14th World Congress on Computational Mechanics (WCCM)
ECCOMAS Congress 2020)
Virtual Congress: 11-–15 January 2021
F. Chinesta, R. Abgrall, O. Allix and M. Kaliske (Eds)

PARALLEL UNSTRUCTURED MESH ADAPTATION BASED ON ITERATIVE REMESHING AND REPARTITIONING

Luca Cirrottola[1]*, Algiane Froehly[2]

[1] INRIA, Université de Bordeaux, CNRS, Bordeaux INP, IMB UMR 5251, 200 Avenue de la Vieille Tour, 33405 Talence cedex, France, luca.cirrottola@inria.fr

[2] INRIA Direction Générale Déléguée à l'Innovation, InriaSoft, Mmg Consortium, algiane.froehly@inria.fr

Key words: Parallel mesh adaptation, unstructured meshes, mesh migration

Abstract. We present 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. The numerical procedure is implemented in the open source ParMmg software package. It enables the reuse of existing sequential remeshing libraries, a non-intrusive linkage with third-party solvers, and a tunable exploitation of distributed parallel environments. We show the efficiency of the approach by comparing interface displacement repartitioning with graph-based repartitioning, and by showing isotropic weak-scaling tests and preliminary anisotropic tests.

1 INTRODUCTION

Modern computational mechanics solvers nowadays routinely exploit parallel, distributed memory computer architectures. Even if a parallel solver could in principle use a sequential remesher by gathering the distributed mesh on a single process, due to memory limitations it is not always possible to guarantee that a large distributed mesh can be gathered on a single computing node. Beside this primary feasibility consideration, sequential remeshing in a parallel simulation also represents a significant performance bottleneck [1]. Parallel remeshing is thus becoming increasingly demanded.

Among the many previous works on parallel remeshing (for example [2] [3] [4] [5] [6] [7] [8] [9]), a broad classification can be made into methods that aim at parallelizing the remeshing kernel itself, and methods that aim at employing a sequential remeshing kernel in a parallel framework by leveraging a parallel repartitioning phase. We have opted for a modular approach by adopting an iterative remeshing-repartitioning scheme that does not modify the adaptation kernel [9], allowing the reuse of an existing sequential remeshing kernel. Our method is implemented into the parallel `ParMmg` application and library [10], built on top of the sequential `Mmg3d` remesher [11] for tetrahedral mesh adaptation. Both software are free and open source.

The rest of this paper is organized as follows. Section 2 briefly recalls the parallelization algorithm, which is detailed in [12]. Section 3 discusses the parallel performances at the current level of implementation, while section 4 sums up our observations and future research lines.

2 METHOD

As described in depth in [12], at each iteration of the parallel algorithm the sequential remeshing kernel is applied on the interior partition on each process, while maintaining fixed (non-adapted) parallel interfaces. Then the adapted mesh is repartitioned in order to move the non-adapted frontiers to the interior of the partitions at the next iteration, thus eliminating the presence of non-adapted zones as the iterations progress. Although the repartitioning step can be accomplished through standard mesh partitioning libraries, differently from [9] we have explored the usage of a direct parallel interface displacement method (or *diffusion* algorithm) to explicitly prioritize interface displacement over load balancing.

3 PARALLEL PERFORMANCE ASSESSMENT

We show the performances of the parallel algorithm on two academic edge sizemaps, by means of a weak scaling and a strong scaling test. Both the weak and strong scaling tests have been performed with the release v1.3.0 of `ParMmg`[10].

3.1 Weak scaling

A weak scaling test is shown in table 1. A sphere of radius 10 is uniformly refined while keeping the workload of each process as constant as possible as the number of processes is increased. The test is performed on the `bora` nodes of the `PlaFRIM` cluster[1]. The weak scaling performances are shown in figure 1a. The slow increase in the time spent in the repartitioning and redistribution phase still leaves space for some implementation optimization. Anyway, a major improvement is visible with respect to the preliminary results shown in [12], where a significant performance degradation was present on more than 64 cores. This improvement is due to an algorithmic optimization concerning the pre-computation of the pair of processes interacting in the communication phase, which replaces a direct probing on the presence of data to exchange whose cost grew quadratically.

