The 8th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2022
5–9 June 2022, Oslo, Norway

# A RUNTIME-BASED DYNAMIC MESH-PARTITIONING APPROACH

**GIACOMO BALDAN[1], RICARD BORRELL[2] AND JENS JÄGERSKÜPPER[*1]**

[1] German Aerospace Center (DLR)
Institute of Aerodynamics and Flow Technology
38108 Braunschweig, Germany
*jens.jaegerskuepper@dlr.de

[2] Barcelona Supercomputing Center
08034 Barcelona, Spain

**Key words:** Mesh Partitioning, Space Filling Curves, High Performance Computing (HPC), Computational Fluid Dynamics (CFD)

**Abstract.** Large-scale parallel numerical simulations are fundamental for the understanding of a wide variety of aeronautical problems. Mesh decomposition is applied to make use of parallel hardware. In particular, when using a massively parallel architecture, not only the final quality of the mesh subdivision is relevant. Also the partitioning algorithm itself needs to be robust as well as efficient. A strategy for dynamic mesh partitioning based on runtime measurements is presented. We integrate the "Geometric Mesh Partitioner" (GeMPa), which is a partitioning library based on Hilbert Space-Filling Curve (HSFC), in the FlowSimulator (FS) software. FS is a platform designed to run multi-disciplinary simulations on massively parallel cluster architectures. The algorithm performance is evaluated on an unstructured mesh representing the ONERA M6 wing. In particular, the load imbalance among processes is evaluated and compared with a well-known graph-based partitioning approach. Finally, we analyze how the number of processes influences the load imbalance.

## 1 INTRODUCTION

During the last decades, the rapid advance of computational fluid dynamics (CFD) technology has fundamentally changed the aerospace design process. A massive use of CFD has allowed for drastic reductions in wind tunnel time for aircraft development programs, as well as lower numbers of experimental rig tests in gas turbine engine design processes. The evolution in simulation capabilities is intimately related to the improvements in both, hardware and software [1].

Domain decomposition is a fundamental part in almost all parallel applications [2]. Furthermore, to fully leverage today's high-performance computing (HPC), most toolchains incorporate a second level of hybrid parallelism, such as OpenMP [3], or make use of GPUs [4, 5] or FPGAs [6] in heterogeneous architectures.

Domain decomposition methods are mainly categorized in two groups in literature: graph-based and geometric [7]. In graph-based partitioning, the first step consists in extracting the

graph from the domain connectivity. Two alternative graphs can be built: the nodal graph and the dual graph. In the former, which is usually employed in methods with nodal data representations, each mesh node corresponds to a graph vertex and all mesh edges coincide with graph edges. In the latter, which is an appropriate choice for cell-centered numerical methods, each graph vertex refers to a mesh element and graph edges represent the mesh faces. After the graph extraction, the partitioning is computed. This task is generally addressed using multilevel heuristics, where each level consist of three phases: coarsening, partitioning, and uncoarsening [8]. The partitioning usually tries to grant the same graph vertex weights in each process, to avoid imbalances in the computation, as well as the minimum cut of edge weights, to reduce the communication among processes. Several libraries are publicly available, for instance Parmetis [9], Zoltan [10], and PT-Scotch [11]. The alternative approach are geometric partitioning techniques, which obviate topological interactions between mesh entities and, thus, avoid the graph extraction from the computational mesh [12, 13]. The partitioning is performed considering only a representative coordinate for each mesh entity. Common criteria are the integration node coordinates if the CFD solver is node-based or, alternatively, the average of the integration node coordinates for an element if the code is cell-based. Geometric methods are usually fast and have low memory consumption. In addition, for small data changes, the domain cuts move only slightly, resulting in reduced data redistribution. When geometric methods are used for complex geometries, however, they can generate disconnected domains leading to an increased communication volume compared to graph-based ones. An example of a geometric method is the Recursive Coordinate Bisection (RCB), which consists in recursively subdividing the computational domain into respectively two parts until the desired number of partitions is reached. Another example are Space-Filling Curve (SFC) methods which map a multi-dimensional space into a one-dimensional one preserving geometric locality [14]. There are many possible definitions of an SFC, based on different mapping options, among them the well-known Peano and Hilbert ones.

