette raison, nous présentons un algorithme de remaillage parallèle qui permet la réutilisation des bibliothèques séquentielles de remaillage déjà existantes, un couplage non intrusif avec des solveurs externes, et une exploitation efficiente des environnements parallèles à mémoire distribuée. La procédure numérique est implémentée dans le logiciel open source `ParMmg`.

Mots-clés : adaptation de maillage parallèle, maillages non structurés, migration de maillage, déplacement d'interface, environnemments à mémoire distribuée.

Contents

1 Introduction **3**

2 Parallel iterative remeshing over constrained interfaces **4**

3 Metrics interpolation **5**

4 Mesh repartitioning **5**
4.1 Interface displacement by an advancing-front method 6
4.2 Groups sorting for interface advancement direction 6
4.3 Correcting disconnected partitions 8
4.3.1 Examples of configurations leading to disconnected partitions 8
4.3.2 A posteriori corrections 10

5 Summary of the algorithm **11**

6 Open-source software implementation **11**

7 Results **12**
7.1 Qualitative comparison of graph-based repartitiong and interface displacement algorithms 12
7.2 Parallel weak-scaling of a uniform refinement test case 13
7.3 Preliminary results on anisotropic test cases 13

8 Conclusion and future developments **21**

1 Introduction

Mesh adaptation is becoming an increasingly used tool for improving the accuracy of numerical solutions in computational solid mechanics (CSM) and computational fluid dynamics (CFD)[1]. The need to adapt along sharp solution patterns (such as wave fronts and shock waves in fluid flows, or crack growth in solid mechanics, for example) has risen the interest to include preferential directions for local mesh refinement, derefinement and optimization, commonly referred as anisotropic mesh adaptation.

Metric-based mesh adaptation (see for example [FG07] [Dob05] [DF08] [FA05]) offers a general framework for both isotropic and anisotropic mesh adaptation. The target mesh edge sizes and directions are computed from the numerical solution and its derivative, and used to build a Riemaniann metric tensor which describes a transformed space where the mesh should have unit sizes in each direction. In this framework, classical mesh modification operations can be applied by simply re-evaluating edge lengths through the metric tensor. The problem of finding an error estimator to drive mesh adaptation thus becomes independent from that of adapting the mesh, once the error estimator is translated into a metric tensor, simplifying the development of general-purpose remeshing libraries.

Since most of CSM/CFD solvers nowadays work on parallel computers, sequential mesh adaptation constitutes a bottleneck in the numerical solution workflow [Par+16], impacting both the computational time and the problem size, since this latter is limited by the memory available

[1]The reader is referred for example to [Par+16] for a thorough review of the field in its application to CFD.

on a single computing node. From a general viewpoint, parallel mesh adaptation presents further complications compared to the sequential case, as several computational steps need to be considered:

1. Mesh partitioning (can be provided by the CSM/CFD solver).
2. Parallel remeshing.
3. Mesh repartitioning - Load balancing.

Depending on the strategies, the last two stages can be repeated iteratively until a satisfying mesh quality is reached.

Parallel remeshing strategies can be classified in two main categories. The first class of techniques aims at parallelizing the mesh generation techniques. Once an initial mesh partitioning is provided, the main difficulty of these techniques resides in adapting on the partition interfaces, as a non-negligible amount of communication is needed to preserve mesh conformity on the interface (see for example [CS97] [DCS99] [OBG00] [CN03]). A final load balancing step can be required in order to evenly distribute mesh entities over the processors before returning the adapted mesh to an external solver.

The second class aims at re-using existing sequential remeshers in a parallel framework. Once an initial mesh partitioning is established, mesh adaptation is forbidden on partition interfaces, in order to avoid race conditions and preserve mesh conformity, and adaptation is performed only in the partitions interior. The resulting mesh is then re-partitioned so to move the former partition interfaces, and remeshing is repeated in order to adapt on formerly forbidden zones. By iterating on these remeshing-repartitioning stages, a globally adapted mesh is obtained (see for example [Fla+98] [CSF05] [Dig+17] [MDV11]).

The aim of the `ParMmg` software package is to build parallel mesh adaptation capabilities on top of the sequential open-source remesher `Mmg` [DDF14] [Mmg], while preserving the open-source spirit and the support for general-purpose applications through the library application-programming interface (API). Therefore, in the following we will focus on the development of a parallel iterative remeshing algorithm allowing the re-use of existing sequential remeshers, coupled with a repartitioning algorithm capable of effectively move parallel interfaces.

2 Parallel iterative remeshing over constrained interfaces

The core idea of this family of method is to employ sequential remeshing techniques in the domains interior, while parallel interfaces are constrained (i.e. adaptation is forbidden on the interface), then the interface is moved by a repartitioning method, mesh migration is performed and the process is iterated until an overall good mesh quality is reached. The same idea has been already exploited in the context of parallel mesh generation [Löh13] [Löh14].

Sequential remeshing over constrained interfaces have been already exploited in a parallel iterative framework in [Ham92] [She+95] [Ozt95] [Fla+98], where the interface is moved by an advancing front and mesh migration is performed by one layer of elements at a time. The interface is instead moved by several layers at a time in [CSF05] [DR07] before performing mesh migration. In [Dig+17], the mesh migration by advancing front presented in [CDD00] is coupled with load balancing by processor pairing. In [LDP14] [Lac+17] remeshing by means of `Mmg` is iteratively performed on parts of the mesh that need remeshing after they have been identified and partitioned through `Scotch`. In [AL09], Hilbert space-filling curves plus a correction algorithm are used for the purpose of producing connected mesh partitions on every process, while `Mmg` is employed as sequential remesher, then partitions are merged and repartitioned to perform

new remeshing steps. In [LM13], after remeshing is performed in the partition interiors, one or two elements layers next to the interfaces are extracted and either gather or repartitioned to be adapted. In [LMA15], a multiple-levels partitioning is employed to partition the domain, than the set of elements touching the interfaces. Once the interior of the partitions are adapted, the interface layers are enlarged so to include all mesh cavity that will be needed to locally adapt on the interfaces.

Following previous works on the parallel usage of `Mmg` [MDV11] [Ben+16], we focus on a two-levels partitioning scheme where the mesh is first partitioned on the parallel computer. Then, local data are further partitioned into mesh groups, which can be remeshed sequentially by the local process one after the other. The size of the mesh group is targeted to an optimal size for the `Mmg` remeshing process. Both interfaces between partitions and mesh groups are kept unchanged during the remeshing step.

After the remeshing step, the mesh is repartitioned in order to change the parallel interface and to allow a new remeshing step. This can be accomplished through several strategies. In [MDV11] [Ben+16], a load balancing step is applied to the partitioning of the parallel mesh group graph rather, than of the parallel mesh element graph, in order to ease the partitioning task for the `Metis` library. To do that, the size of the mesh groups is recalibrated for the repartitiong task. A high weight is placed on graph edges proportionally to old parallel interfaces, so to penalize the occurrence of these edges in new partition interiors. `ParMmg` supports the same algorithm.

A certain compromise between an optimal load balancing and the number or remeshing-repartitioning iterations needs to be sought. We have found that load balancing algorithms can be difficult to tune for an effective interface displacement that limits the number of total iterations performed, as their primary aim is that of minimizing a communication cost function, without explicitly targeting interface displacement. For this reason, we have investigated the usage of direct interface displacement methods for mesh repartitioning, at the expense of optimal load balancing.

3 Metrics interpolation

As the original metrics can be modified by the sequential remesher in order to impose additional geometrical requirements or gradation (this is done in our case by `Mmg`), the original user-defined metrics needs to be recovered after each remeshing iteration in order to prevent information loss as the remeshing iterations are continued.

In our algorithm, this is done by interpolating the metrics from the mesh at the previous iteration to the mesh to the current adapted mesh. This allows to avoid parallel communication, as both the background and current group meshes are local. A classical walk-search algorithm is employed for node localization, and interpolation on P1 finite elements is used (both are resumed, for example, in [AM10]).

Solution interpolation is not currently supported, but different strategies could be envisaged. In this case, it would be better to interpolate only from the initial to the final mesh in order to reduce the computational time, with parallel box-intersection to pair the differently distributed initial and final groups (as done in [Dig+17]).

4 Mesh repartitioning

Many algorithms can be used to perform mesh partitioning. One category of methods falls into general graph partitioning approaches, such as those commonly used through the `Metis` and

`Scotch` libraries [Kar] [Pel09]. A second category exploits geometrical or topological information which are specific about the mesh partitioning problem, like recursive spectral bisection (described for example in [Sim91], inertia axis methods (described in [FG07] [FL93]) and greedy algorithms [Far88] [FL93], among others.