3.2 Strong scaling

A strong scaling test is performed with an isotropic sizemap h describing a double Archimedean spiral

$$h(x,y) = \min(1.6 + |\rho - a\theta_1| + 0.005, 1.6 + |\rho + a\theta_2| + 0.0125) \tag{1}$$

with

$$\begin{aligned} \theta_1 &= \phi + \pi\left(1 + \text{floor}\left(\tfrac{\rho}{2\pi a}\right)\right) \\ \theta_2 &= \phi - \pi\left(1 + \text{floor}\left(\tfrac{\rho}{2\pi a}\right)\right) \end{aligned} \tag{2}$$

and $\phi = \text{atan2}(y,x)$, $\rho = s\sqrt{x^2+y^2}$, $a = 0.6$, $s = 0.5$, into a sphere of radius 10 with uniform unit edge length. The surfacic adaptation and a volumic cut of the adapted meshes on 1 and 1024 processes are shown in figure 2. The resulting edge length statistics are shown in table 2. The strong scaling performances are shown in logarithmic scale in figure 1b, where the speedup S_p over p processes is

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

p	n_v^{in}/p	n_v^{out}/p	n_v^{out}/n_v^{in}	n_v^{out}	n_e^{in}/p	n_e^{out}/p	n_e^{out}/n_e^{in}	n_e^{out}
2	3625	1293637	356.81	2587274	18876	7780974	412.21	15561948
4	3467	1341637	386.88	5366549	18798	8072081	429.39	32288325
8	3346	1380055	412.38	11040444	18666	8306084	444.98	66448675
16	3264	1412516	432.66	22600269	18599	8503695	457.2	136059129
32	3190	1437267	450.46	45992552	18557	8654569	466.36	276946210
64	3214	1431186	445.2	91595935	18625	8619098	462.75	551622317
128	3215	1444674	449.31	184918370	18878	8701524	460.91	1113795077
256	3345	1468905	439.01	376039759	19705	8848464	449.03	2265206835
512	3375	1446532	428.52	740624790	19998	8714450	435.74	4461798709
1024	3335	1449215	434.54	1483996788	19821	8731162	440.49	8940710661

Table 1: Mesh statistics for the `ParMmg` weak scaling test. Number of vertices n_v and tetrahedra n_e in input and output using p processors.

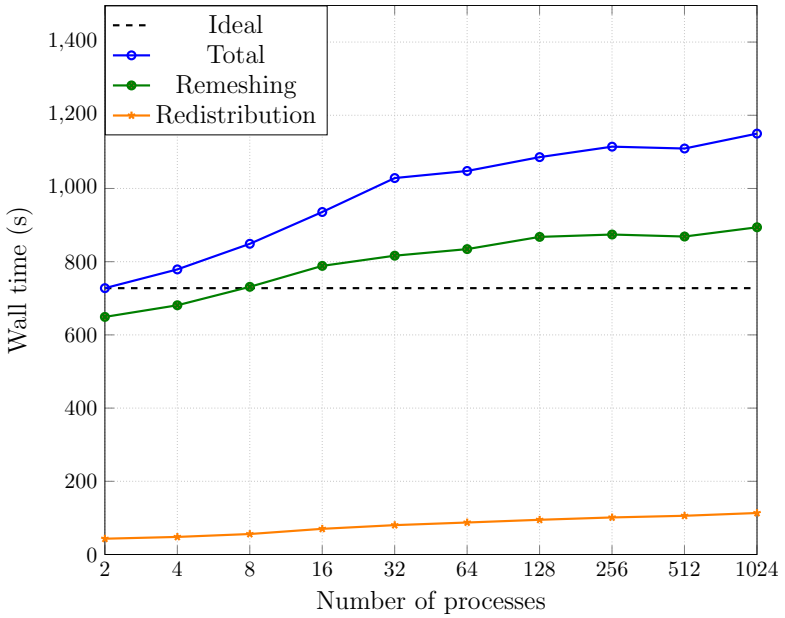
p	$N^{(0,0.3]}$	$N^{(0.3,0.6]}$	$N^{(0.6,0.7]}$	$N^{(0.71,0.9]}$	$N^{(0.9,1.3]}$	$N^{(1.3,1.41]}$	$N^{(1.41,2]}$	$N^{(2,5]}$	$N^{>5}$
1	1.34 %	35.34 %	16.89 %	25.49 %	20.15 %	0.63 %	0.16 %	0	0
2	1.14 %	34.87 %	16.97 %	25.79 %	20.45 %	0.63 %	0.15 %	0	0
4	1.04 %	34.20 %	17.03 %	26.08 %	20.85 %	0.64 %	0.16 %	0	0
8	0.98 %	33.66 %	17.06 %	26.30 %	21.18 %	0.66 %	0.16 %	0	0
16	1.07 %	32.41 %	17.06 %	26.78 %	21.84 %	0.68 %	0.16 %	0	0
32	0.91 %	30.78 %	17.29 %	27.59 %	22.57 %	0.70 %	0.16 %	0	0
64	0.93 %	30.22 %	17.20 %	27.76 %	23.01 %	0.72 %	0.17 %	0	0
128	0.88 %	29.48 %	17.20 %	28.06 %	23.48 %	0.73 %	0.17 %	< 0.01 %	0
256	0.70 %	27.63 %	17.20 %	28.84 %	24.67 %	0.77 %	0.18 %	< 0.01 %	< 0.01 %
512	0.66 %	27.19 %	17.14 %	28.96 %	25.04 %	0.80 %	0.20 %	< 0.01 %	< 0.01 %
1024	0.69 %	26.14 %	16.91 %	29.03 %	25.99 %	0.92 %	0.30 %	0.02 %	< 0.01 %

Table 2: Mesh statistics for the `ParMmg` strong scaling test. Percentage of edges $N^{(a,b]}$ whose length in the assigned metrics falls in the interval $l_{\mathcal{M}} \in (a,b]$, for each simulation on p processors.