In this work, we focus on a strategy for dynamic mesh partitioning based on runtime measurements. This approach mitigates the limitations imposed by a-priori estimated weights. In fact, assigning optimal weights to each element of a mesh is difficult when using complex algorithms. For instance, when adaptive techniques are adopted, such as *hp*-adaptive finite-element methods (*hp*-FEM), multiple refinement levels may be present. Other examples are multigrid methods, in which non-uniform agglomeration rates can be exploited, or the different cost for the reconstruction of boundary conditions. In addition, partitioning weights are usually influenced by the underlying computer architecture. When using a dynamic approach, the mesh subdivision is performed multiple times during the simulation. Therefore, it is necessary to keep the repartitioning time as low as possible in order to achieve a reduction of the overall computational time.

To address these challenges, we integrate the Hilbert Space-Filling Curve (HSFC) based partitioning library "Geometric Mesh Partitioner" (GeMPa) in the FlowSimulator (FS) software. FS is a platform designed to run multi-disciplinary simulations on massively parallel cluster architectures [15]. A relevant plugin to the FS platform is the flow solver CODA which is commonly developed by ONERA, DLR, Airbus and bases upon the infrastructure of the flexible unstructured CFD software *"Flucs"* [16]. Mesh handling is performed through the FlowSimulator Data Manager (FSDM). A new plugin has been developed to interface the GeMPa library [17].

GeMPa is a parallel geometric partitioning library based on the Hilbert Space Filling Curve. The GeMPa partitioning procedure can be summarized in the following steps: a bounding box is defined enclosing the unstructured mesh to be partitioned; a regular grid is constructed inside the bounding box; the grid bins are weighted according to the elements contained in them; the HSFC is used to project the bins to a 1D space; finally the 1D partitioning problem is solved. The parallel implementation divides the initial bounding box into sub-boxes, and each parallel process performs a local partition within its sub-box. A coherent partition of the overall mesh is obtained by connecting the local partitions. In this manner, the solution achieved is independent (discounting rounding off errors) of the number of parallel processes used to compute it. The main overhead regarding the parallelization is the point-to-point communications required to distribute the initial data into the sub-boxes. In a recent optimization, collective reduction communications are used to define the SFC splitting points, reducing the point-to-point communication requirements. The reader is referred to the reference [18] for a comprehensive description of the initial implementation of GeMPa.

This paper is organized as follows: in Section 2 we explain the adopted dynamic repartitioning approach and how runtime measurements are converted into weights in the partitioning process; in Section 3 an overview of the performance for available partitioning algorithms in FS (which can be used with CODA) is given; in Section 4 the ONERA M6 wing test case performance measurements are reported and discussed. Finally, general conclusions are outlined in Section 5.

## 2 DYNAMIC APPROACH

The classical static mesh-partitioning approach consists in subdividing the domain in a preparatory step and keeping the mesh distribution unchanged for the entire simulation. In contrast, when a dynamic mesh-partitioning approach is chosen, the partitioning step is included in the evolving part of the simulation. An iterative process tries to achieve an improved domain decomposition overcoming some limitations imposed by the static approach. This contribution proposes a solution that implements dynamic mesh repartitioning based on runtime measurements. The main goal is to reduce the imbalance among the processes in order to improve the overall parallel efficiency of the execution. The so called “imbalance factor” is a representative quantity to evaluate the partitioning quality and is defined as follows:

$$\text{imbalance factor} = \frac{\text{local time}}{\text{average time among processes}}$$

This factor is equal to one for a perfectly balanced subdivision, where literally equal amounts of work are assigned to the processes. In the worst case, the imbalance factor is equal to the number of processes. Then all the work is assigned to a single process, where all other processes remain idle. An important aspect to consider is that, in principle, the process with the highest workload is never idle and, hence, dictates the progress of the overall coupled parallel simulation. For this reason, having a mesh subdivision in which only one process has a high positive imbalance implies that all other processes remain idle for at least part of the time. On the contrary, a process with an imbalance lower than 1 is less problematic because it means that only that process is idle, even if some computational time is wasted. In accordance with best practice for real applications, the maximum imbalance factor over all processes should not exceed 1.05.

In mesh-based CFD solvers, data can be related to elements, faces, edges or nodes. During execution, when data has to be known at different entities, or derived quantities have to be calculated, e.g. variable gradients, it is necessary to cycle over faces, elements or nodes to retrieve them. In CODA, which is a cell-based code, three different cycles are performed: elements, inner faces, and boundary faces. Since most of the computational time is spent in the loops over elements and faces (inner and boundary), in this work the time spent over element and face loops has been chosen as the representative quantity to base the partitioning weights on. Direct measurement for each face and element is not feasible because each runtime measurement introduces an overhead. For this reason, timings are recorded per loop type in CODA: elements, inner faces, and boundary faces. In order to satisfy the requirement of the partitioning algorithm, the measured time is attributed to the elements proportionally to the number of integration points. Element timings are directly distributed to elements, internal face timings are split to elements sharing the face, and boundary face timings are assigned to the unique element owning the boundary face.