Graph-based partitioning schemes generally aim at minimizing a given cost function, as a model of the communication cost, and old parallel interfaces can be penalized by applying a high weight on the corresponding graph edge, thus only indirectly trying to minimize the occurrence of a new parallel interface coincident with the old one. Similarly, mesh-based partitioning methods are still designed to produce well-balanced partitions and to reduce the communication cost. This is often done by exploiting more information about the mesh as a bipartite graph, through element and node adjacency. In the following sections, we will rather focus on the construction of a repartitioning algorithm explicitly targeting interface displacement, as this is the primary interest in an iterative remeshing scheme in order to converge to an adapted mesh all over the domain.

4.1 Interface displacement by an advancing-front method

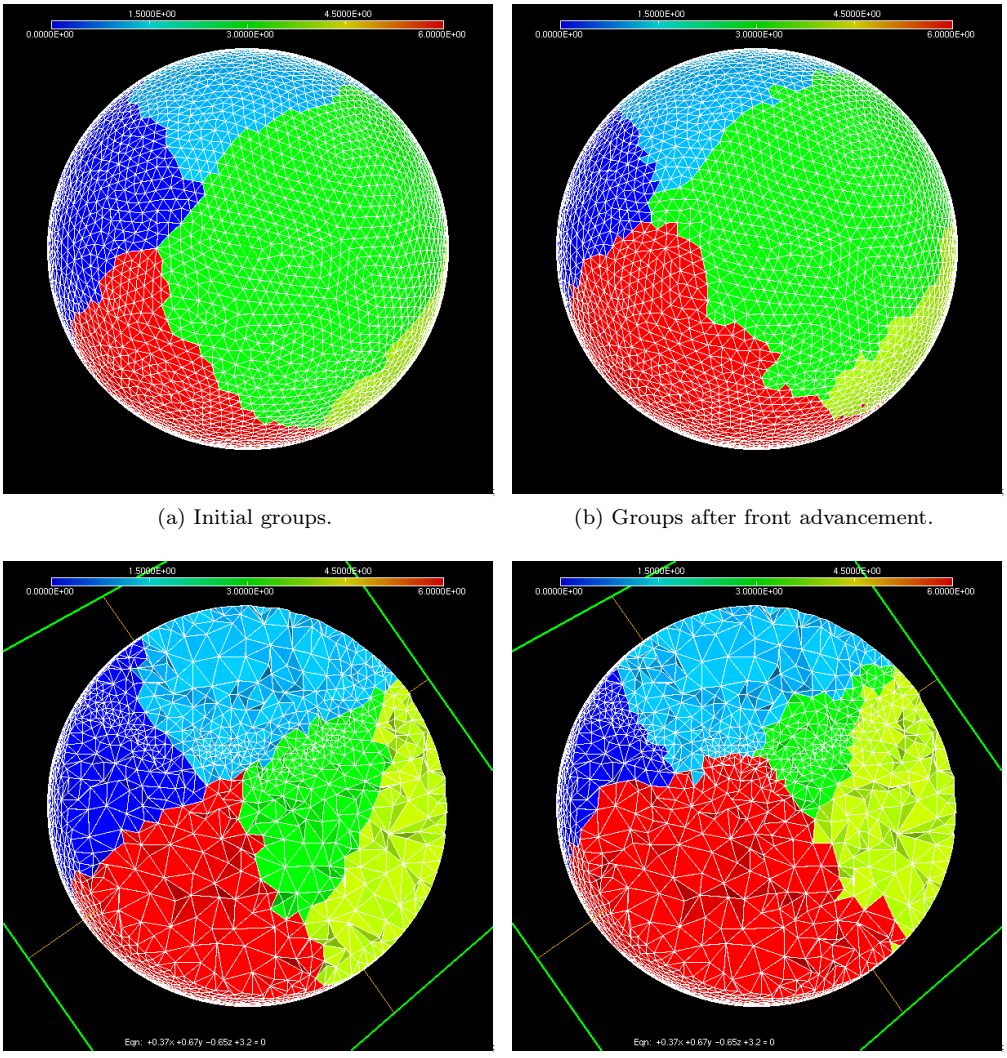
We follow the same idea employed in [DR07] [CSF05] to move the interface by an advancing-front algorithm. We adopt an element-based partitioning scheme, thus we aim at propagating the front of parallel nodes by walking on edge connectivity, in order to mark the visited tetrahedra with the color brought by the front.

To this purpose, a front direction needs to be chosen. This operation allows to give each front node a unique color, based on the colors of the tetrahedra in its ball, and a logical operator to decide whether this color should be passed to the tetrahedra visited by the front, or not. This operation also allows to handle the case of colliding fronts. Once a direction is chosen for the propagation of the interface, the front nodes visit all the tetrahedra in their ball and mark them for repartitioning, if it is compatible with the front direction. Then, the color is passed to the outer points of the ball, and a new front is created. By performing this process N times, an $N-$layer mesh subgroup is built just beside the old parallel interfaces. This process is reminiscent of a parallel version of the greedy algorithm for sequential mesh partitioning [Far88], where the seeds are not chosen independently but only the current node interfaces are allowed to grow their ball by N new layers. At the end of each step, a parallel update is performed for the color of the nodes on current parallel interfaces.

This algorithm effectively produces a new mesh partitioning that can be directly used for any already available mesh migration routines. An example of front advancement on 4 partitions in a sphere is given in figure 1. From the practical point of view, this algorithm needs to be complemented with: 1) the choice of the front advancement direction, and 2) handling of disconnected partitions, i.e. partitions where all elements cannot be reached by face adjacency (see for example figure 2), which are very likely to be produced by the advancing front [Far88] [Ozt95].

4.2 Groups sorting for interface advancement direction

For every interface, a propagation direction needs to be chosen. In [Ham92] and [Dig+17], processors are paired in order to balance the computational load. Since optimal load balancing is not the main aim of the remeshing phase, we have preferred to sort the current groups by the number of elements, and move the interface in the direction of the bigger mesh group. This naïve global sorting allows to effectively handling the case of colliding fronts, as each of them continues to propagate into the bigger partition found, without the need to form new pairings.



(a) Initial groups.

(b) Groups after front advancement.

(c) Initial groups (volumic cut).

(d) Groups after front advancement (volumic cut).

Figure 1: Example of mesh repartitioning by interface displacement in a sphere. Left: Initial partitioning. Right: Final partitioning.

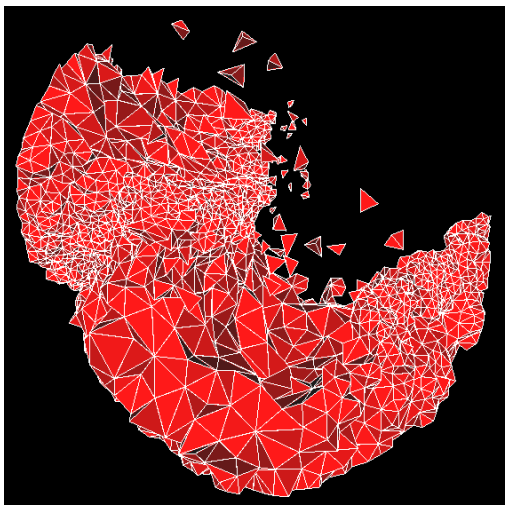


Figure 2: An example of disconnected mesh partition.