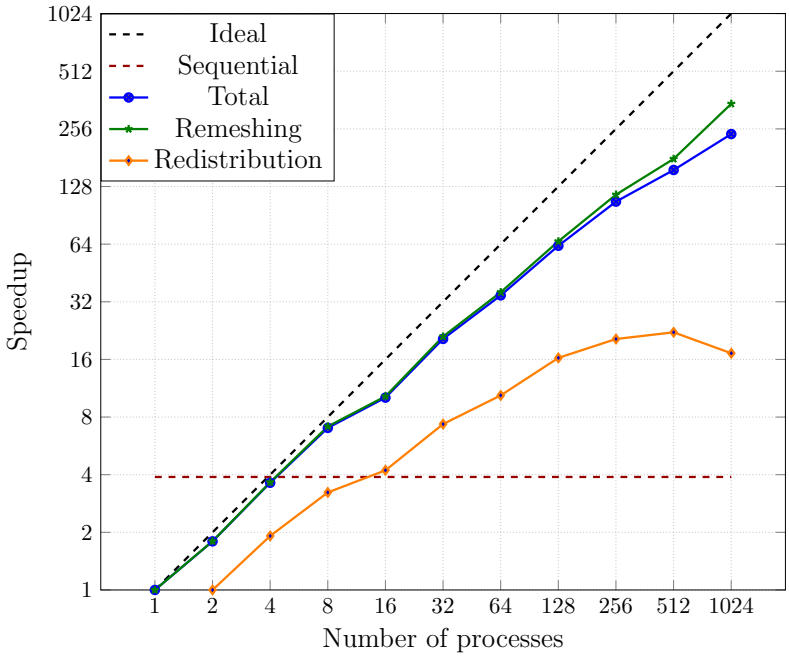
defined as the ratio between the wall time on 1 process and the wall time on p processes

$$S_p = \frac{T_1}{T_p} \tag{3}$$

except for the speedup of the redistribution part of the program, which is defined with respect to the redistribution time on 2 processes instead of 1 (as there is no redistribution on 1 process). As the number of interfaces increases with the number of processes, it can be noticed both in table 2 and in figure 1b that there is still a trend for a more pronounced propagation of large edge sizes from the fixed interfaces as the number of processes increases, as well as a performance reduction in the redistribution phase. These issues are the subject of current improvement efforts.

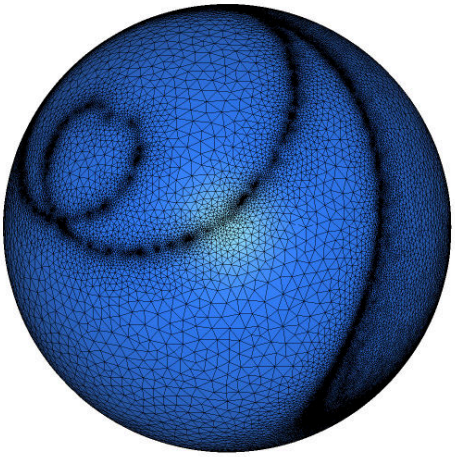


(a) Weak scaling performances for the uniform refinement of a sphere.

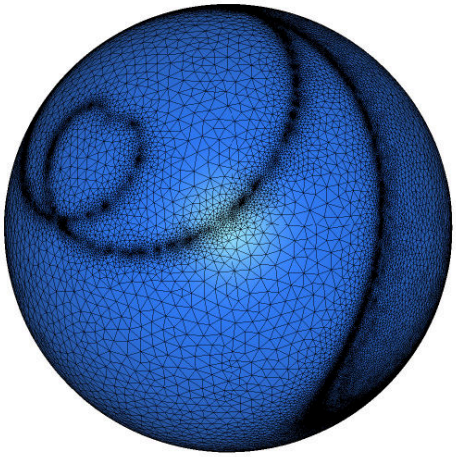


(b) Strong scaling performances for the double Archimedean spiral in a sphere.