Some external libraries, in particular ParMetis and GeMPa, support only integer-valued partitioning weights. In the present code, floating-point timing values are converted to integer values using the following relation:

$$w_i = \text{integer}\left(100\frac{t_i}{\text{avg}(t_i)}\right)$$

where $w_i$ and $t_i$ are the partitioning weight and the runtime measurement, computed from loop timings, of the $i^{th}$ element, respectively. A scale factor equal to 100 is used to achieve a sensitivity of around 1% after the integer conversion. Once the partitioning weights have been computed they are used in the next repartitioning iteration.

## 3 PARTITIONING METHODS

Three different partitioning algorithms have been considered: recursive coordinate bisection (RCB), graph-based using ParMetis, and Hilbert space filling curve (HSFC) as provided by the GeMPa library. The current RCB implementation in FSDM only supports node-based, but not cell-based partitioning. The mesh subdivision resulting from the RCB method is thus not directly used for CODA, which features a cell-centered finite-volume discretization. RCB partitioning is nevertheless employed to balance the initial mesh-data distribution obtained after loading the mesh, to avoid memory bottleneck in the graph-based partitioning. In fact, the graph extraction (as implemented in FSDM) has been observed to suffer from memory-overflow error when applied to the initial distribution (without RCB pre-partitioning).

All performance measurements were executed on the DLR supercomputer CARA. It is composed of 2300 nodes with Infiniband HDR interconnect. Each node consists of two 32-core AMD EPYC 7601 @2.20GHz CPUs. The code was compiled using GCC 8.2.0 with flags `-O3 -march=native`. In Table 1, the time spent during the partitioning procedure is reported, subdivided in: “compute”, the time necessary to compute the new process for each mesh entity; “redistribute”, the time needed to redistribute the mesh among the processes; “graph”, which is present only for ParMetis, is the duration necessary to extract the graph from the mesh. The analysis ranges from 64 to 8192 MPI processes, which corresponds to 1 to 128 compute nodes

of CARA. In Figure 1, the times spent for the different approaches are put into relation using the "speed-up factor" notation.

**Table 1**: Wall-clock time comparison among partitioning methods.

| **Processes** | | **64** | **128** | **256** | **512** | **1024** | **2048** | **4096** | **8192** |
|---|---|---|---|---|---|---|---|---|---|
| **Switches** | | **1** | **1** | **1** | **1** | **1** | **1** | **3** | **5** |
| **RCB** | **Compute** $[s]$ | 9.24 | 6.10 | 4.34 | 3.69 | 4.95 | 10.1 | 34.4 | - |
| | **Redistribute** $[s]$ | 24.1 | 18.7 | 9.73 | 4.66 | 3.17 | 2.11 | 4.47 | - |
| **Parmetis** | **Graph** $[s]$ | 92.4 | 53.0 | 45.5 | 33.9 | 28.8 | 35.7 | - | - |
| | **Compute** $[s]$ | 7.34 | 4.79 | 3.13 | 3.09 | 6.38 | 21.0 | - | - |
| | **Redistribute** $[s]$ | 93.5 | 53.2 | 33.6 | 19.3 | 8.61 | 5.11 | - | - |
| **GeMPa** | **Compute** $[s]$ | 0.503 | 0.323 | 0.215 | 0.185 | 0.141 | 0.129 | 0.301 | 0.410 |
| | **Redistribute** $[s]$ | 100.8 | 67.8 | 41.9 | 19.7 | 11.2 | 7.11 | 5.42 | 4.42 |