4.3 Correcting disconnected partitions

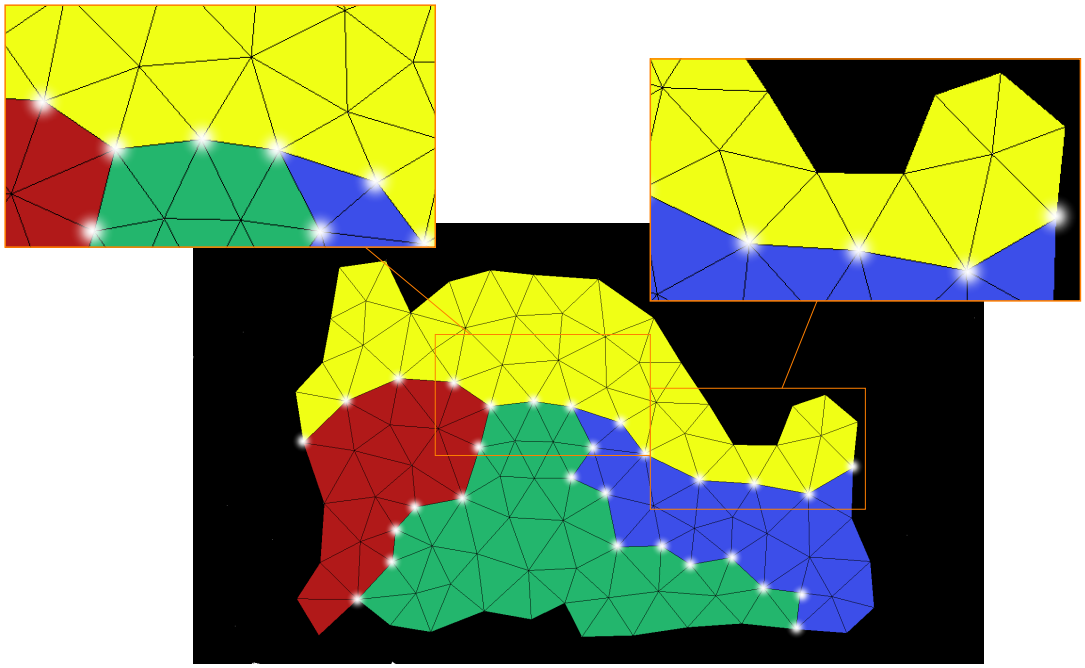
As already hinted in [Far88] [Ham92], experience shows that any advancing-front mesh partitioning algorithm is likely to produce disconnected partitions in given conditions. Although `Mmg` is capable of handling disconnected partitions, and it could be argued that a solver should be capable of handling generic mesh partitions, this is often not an optimal situation to perform computations. For example, in mesh adaptation over constrained interfaces one should aim at keeping the ratio between surface and volume elements as high as possible, which is not the case when disconnected mesh parts appear in a partition. Disconnected partitions also require special treatments for the localization step for metrics interpolation from the old to the new mesh (like tree-searches), which are not needed for connected partitions.

Some examples of common problems are presented in section 4.3.1. Some a priori corrections to prevent the occurrence of disconnected partitions have been hinted in [FL93], where the greedy algorithm is stopped and restarted if disconnected elements are detected. Similarly, in [Ozt95] the interface front advancement is said to be stopped if the target processor becomes disconnected from the sender processor. While this correction effectively prevents disconnections in the growing partition, it doesn't prevent shrinking partitions from becoming disconnected (as for example for a non-convex partition, as it will be shown in the next section). For these reasons, we have implemented in `ParMmg` an a posteriori correction of disconnected partitions, presented in section 4.3.2, to be performed only once after several layers of front advancement have been performed.

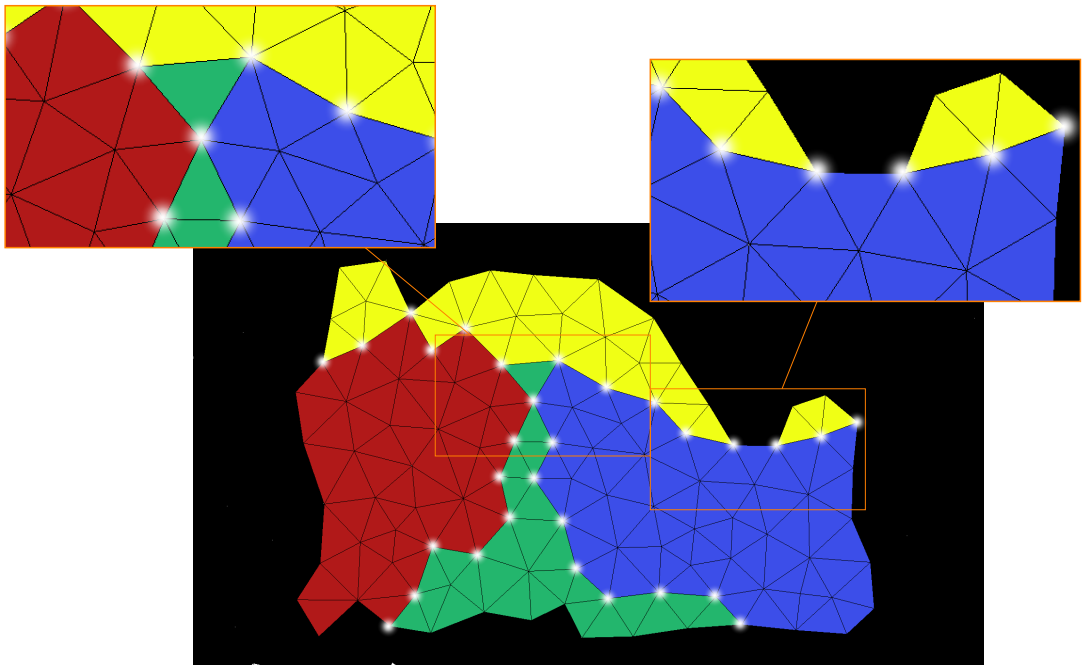
4.3.1 Examples of configurations leading to disconnected partitions

The next paragraph focuses on the presentation of the most common geometrical and partitioning configurations leading to disconnected meshes. For sake of simplicity, we use two-dimensional examples to illustrate this configurations.

Eaten non-convex partitions If the original partition is not convex, the interfaces could move inward so to cut away some partition extremities when the interface hits a boundary or when several interface fronts collide. This issue can be easily solved by checking and fixing



(a) Initial groups and interface front.



(b) Groups after one level of front advancement.

Figure 3: Two-dimensional illustration of some configurations leading to disconnected partitions. Right zoom: The extremity of a non-convex partition is cut away from its main domain by front advancement. Left zoom: unordered front advancement leading to an isolated bite (priority is red-blue-green-yellow).

the contiguity of the interior partition at the end of the interface displacement step. This configuration is illustrated in the right zoom in figure 3.

Isolated bites If care is not given to the order by which each node grows and marks its ball, it can happen that the color of a partition is propagated when it should not have been in the first place, and the following propagation of a higher-priority front takes some *isolated bites*[Far88] that disconnect the previously grown ball from its main partition. This configuration is illustrated in the left zoom in figure 3.

Parallel star configurations In the same way, if the front reaches an interface on a local partition and this information is not communicated to the remote processor, at the next propagation step the layer that propagated on the remote partition could not be reachable anymore by its original partition. This behavior is easily corrected by always performing a parallel update after one level of front advancement.

4.3.2 A posteriori corrections

From the practical point of view, it is difficult to predict all the situations that could lead to a disconnected partition in three dimensions. A more general and robust approach is that of a posteriori correcting partition connection after the front advancement step. This can be achieved by means of two kinds of checks, one on the contiguity of the local partition, one on the reachability of all layers that have propagated in the local partition by means of their remote counterpart. An a posteriori correction also allows for a generic choice of the interface propagation algorithm.

Contiguity fix for the local partition

1. Similarly to greedy algorithms, we choose a color and start from an element of this color: we attempt at touching all elements of this color by adjacency (through the element faces). Elements that we are able to reach by adjacency are stored in a list (*main_list*).
2. While we are able to find an element of the same color outside of *main_list*:
   - we build by adjacency a second list of elements of this color (*next_list*);
   - If *next_list* is not empty, we compare the lengths of *next_list* and *main_list* and we merge the smaller list of elements into an adjacent color. The biggest list is stored as *main_list* and we go back to step 2.
3. Go back to step 1 (choice of a color and construction of *main_list*) until all the colors have been treated.

Reachability fix for the inward-propagating remote partitions As hinted in [Ozt95], the front advancement can create subgroups cut away from any interface, or subgroups connected only to an interface of different color than their remote counterpart. These subgroups are not reachable by their remote partition anymore. To fix this problem, after a parallel exchange of remote colors on the current interface, for each interface face we attempt at building a list of elements of the same remote color which can be touched by adjacency, and we mark them as reachable. At the end of this process, we iteratively look for a list of adjacent elements which have not been reached by remote colors, and merge them into an adjacent color (reachable or not).

The current implementation is conceived in order to leave the interface displacement and the contiguity/reachability correction independent. This allows for a certain freedom in the experimentation of different advancing-front methods (for example, advancing on element adjacency instead of node adjacency). and interface sorting methods. Although a partition could be a priori split into an arbitrary number of disconnected parts, the contiguity correction algorithm only uses two lists to immediately compare two parts and merge the smallest into another color, thus limiting the memory usage. Similarly, the reachability correction algorithm only uses one list to store the current subgroup of unreachable elements. At the end of the two correction steps, all mesh elements are guaranteed to having been touched by a list only once, if the counter is decremented every time an element is merged in a different color.