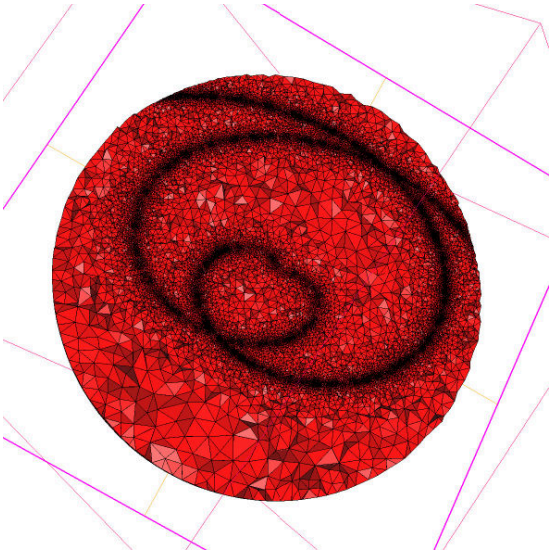
Figure 1: Weak and strong scaling performances of `ParMmg` for two different tests.



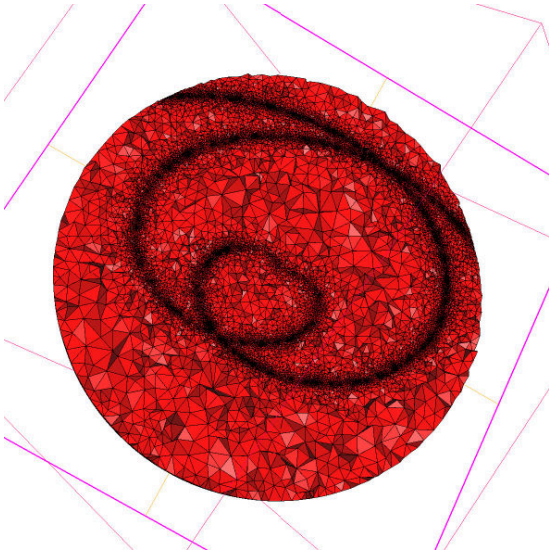
(a) Surface adaptation on 1 process.



(b) Surface adaptation on 1024 processes.



(c) Volume adaptation on 1 process.



(d) Volume adaptation on 1024 processes

Figure 2: Examples of adaptation to the double Archimedean spiral on 1 and 1024 processes.

3.3 Extension to anisotropic metrics

As already shown in [12], the application to anisotropic metrics can suffer from a more serious size diffusion effect than the one shown in the previous section, if specific precautions are not taken when dealing with the metric specification on parallel interfaces. This is the subject of current studies.

4 CONCLUSIONS

We have shown a performance assessment of a parallel remeshing algorithm for unstructured meshes based on iterative remeshing-repartitioning first described in [12]. Parallel iterations are performed by alternating remeshing on interior process partitions, with fixed parallel interfaces, to mesh repartitioning aiming at moving previous interfaces on the interior of the next partitions to be remeshed. A key point for the choice of this specific parallelization strategy rests in software modularity, since building a parallelization framework on top of an existing sequential remeshing kernel allows the parallel software application to benefit from the continuous development of its sequential kernel. Modularity also applies to the choice of the mesh repartitioning strategy, since it is independent from the internal mechanics of the remeshing kernel. While weak scaling results show that meshes of billions of tetrahedra are easily achievable through the current framework, they also show that care needs to be taken in the design of the parallel redistribution phase, as more data and more processes are involved in the communication phase. Strong scaling results show that the parallel application exhibits an overhead with respect to its sequential counterpart, due to the fact that several remeshing iterations are performed. Although this overhead is quickly catched up as the number of processes increases, it has to be kept in mind when the coupling with an external solver is envisaged, as its relative weight on overall performances can depend on the performances of the external solver itself, on the target mesh size, and on the parallel communicator size. Future research lines involve further improvement of the scalability of the mesh redistribution phase, which could be achieved by further optimizing the software implementation of the parallel communication, as well as the reduction of the parallel overhead, which could be achieved by optimizing the number of iterations or the interface displacement phase. Future developments also concern the full support of non-smooth boundary surfaces and anisotropic metrics.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] *ParMmg version 1.3.0*. SWHID:swh:1:rel:bf45d6314a386455d53a6c351be771d6252e0d43; REPOSITORY: https://github.com/MmgTools/ParMmg.

[11] *Mmg version 5.5.2*. SWHID: swh:1:rel:fe173a75f45f079d363d5a82204c9737550c5d79; REPOSITORY: https://github.com/MmgTools/mmg.

[12] Luca Cirrottola and Algiane Froehly. *Parallel unstructured mesh adaptation using iterative remeshing and repartitioning*. Research Report RR-9307. INRIA Bordeaux, équipe CARDAMOM, Nov. 2019. URL: https://hal.inria.fr/hal-02386837.