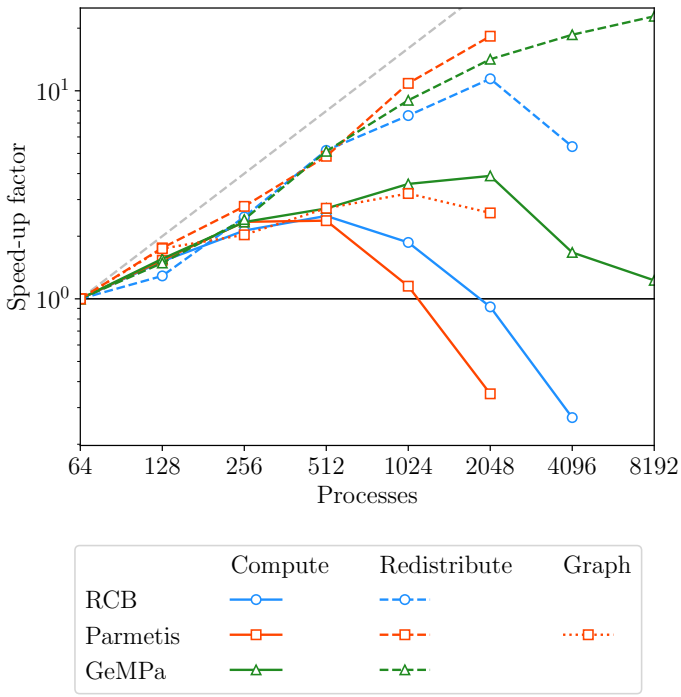


**Figure 1**: Speed-up factor comparison among partitioning methods.

For ParMetis, around half of the overall partitioning time is actually spent in extracting the

graph from the computational mesh. In addition, when increasing the number of MPI processes above 512, the time to compute the new distribution increases. This seems mostly related to MPI collective operations. The same behavior is also present in the RCB method which is implemented using frequent AllToAll communication. On the contrary, the redistribution algorithm, which extensively takes advantage of point-to-point communication, reaches a better scalability. Another aspect that also impacts partitioning performance is the number of network switches. The interconnect of the CARA supercomputer is based on a fat-tree layout. In 64- and 128-node executions, a two-level switch structure is exploited, resulting in higher latency. The degradation of performance is visible for both, GeMPa and RCB. Focusing only on the redistribution algorithm, the RCB method differs from the other two since the partitioning is performed on nodes and relies on a global numbering. ParMetis and GeMPa, instead, compute the subdivision based on cells and use a local numbering. Apparently better redistribution performance is achieved with ParMetis. This seems due to the initial mesh distribution, which has already been reordered with RCB, cf. above. In GeMPa, the unordered mesh distribution does not affect the partitioning because each mesh entity is mapped into the one-dimensional space, so the RCB reordering is avoided. Comparing GeMPa partitioning to RCB and ParMetis, a 2.24x reduction in time is observed when using 64 processes and up to 10.23x reduction for 2048 processes.

## 4 ONERA M6 WING

The proposed partitioning approaches are tested on a well-known test case of aeronautical interest, a transonic flow around the ONERA M6 wing. The free-stream Reynolds number is $\mathrm{Re}_\infty = 14.6 \cdot 10^6$, while the Mach number is $Ma_\infty = 0.84$. The angle of attack is $\alpha = 3.06°$. In Figure 2, the pressure coefficient $C_p$ distribution and the skin friction lines are represented, highlighting the characteristic lambda-shaped double shock on the upper side of the wing.

The compressible Reynolds averaged Navier-Stokes (RANS) equations are discretized using CODA's cell-based finite volume method. The system closure is achieved by the negative formulation of Spalart-Allmaras one-equation turbulence model (SA-neg) [19]. Convective fluxes are computed using a 2nd-order scheme with an approximate Riemann solver of Roe type. A pseudo time-marching approach is applied to drive the equation system to a steady solution using a linearized implicit Euler scheme with a locally constant CFL number equal to 5. The linear systems are solved using the Jacobi method with an element-local LU preconditioner provided by Spliss [20]. A hybrid mesh composed of 55.1M hexahedra and 11M prisms is used [21].

Focusing on mesh-partitioning performance, the test case is executed on different configurations ranging from 512 to 8192 pure MPI processes, which correspond to 8 to 128 compute nodes. The 2nd-level shared-memory parallelization present in CODA is not exploited in these computations, even though CODA's parallel scalability is significantly extended by this feature. Here, the focus is on FS(DM) partitioning of the mesh on process level, which is adopted by CODA. In Figure 3, a visual representation of the imbalance is given for an execution with 1024 MPI processes combining GeMPa and Parmetis methods with static and dynamic approaches. The multi-constraint capability of ParMetis grants a satisfactory imbalance distribution using only the geometric properties of the mesh, namely the static approach. When the dynamic approach and runtime measurements are applied with ParMetis, the imbalance does not show