5 Summary of the algorithm

The key features of the parallel remeshing algorithm can be summarized as follows:

- Two-level parallelization scheme (*distributed* mesh partitions and *local* mesh groups);
- Sequential remeshing of mesh groups over constrained parallel interfaces.;
- Mesh repartitioning by interface displacement.

A pseudocode for the `ParMmg` workflow is presented in algorithm 1. Mesh migration directly exploits the two-level parallelization schemes, as mesh parts to be sent to other processors are assembled into mesh groups to be used for parallel communication, while mesh repartitioning acts as an independent step which can be further tailored to ensure a more robust load balancing. A pseudocode for mesh repartitioning is presented in algorithm 2

Algorithm 1 `ParMmg` algorithm pseudocode.

```
Input(mesh,metrics);                                  /* Initialization (sequential or parallel) */
Group split;                                          /* Split partitions into groups */
for i = 0, ..., i_max - 1 do                          /* Iterative remeshing-repartitioning */
    Update old groups;                                /* Set background mesh for metrics interpolation */
    for igrp = 0, ..., ngrp - 1 do                    /* Loop on mesh groups */
        Mmg call;                                     /* Remeshing */
    end for
    Interpolate metrics;                              /* Recompute metrics */
    Mesh repartitioning;                              /* Interface displacement and mesh migration */
end for
Output(mesh,metrics);                                 /* Return the adapted mesh */
```

6 Open-source software implementation

The presented algorithm is implemented into the `ParMmg` software package for parallel unstructured mesh adaptation, released under the GNU Lesser General Public License (LGPL) and available at the repository `https://github.com/MmgTools/ParMmg`. The implementation is targeted to provide:

- Reusage of existing sequential remeshing libraries;

Algorithm 2 Mesh repartitioning pseudocode.

```
1: Input(mesh groups);
2: Sort interfaces;                                  /* Choose interface propagation direction */
3: Parallel update of interface colors;
4: for i_layer = 0, ..., n_layer − 1 do              /* Loop on mesh groups */
5:     Propagate node front;
6: end for
7: Correct contiguity;
8: Correct reachability;
9: Mesh migration;
10: Output(mesh groups);                             /* Return repartitioned groups */
```

- Non-intrusive linkage with third-party solvers;
- Improvable parallel performances by means of dynamic load balancing.

Open-source software packages are used for every step in the computing chain, from mesh partitioning, remeshing, node renumbering, mesh visualization. The remeshing kernel is the sequential `Mmg` library [DDF14] [Mmg]. Parallelization is performed through Message Passing Interface (MPI) libraries. Partitioning of a centralized input mesh is performed by means of the `Metis` library [Kar] [Met], and the `Scotch` library [Sco] is employed for nodes renumbering to reduce cache misses. Finally, mesh files can be saved for visualization in the Medit format (readable by `Medit` [Fre01] [Med] and `Gmsh` [GR09]) and in VTK format [Sch+06][Vtk]. Finally, version control is performed with `Git` and continuous integration testing with `Jenkins`.

7 Results

7.1 Qualitative comparison of graph-based repartitiong and interface displacement algorithms

In this section we compare the behaviour of graph-based repartitioning and repartitioning by interface displacement in effectively enabling mesh adaptation near old parallel interfaces. For graph-based repartitioning, we penalize each old parallel face f by computing an associated weight w_f as

$$w_f = \min\left(\frac{1}{\exp\left(\frac{\alpha}{n_a}\sum_{i=1}^{n_a} e_i\right)}, w_{\max}\right) \tag{1}$$

where index i runs on the edges of the face ($n_a = 3$ for a simplicial mesh), $\alpha = 28$ and $w_{\max} = 10^6$. Quantity e_i is built using the length of the edge $l_{\mathcal{M},i}$ in the metrics $\mathcal{M}$ as

$$e_i = \begin{cases} l_{\mathcal{M},i} - 1 & \text{if } l_{\mathcal{M},i} \leq 1 \\ \dfrac{1}{l_{\mathcal{M},i}} - 1 & \text{if } l_{\mathcal{M},i} > 1 \end{cases} \tag{2}$$

This expression is inspired by the effectiveness index introduced in [Dob05] and used in [Dob+06]. This weight is converted in integer format and directly applied to graph edges when a mesh partition is split into groups. When groups have to be repartitioned, instead, the sum of the weights on the parallel interface between two groups is applied on the graph edges.

The effectiveness of the proposed weight is shown in figure 7.1. The test case is a sphere of radius 10, with a *tennis ball* isotropic metrics

$$h(x,y,z) = h_{\max} - (h_{\max} - h_{\min}) \exp\left(\left(\hat{y} - \hat{x}^2 + \hat{z}^2\right)^2\right), \qquad \hat{x} = sx,\ \hat{y} = sy,\ \hat{z} = sz \quad (3)$$

with $s = 0.1$, $h_{\min} = 0.3$ and $h_{\max} = 1.5$. For tetrahedra on a parallel interface, the sum of the weights on parallel faces is visualized. For interior tetrahedra, an arbitrary unitary weight is visualized. The weight is capable of capturing the tetrahedra that do not respect the imposed metrics. This includes boundary regions, as boundary surfaces are not adapted in the current implementation.

Anyway, graph-based repartitioning has difficulties in converging towards a fully adapted mesh as the number of remeshing-repartitioning is increased when a significant number of partitions/groups is involved. This can be seen in figure 5, where meshes obtained after three iterations of remeshing and load balancing are compared with meshes obtained after three iterations of remeshing and interface displacement. Old parallel interfaces are clearly visible as poorly-adapted regions in graph-based partitions, while this effect is much less evident with direct interface displacement.

7.2 Parallel weak-scaling of a uniform refinement test case

For a preliminary evaluation of the parallel performances of the algorithm, we test its weak scalability for an uniform refinement case. The aim is that of distributing the work load on each mesh group as uniformly as possible.

The geometry is a sphere of radius 10. Input meshes have been generated with `Gmsh`, and their size grows with the number of processors (the input edge size is shown in table 1).

The assigned target mesh size is about 1/6 the original edge size. The size of the output adapted meshes are shown in table , compared with the sizes of the corresponding input meshes. The same results are visualized in figure 7, in order to check that the load is maintained approximately constant in the weak scaling test.

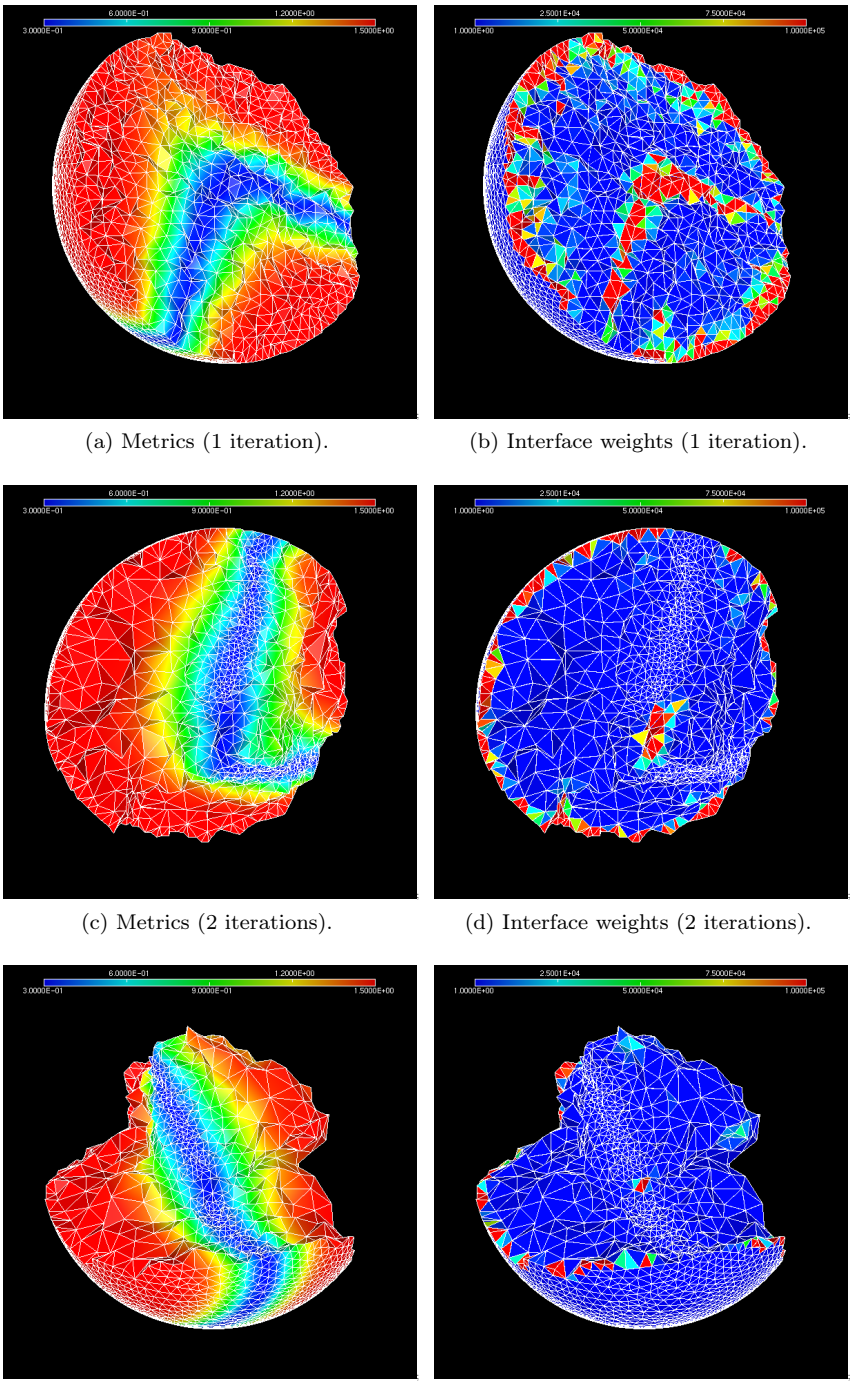
Tests have been performed on the Miriel nodes of PlaFRIM cluster[2], equipped with 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 (2.5 GHz) and 128 GB of RAM (see figure 6 for the lstopo view), connected through Infiniband QDR TrueScale (40 Gb/s) and Omnipath (100 Gb/s).