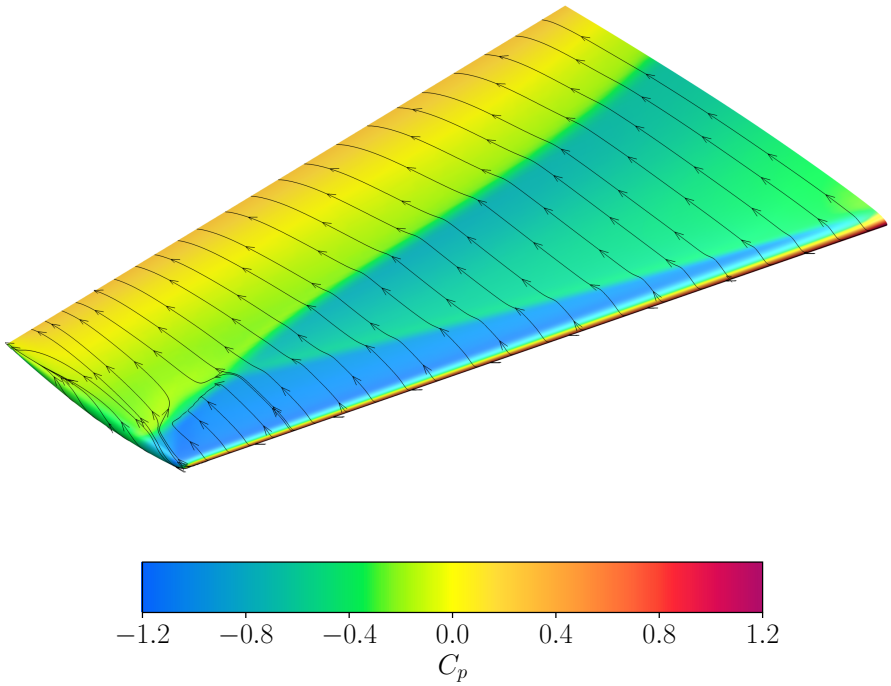


**Figure 2**: Pressure coefficient $C_p$ distribution and skin friction lines over the ONERA M6 wing.

an improvement and remains in the $0.95 - 1.05$ range, just as in the static approach. The GeMPa limitation of balancing only one constraint, and the geometric nature of the library, lead to unacceptable imbalances in the static approach with 1.3 imbalance peaks. This is due to high concentrations of boundary faces (which are more expensive than inner ones) for only a few processes. The dynamic approach, however, combined with the timings-based weights, allows an improved imbalance in the $0.98 - 1.02$ range. The actual two alternatives here are static ParMetis and dynamic GeMPa. These are further analyzed in the following. (In cases for which a higher imbalance is expected, however, also dynamic graph-based partitioning can be a reasonable alternative.) A quantitative analysis is shown in Figure 4, where the maximum and minimum imbalance factors are reported as a function of the number of used processes. A direct comparison with ParMetis is only possible up to 2048 cores, the largest set of processes that could be run without execution failure in our particular setting. GeMPa is able to achieve a maximum imbalance of about 2% up to 2048 processes. Moving to 4096 and 8192 processes, a performance degradation is observed, reaching a 6% deviation from the average, although still acceptable for a simulation. Regarding ParMetis, instead, a noticeably higher imbalance is observed already for low process counts. As the number of processes is increased, the imbalance worsens further.

In Figure 5, the ratio $\psi$ between idle time and the overall maximum time, namely the computational time if all processes did start and also finish at the same points in time, respectively,