The computational time in the weak scaling test is shown in figure 8. The total time is split into the time spent in the remesher and the time spent in mesh repartitioning. In this phase, the time spent in migrating the mesh (MPI communication) is explicitly highlighted. It can be seen that the remeshing time is kept approximately constant with the number of cores, while the time for mesh migration constitute a bottleneck on more than 64 cores. The reasons for this behavior are currently under investigation. Even if performances can clearly still be optimized, parallel remeshing is able to generate billion-element meshes over 128 cores (table 2).

7.3 Preliminary results on anisotropic test cases

A qualitative assessment of the capabilities of our parallel remeshing algorithm to handle anisotropic mesh adaptation is given in this section. The implementation of interface displacement algorithm has not been tuned for anisotropic cases yet, so the following tests are mostly intended to verify that metrics interpolation is correct, and to assess the effects of the iterations in order to collect research directions for future developments.

[2]Supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/)

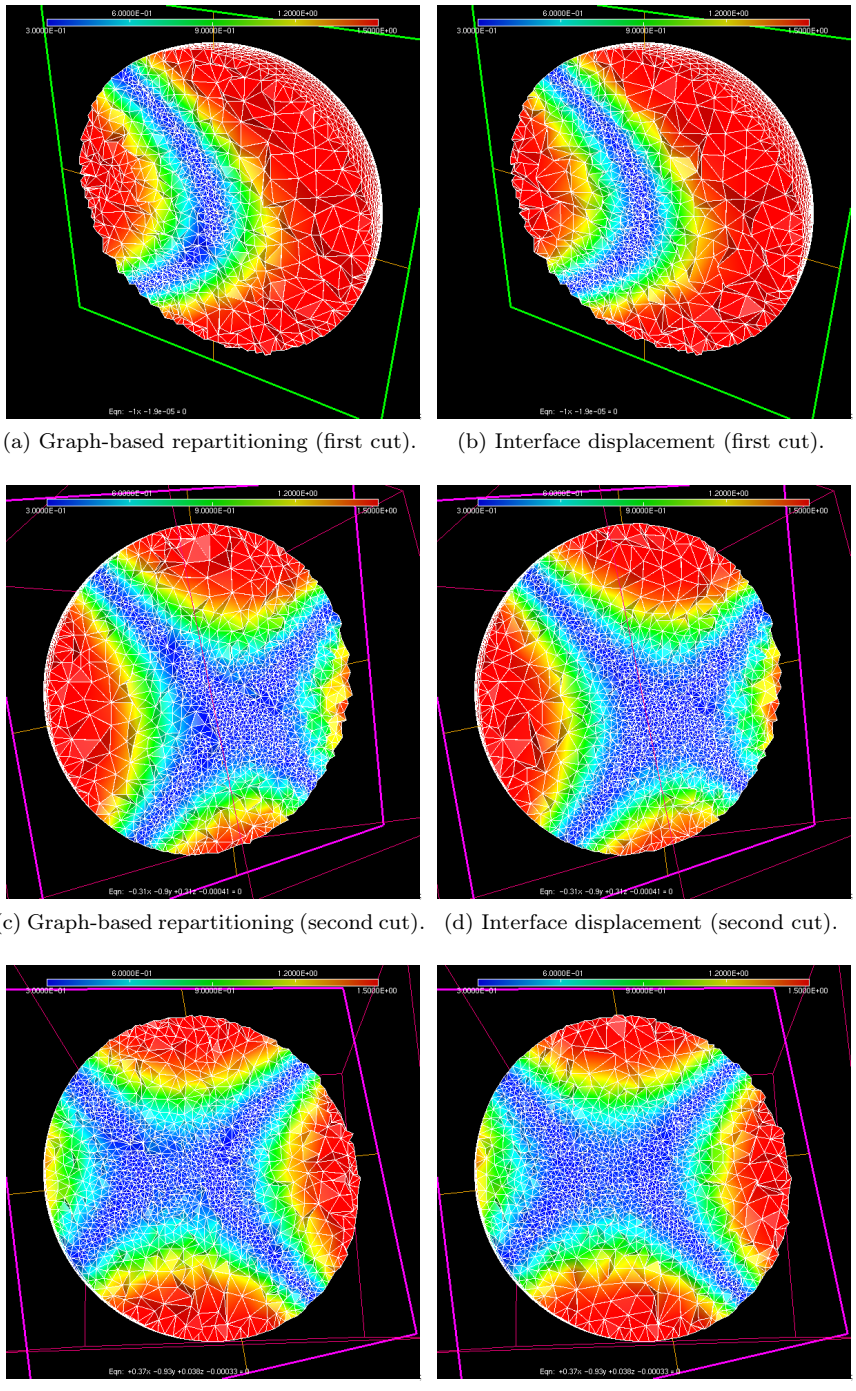


(a) Metrics (1 iteration). (b) Interface weights (1 iteration).

(c) Metrics (2 iterations). (d) Interface weights (2 iterations).

(e) Metrics (3 iterations). (f) Interface weights (3 iterations).

Figure 4: Mesh partitions produced by graph-based repartitioning for different number of remeshing-repartitionig iterations, for the isotropic tennis ball case. Left: isotropic metrics. Right: Interface weights.



(a) Graph-based repartitioning (first cut). (b) Interface displacement (first cut).

(c) Graph-based repartitioning (second cut). (d) Interface displacement (second cut).

(e) Graph-based repartitioning (third cut). (f) Interface displacement (third cut).

Figure 5: Comparison of graph-based repartitioning (left) and interface displacement for different volume cuts on the isotropic tennis ball case, for three iterations of remeshing-repartitioning on 8 cores (between 2 and 8 groups per partition are used). Colors represent the isotropic metrics.