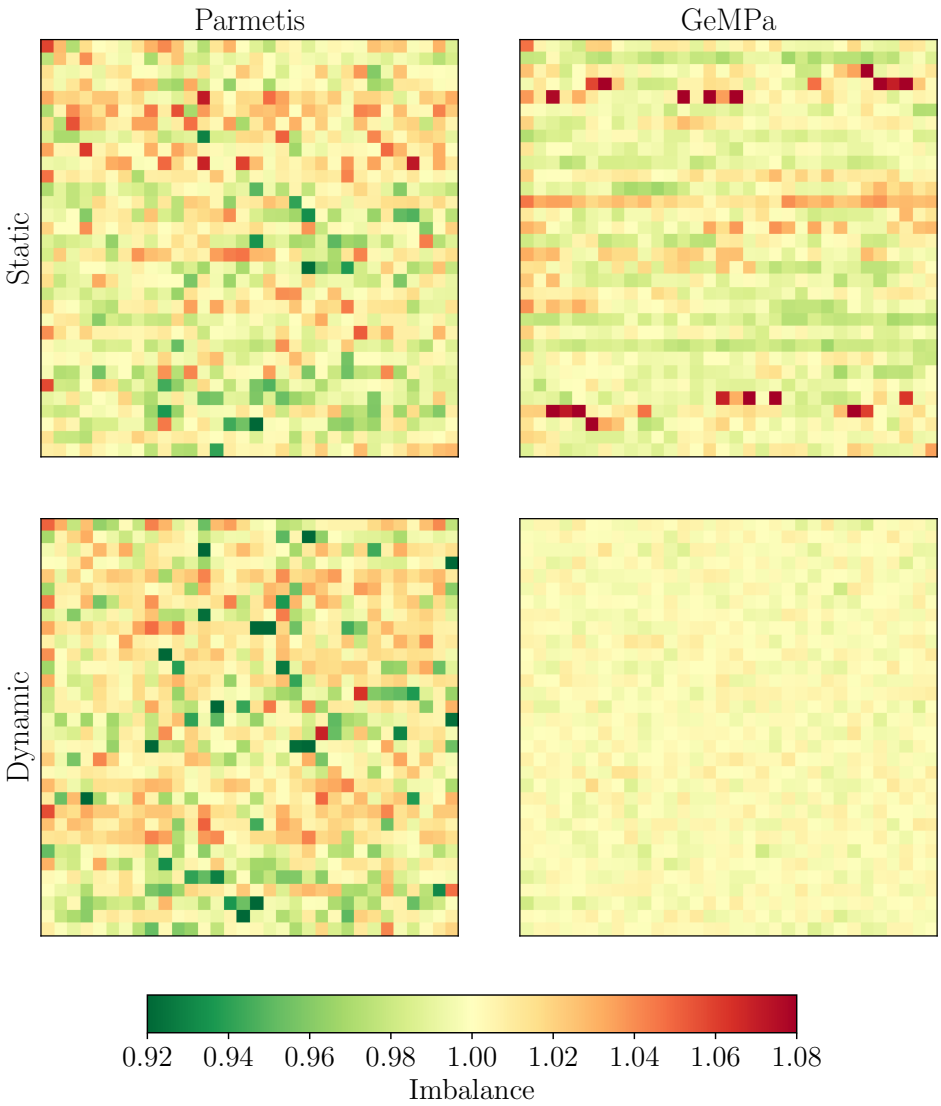


**Figure 3**: Imbalance comparison for the proposed approaches using 1024 MPI processes.

is reported. In detail, $\psi$ is computed as

$$\psi = \frac{\text{idle time}}{\text{overall maximum time}} = \frac{\sum_{i=1}^{N_{\text{proc}}} (\max t_k - t_i)}{N_{\text{proc}} \cdot \max t_k}$$

with $t_i$ the time spent in loops and $N_{\text{proc}}$ the number of MPI processes in the execution. In the dynamic GeMPa approach, 10 repartitioning iterations are performed to investigate the subdivision quality. Considering only the first partitioning iteration, using only the geometric constraint, a direct comparison between GeMPa and Parmetis is possible. The graph-based method clearly outperforms geometric one. When runtime measurements are adopted as par-

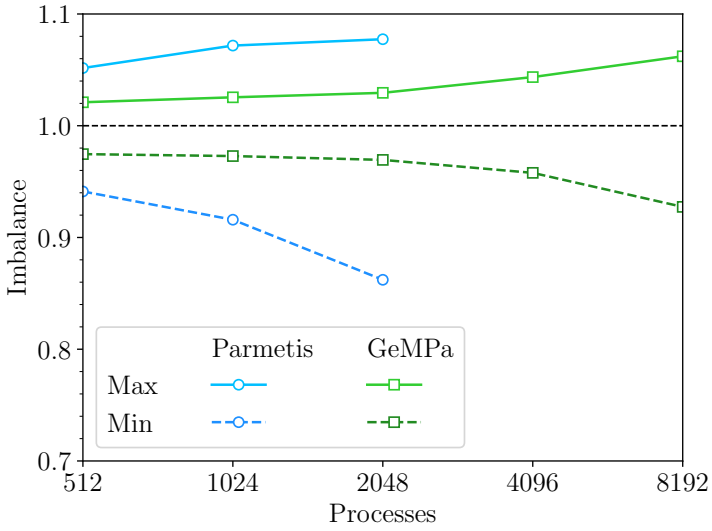


**Figure 4**: Maximum and minimum imbalance comparison for the proposed approaches.

titioning weights, however, idle times are reduced. After 3 to 5 iterations, depending on the number of processes, continuing to further refine the partitioning (utilizing additional runtime measurements) no longer improves the partition quality.