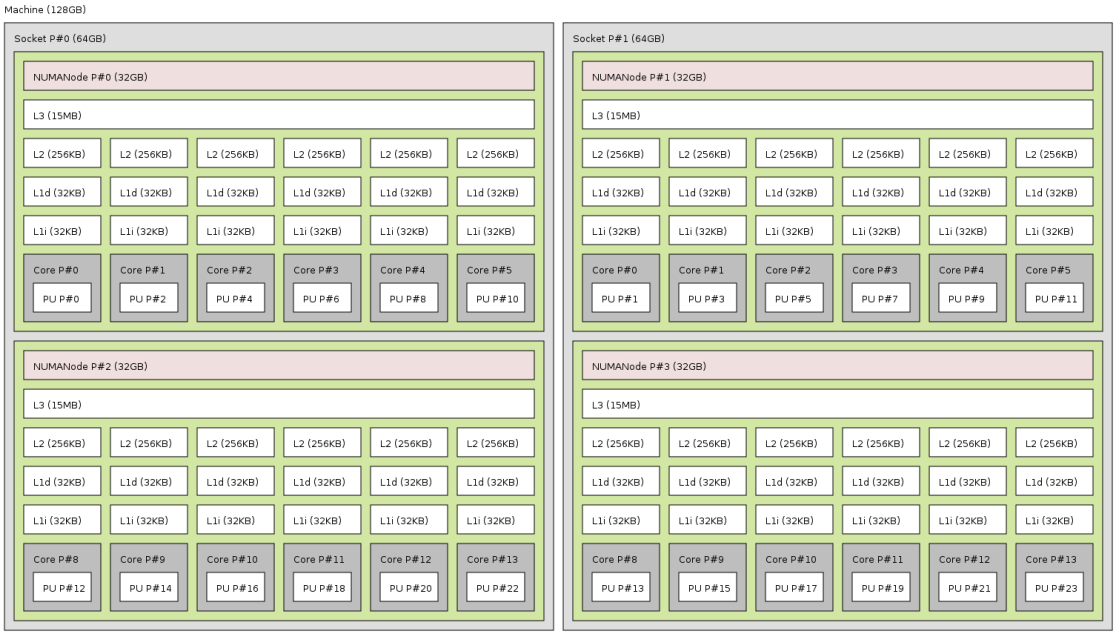


Figure 6: lstopo view of a miriel node

Cores	$h^{(\text{in})}$
1	0.16667
2	0.13228
4	0.10499
8	0.08333
16	0.066142
32	0.052497
64	0.041667
128	0.033071
256	0.026248

Table 1: Input mesh edge size with different number of cores.

Cores	Nodes (in)	Nodes (out)	Elements (in)	Elements (out)
1	3842	1591657	18990	9418375
2	7251	2645804	37752	15893278
4	13871	5439170	75195	32688539
8	26772	11147795	149329	67019257
16	52235	22703159	297586	136515675
32	102101	46023487	593845	276849115
64	205737	77677606	1192034	552962469
128	411553	164118099	2416482	1115114432
256	856547	328668480	5044573	2260289909

Table 2: Number of nodes and elements in the input and output meshes, with different number of cores.

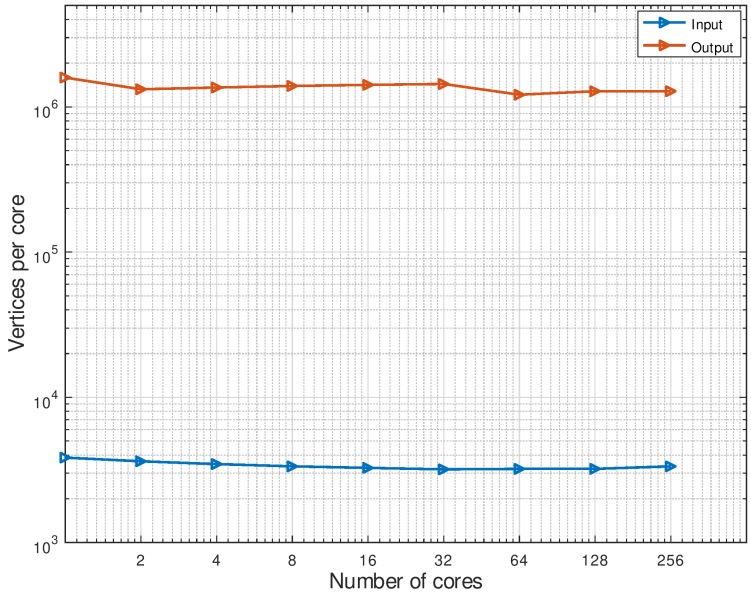


Figure 7: Number of nodes in the input and output meshes for the weak scaling test.

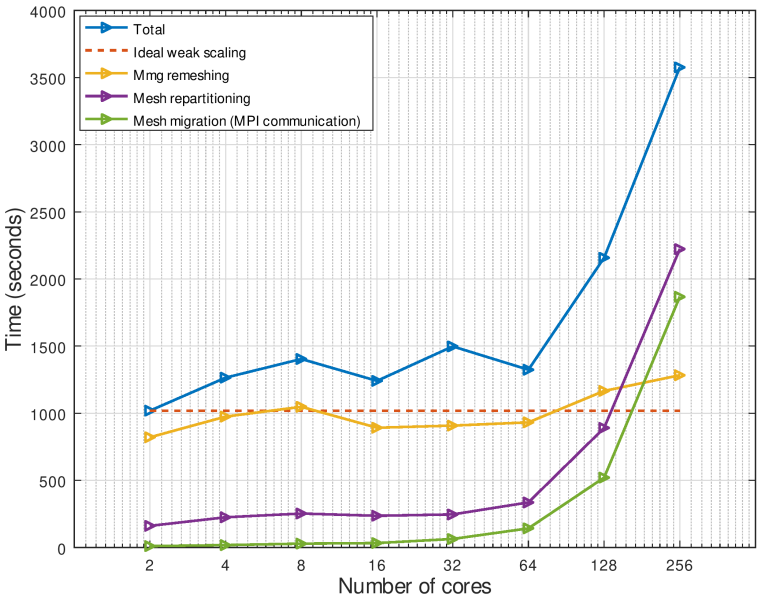


Figure 8: Computational time in the weak scaling test.

Uniform anisotropic metrics An example of anisotropic adaptation in a sphere of radius 10 and uniform metrics

$$\mathcal{M} = \begin{pmatrix} 25 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 0.4 \end{pmatrix} \tag{4}$$

is given in figure 9, for 6 iterations on 8 processors. It can be seen that still more iterations are needed to converge towards a fully adapted mesh, with respect to the isotropic case. In particular, we notice that some isotropic regions are visible near old parallel interfaces (left zoom in figure 9).

Continuous sinusoidal shock The last effect is confirmed when adapting on a more complex function, like the *continuous sinusoidal shock* from [Ala15]

$$f = \tanh(20(x + 0.3\sin(-10y) - 0.3\sin(-0.5(z - 0.1)))), \qquad \hat{x} = sx,\ \hat{y} = sy,\ \hat{z} = sz \tag{5}$$

with $s = 0.1$. The anisotropic metrics is computed from the Hessian matrix of the function following the approach in [AF03]. The eigenvalues λ_i of the Hessian matrix are truncated as

$$\tilde{\lambda}_i = \min\left(\max\left(\frac{c_d}{\epsilon}|\lambda_i|, \frac{1}{h_{\max}^2}\right), \frac{1}{h_{\min}^2}\right) \qquad i = 1, \dots, 3 \tag{6}$$

with $h_{\min} = 0.1$ and $h_{\max} = 1.5$. Parallel results, for 6 iterations on 8 processors, are compared with the sequential ones in figure 10. It can be noticed that the effects of parallel interfaces in spreading isotropic zones is evident. This is probably due to the way the metrics is corrected in `Mmg` on parallel interfaces, and an improvement of this correction is likely to improve the anisotropic results too.

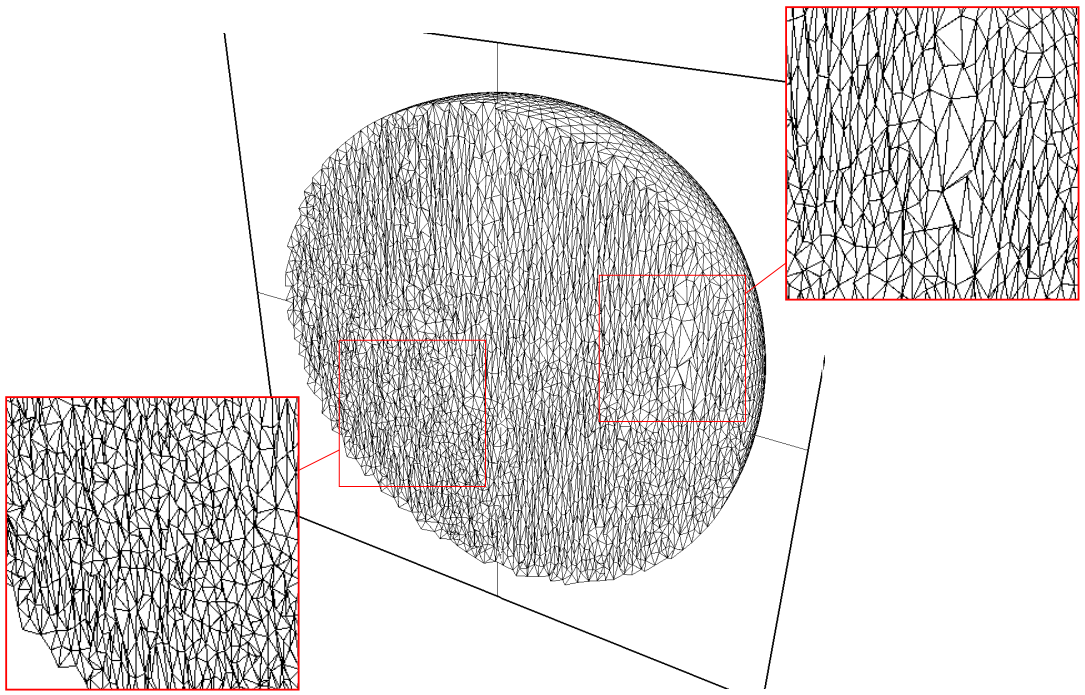
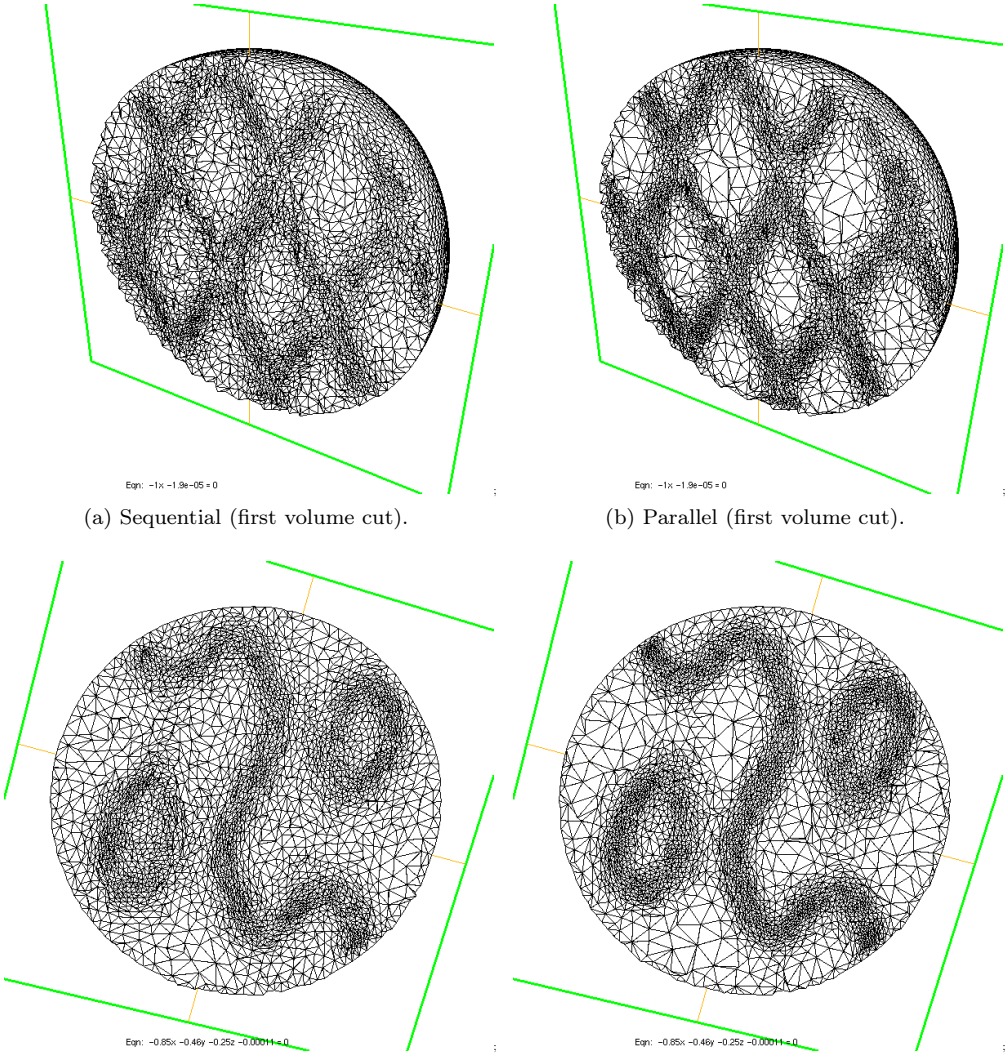


Figure 9: Uniform anisotropic mesh adaptation (volume cut), 6 iterations on 8 processors. Right zoom: A parallel interface is still visible. Left zoom: Isotropic zones induced by a parallel interface.



(a) Sequential (first volume cut).

(b) Parallel (first volume cut).

(c) Sequential (second volume cut).

(d) Parallel (second volume cut).

Figure 10: Adaptation on the *continuous sinusoidal shock* (volume cuts). Left: Sequential `Mmg`. Right: `ParMmg` with 6 iterations on 8 processors.

8 Conclusion and future developments

We have presented a parallel unstructured mesh adaptation algorithm based on iterative remeshing and mesh repartitioning. The algorithm rests on a two-level parallelization scheme allowing to tweak the mesh group size for remeshing, and on a mesh repartitioning scheme based on interface displacement by front advancement.

Specific corrections have been developed to avoid the occurrence of disconnected mesh groups during the interface propagation step, and the efficacy of an interface displacement repartitioning algorithm compared to a graph-based repartitioning approach has been shown.

Isotropic weak-scaling tests and anisotropic tests provide directions for future software improvement. New developments are foreseen in order to enable:

- Adaptation on boundary surfaces.
- Discretization of implicit surfaces described by a level-set function.
- Load balancing improvement during repartitioning.

The described algorithm is implemented into the open-source `ParMmg` software application and library, released under LGPL license, and available at the reporitory `https://github.com/MmgTools/ParMmg`.

Acknowledgements

This work was performed within the framework of the ExaQUte project, founded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 800898.

This work has received funding from the FUI22 ICARUS project, founded by Région Nouvelle-Aquitaine (RNA) under grant agreement 17001260-031 and the Fonds Unique intérministeriel (FUI) under grant agreement No DOS0050687100.

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/).

The authors would like to thank the scientific support of the YALES2 development team at the CORIA research center in the implementation of the parallel remeshing algorithm.

The authors would like to recognize the fundamental impulse that Cécile Dobrzynski brought to this work.

References

[AF03] Frédéric Alauzet and Pascal Frey. *Estimateur d'erreur géométrique et métriques anisotropes pour l'adaptation de maillage. Partie I : aspects théoriques*. Research Report RR-4759. INRIA, 2003. URL: `https://hal.inria.fr/inria-00071827`.

[AL09] Frédéric Alauzet and Adrien Loseille. "On the Use of Space Filling Curves for Parallel Anisotropic Mesh Adaptation". In: *Proceedings of the 18th International Meshing Roundtable*. Ed. by Brett W. Clark. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 337–357.

[Ala15] Frédéric Alauzet. *A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes*. Research Report RR-8785. INRIA Paris-Rocquencourt, Oct. 2015, 32 p. URL: `https://hal.inria.fr/hal-01211749`.

[AM10] F. Alauzet and M. Mehrenberger. "P1-conservative solution interpolation on unstructured triangular meshes". In: *International Journal for Numerical Methods in Engineering* 84.13 (2010), pp. 1552–1588. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2951. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2951.

[Ben+16] Pierre Benard et al. "Mesh adaptation for large-eddy simulations in complex geometries". In: *International Journal for Numerical Methods in Fluids* 81.12 (2016), pp. 719–740. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.4204. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4204.

[CDD00] T. Coupez, H. Digonnet, and R. Ducloux. "Parallel meshing and remeshing". In: *Applied Mathematical Modelling* 25.2 (2000). Dynamic load balancing of mesh-based applications on parallel, pp. 153 –175. URL: http://www.sciencedirect.com/science/article/pii/S0307904X00000457.

[CN03] Nikos Chrisochoides and Démian Nave. "Parallel Delaunay mesh generation kernel". In: *International Journal for Numerical Methods in Engineering* 58.2 (2003), pp. 161–176. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.765. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.765.

[CS97] José G. Castaños and John E. Savage. "The Dynamic Adaptation of Parallel Mesh-Based Computation". In: *PPSC*. 1997.

[CSF05] Peter A. Cavallo, Neeraj Sinha, and Gregory M. Feldman. "Parallel Unstructured Mesh Adaptation Method for Moving Body Applications". In: *AIAA Journal* 43.9 (2005), pp. 1937–1945. eprint: https://doi.org/10.2514/1.7818. URL: https://doi.org/10.2514/1.7818.