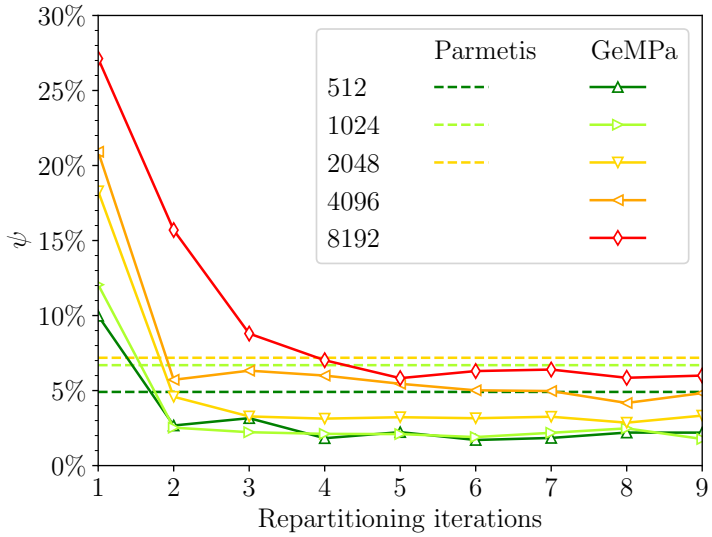


**Figure 5**: Comparison of idle time for the proposed approaches.

A final analysis is presented of the wall-clock time spent in computing one iteration of the linearized Euler method for the ONERA M6 case. The number of iterations to solve the linear system is maintained constant, as well as the cell-local CFL number, to ensure the same amount of work. The results are summarized in Table 2 and in Figure 6. The dynamic GeMPa algorithm

**Table 2**: Wall-clock time of one linearized Euler method iteration for the proposed approaches.

| **Processes** | **512** | **1024** | **2048** | **4096** | **8192** |
|---|---|---|---|---|---|
| **Parmetis** $[s]$ | 11.720 | 6.008 | 3.091 | - | - |
| **GeMPa** $[s]$ | 11.526 | 5.797 | 2.991 | 1.538 | 0.810 |
| **Reduction** | 1.66% | 3.51% | 3.24% | - | - |

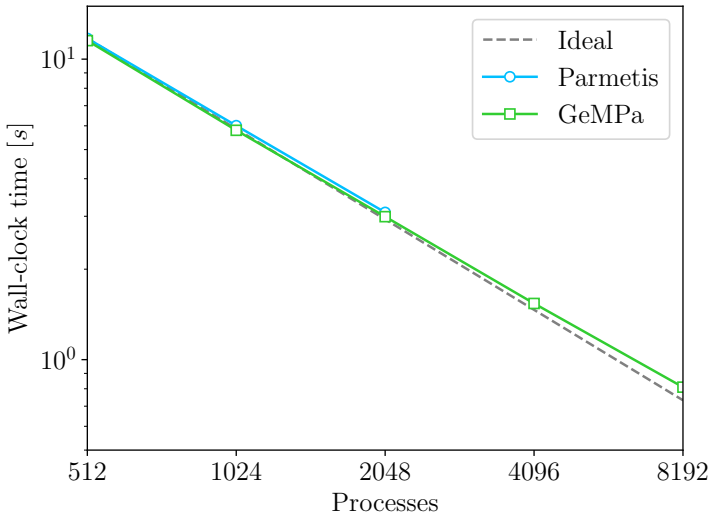


**Figure 6**: Wall-clock time of one linearized Euler method iteration for the proposed approaches.

allows a relative wall-cock time reduction of around 3% when using 1024 and 2048 processes. The improvement is in accordance with the reduction of imbalance from 5% of static ParMetis to 2% of dynamic GeMPa. Another advantage of the proposed approach is the possibility to use a larger number of processes with FSDM without incurring in memory overflow errors. (Using a larger set of cores is achievable also when using ParMetis, namely when exploiting the CODA hybrid parallelism.) Finally, the solver's wall-clock time per integration step decreases steadily and almost linearly up to 8192 processes, indicating very good parallel scalability in this regime.

## 5 CONCLUSIONS

A dynamic mesh-partitioning algorithm based on runtime measurements has been presented. The use of timings as partitioning weights demonstrated an improved balance among processes. Furthermore, graph partitioning libraries require a considerable amount of computational resources when the number of parts is above a certain threshold. Instead, Hilbert space-filling curve partitioning has been shown as an efficient alternative with a lower memory requirement. Finally, the reduced computational time to compute the partitioning of HSFC methods is particularly suitable for cases that require frequent repartitioning.

The runtime measurements of the current implementation are limited to the loops over mesh entities in CODA. A future development is the possibility of using timings provided by separate libraries that are currently not accounted for. One relevant example is the influence of the linear system solver Spliss that can become predominant for implicit time integration. Further investigations are also required to better understand how the additional shared-memory level (which was not considered here) affects the timings and the partitioning. Finally, additional test cases characterized by more challenging imbalances, as well as actually dynamic setups, are to be studied next.

## ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the NextSim project, grant agreement No. 956104.

## REFERENCES

[1] J. P. Slotnick, A. Khodadoust, J. J. Alonso, D. L. Darmofal, W. Gropp, E. A. Lurie, and D. J. Mavriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. 2014.

[2] P. Mohanamuraly and G. Staffelbach. Hardware locality-aware partitioning and dynamic load-balancing of unstructured meshes for large-scale scientific applications. *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2020.

[3] I. Huismann, L. Reimer, S. Strobl, J. Eichstädt, R. Tschüter, A. Rempke, and G. Einarsson. Accelerating the FlowSimulator: Profiling and scalability analysis of an industrial-grade CFD-CSM toolchain. In *IX. International Conference on Coupled Problems in Science and Engineering*, volume 9, 2021.

[4] M. Bernardini, D. Modesti, F. Salvadore, and S. Pirozzoli. Streams: A high-fidelity accelerated solver for direct numerical simulation of compressible turbulent flows. *Computer Physics Communications*, 263:107906, 2021.

[5] R. Borrell, D. Dosimont, M. Garcia-Gasulla, G. Houzeaux, O. Lehmkuhl, V. Mehta, H. Owen, M. Vázquez, and G. Oyarzun. Heterogeneous CPU/GPU co-execution of CFD simulations on the POWER9 architecture: Application to airplane aerodynamics. *Future Generation Computer Systems*, 107:31–48, 2020.

[6] M. Karp, A. Podobas, T. Kenter, N. Jansson, C. Plessl, P. Schlatter, and S. Markidis. A high-fidelity flow solver for unstructured meshes on field-programmable gate arrays: Design, evaluation, and future challenges. In *International Conference on High Performance Computing in Asia-Pacific Region*, page 125–136, 2022.

[7] L. Zhang, G. Zhang, Y. Liu, and H. Pan. Mesh partitioning algorithm based on parallel finite element analysis and its actualization. *Mathematical Problems in Engineering*, 2013.

[8] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[9] G. Karypis, K. Schloegel, and V. Kumar. Parmetis parallel graph partitioning and sparse matrix ordering library. 1997.

[10] K. Devine, E. Boman, R. Heapby, B. Hendrickson, and C. Vaughan. Zoltan data management service for parallel dynamic applications. *Computing in Science and Engineering*, 4:90–97, 03 2002.

[11] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6):318–331, 2008.

[12] G. Baldan, T. Bellosta, and A. Guardone. Efficient parallel algorithms for coupled fluid-particle simulation. In *7th edition of the International Conference on Particle-based Methods*, 2021.

[13] G. Baldan, T. Bellosta, and A. Guardone. A scalable lagrangian particle tracking method. In *9th edition of the International Conference on Computational Methods for Coupled Problems in Science and Engineering*, 2021.

[14] J. Luitjens, M. Berzins, and T. Henderson. Parallel Space-Filling Curve Generation through sorting: Research articles. *Concurrency and Computation: Practice & Experience*, 19(10):1387–1402, 2007.

[15] M. Meinel and G. Einarsson. The FlowSimulator framework for massively parallel CFD applications. 2010.

[16] T. Leicht, J. Jägersküpper, D. Vollmer, A. Schwöppe, R. Hartmann, J. Fiedler, and T. Schlauch. DLR-Project Digital-X - Next Generation CFD Solver 'Flucs'. In *Deutscher Luft- und Raumfahrtkongress 2016*, 2016.

[17] R. Borrell, G. Oyarzun, D. Dosimont, and G. Houzeaux. Parallel SFC-based mesh partitioning and load balancing. *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 72–78, 2019.

[18] R. Borrell, J.C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez, and G. Houzeaux. Parallel mesh partitioning based on space filling curves. *Computers & Fluids*, 173:264–272, 2018.

[19] S. R. Allmaras, F. T. Johnson, and P. R. Spalart. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. 2011.

[20] O. Krzikalla, A. Rempke, A. Bleh, M. Wagner, and T. Gerhold. Spliss: A Sparse Linear System Solver for transparent integration of merging HPC Technologies into CFD solvers and applications. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, 2021.

[21] H. Nishikawa and B. Diskin. Customized grid generation codes for benchmark three-dimensional flows. In *2018 AIAA Aerospace Sciences Meeting*.