[DCS99] H. L. De Cougny and M. S. Shephard. "Parallel refinement and coarsening of tetrahedral meshes". In: *International Journal for Numerical Methods in Engineering* 46.7 (1999), pp. 1101–1125. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-0207%2819991110%2946%3A7%3C1101%3A%3AAID-NME741%3E3.0.CO%3B2-E. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0207%2819991110%2946%3A7%3C1101%3A%3AAID-NME741%3E3.0.CO%3B2-E.

[DDF14] C. Dapogny, C. Dobrzynski, and P. Frey. "Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems". In: *Journal of Computational Physics* 262 (2014), pp. 358 –378. URL: http://www.sciencedirect.com/science/article/pii/S0021999114000266.

[DF08] C. Dobrzynski and P. Frey. "Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations". In: *Proceedings of the 17th International Meshing Roundtable*. Ed. by Rao V. Garimella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 177–194.

[Dig+17] Hugues Digonnet et al. "Massively parallel anisotropic mesh adaptation". In: *International Journal of High Performance Computing Applications* (Mar. 2017). URL: https://hal.archives-ouvertes.fr/hal-01487424.

[Dob05] Cécile Dobrzynski. "Adaptation de maillage anisotrope 3D et application à l'aérothermique des bâtiments". Theses. Université Pierre et Marie Curie - Paris VI, Nov. 2005. URL: https://tel.archives-ouvertes.fr/tel-00120327.

[Dob+06] Cécile Dobrzynski et al. "Fast and accurate simulations of air-cooled structures". In: *Computer Methods in Applied Mechanics and Engineering* 195.23 (2006). Incompressible CFD, pp. 3168 –3180. URL: http://www.sciencedirect.com/science/article/pii/S0045782505002483.

[DR07] C Dobrzynski and J-F Remacle. *Parallel mesh adaptation*. Poster. 2007. URL: https://www.math.u-bordeaux.fr/~dobrzyns/telechargement/parallmeshposter.pdf.

[FA05] Pascal J. Frey and Frédéric Alauzet. "Anisotropic mesh adaptation for CFD computations". In: 2005.

[Far88] Charbel Farhat. "A simple and efficient automatic fem domain decomposer". In: *Computers & Structures* 28.5 (1988), pp. 579 –602. URL: http://www.sciencedirect.com/science/article/pii/0045794988900041.

[FG07] Pascal Jean Frey and Paul-Louis George. *Mesh Generation: Application to Finite Elements*. ISTE, 2007.

[FL93] Charbel Farhat and Michel Lesoinne. "Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics". In: *International Journal for Numerical Methods in Engineering* 36.5 (1993), pp. 745–764. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620360503. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620360503.

[Fla+98] J.E. Flaherty et al. "Parallel structures and dynamic load balancing for adaptive finite element computation". In: *Applied Numerical Mathematics* 26.1 (1998), pp. 241 –263. URL: http://www.sciencedirect.com/science/article/pii/S0168927497000949.

[Fre01] Pascal Frey. *MEDIT : An interactive Mesh visualization Software*. Tech. rep. RT-0253. INRIA, Dec. 2001, p. 41. URL: https://hal.inria.fr/inria-00069921.

[GR09] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2579. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579.

[Ham92] Steven Warren Hammond. "Mapping Unstructured Grid Computations to Massively Parallel Computers". UMI Order No. GAX93-02685. PhD thesis. Troy, NY, USA: Rensselaer Polytechnic Institute, 1992.

[Kar] G Karypis. *METIS 5.1.x Manual*. URL: https://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf.

[Lac+17] Cédric Lachat et al. *Fast parallel remeshing for accurate large-eddy simulations on very large meshes*. Research Report RR-9133. Inria Bordeaux Sud-Ouest, Dec. 2017, p. 13. URL: https://hal.inria.fr/hal-01669775.

[LDP14] C Lachat, C Dobrzynski, and F Pellegrini. "Parallel mesh adaptation using parallel graph partitioning". In: *5th European Conference on Computational Mechanics (ECCM V)*. Ed. by Eugenio Oñate, Xavier Oliver, and Antonio Huerta. Vol. 3. Minisymposia in the frame of ECCM V. ISBN 978-84-942844-7-2. IACM & ECCOMAS. Barcelone, Spain: CIMNE - International Center for Numerical Methods in Engineering, July 2014, pp. 2612–2623. URL: https://hal.inria.fr/hal-01099259.

[LM13] Adrien Loseille and Victorien Menier. "Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive". In: *IMR*. 2013.

[LMA15] Adrien Loseille, Victorien Menier, and Frédéric Alauzet. "Parallel generation of large-size adapted meshes". In: *Procedia Engineering* 124.57 – 69 (2015). URL: https://hal.inria.fr/hal-01270708.

[Löh13] Rainald Löhner. "A 2nd generation parallel advancing front grid generator". In: *Proceedings of the 21st international meshing roundtable*. Springer, 2013, pp. 457–474.

[Löh14] Rainald Löhner. "Recent Advances in Parallel Advancing Front Grid Generation". In: *Archives of Computational Methods in Engineering* 21.2 (2014), pp. 127–140. URL: https://doi.org/10.1007/s11831-014-9098-8.

[MDV11] Vincent Moureau, Pascale Domingo, and Luc Vervisch. "Design of a massively parallel CFD code for complex geometries". In: *Comptes Rendus Mécanique* 339.2 (2011). High Performance Computing, pp. 141 –148. URL: http://www.sciencedirect.com/science/article/pii/S1631072110002111.

[Med] *Medit software package for scientific visualization on unstructured meshes*. URL: https://github.com/ISCDtoolbox/Medit.

[Met] *Metis software package for graph partitioning*. URL: https://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

[Mmg] *Mmg software package for simplicial remeshing*. URL: https://www.mmgtools.org/.

[OBG00] Leonid Oliker, Rupak Biswas, and Harold N Gabow. "Parallel tetrahedral mesh adaptation with dynamic load balancing". In: *Parallel Computing* 26.12 (2000). Graph Partitioning and Parallel Computing, pp. 1583 –1608. URL: http://www.sciencedirect.com/science/article/pii/S0167819100000478.

[Ozt95] Can Ozturan. "Distributed Environment and Load Balancing for Adaptive Unstructured Meshes". UMI Order No. GAX96-22925. PhD thesis. Troy, NY, USA: Rensselaer Polytechnic Institute, 1995.

[Par+16] Michael A Park et al. "Unstructured Grid Adaptation: Status, Potential Impacts, and Recommended Investments Towards CFD 2030". In: *AIAA Fluid Dynamics Conference, AIAA AVIATION Forum*. Washington DC, United States, 2016. URL: https://hal.inria.fr/hal-01438667.

[Pel09] François Pellegrini. "Contributions au partitionnement de graphes parallèle multi-niveaux". Habilitation à diriger des recherches. Université Sciences et Technologies - Bordeaux I, Dec. 2009. URL: https://tel.archives-ouvertes.fr/tel-00540581.

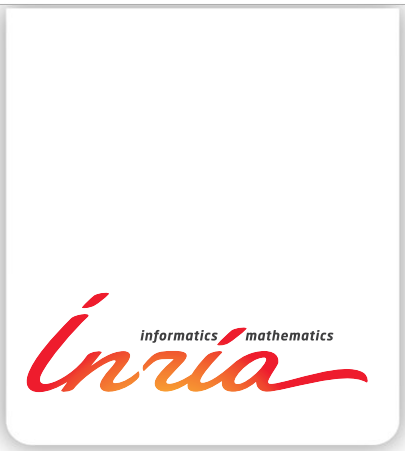
[Sch+06] W. Schroeder et al. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Ed. by Kitware. Kitware, 2006.

[Sco] *Scotch software package for graph partitioning*. URL: https://gitlab.inria.fr/scotch/scotch.

[She+95] MS Shephard et al. "Parallel automated adaptive procedures for unstructured meshes". In: *Parallel Computing in CFD* 807 (1995), pp. 6–1.

[Sim91] H.D. Simon. "Partitioning of unstructured problems for parallel processing". In: *Computing Systems in Engineering* 2.2 (1991). Parallel Methods on Large-scale Structural Analysis and Physics Applications, pp. 135 –148. URL: http://www.sciencedirect.com/science/article/pii/095605219190014V.

[Vtk] *VTK software package for scientific visualization*. URL: https://vtk.org/.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399