XI International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2025
A. Larese, S. Perotto and G. Rozza (Eds)

ACCELERATION OF EXTREME SCALE FLOW SIMULATIONS THROUGH HIERARCHICAL MESH PARTITIONING

JONATHAN A. FENSKE*, MARCO CRISTOFARO*, ARNE REMPKE*

*Institute of Software Methods for Product Virtualization
German Aerospace Center (DLR)
Nöthnitzer Str. 46b, 01187 Dresden, Germany
e-mail: Jonathan.Fenske@dlr.de, web page: http://www.dlr.de/sp

Key words: Coupled Problems, Mesh Deformation, Mesh Partitioning, High-Performance Computing, CFD, CSM

Abstract. This paper investigates how extreme scale computational fluid dynamics (CFD) and computational solid mechanics (CSM) simulations can be accelerated through hierarchical graph partitioning, i.e., partitioning a graph representing the unstructured mesh used for these numerical simulations on multiple hierarchy levels with respect to the architecture of high-performance computing (HPC) systems. Hierarchical partitioning can reduce communication between higher hardware hierarchy levels and, thus, reduce the communication time within simulations. This effect is expected to be greater with bigger meshes and a higher number of compute nodes used for these simulations. Consequently, applying hierarchical partitioning to industrial scale-resolving simulations can speed up and enable larger simulations of CFD, CSM, and coupled fluid-structure interaction (FSI) simulations than before. We propose an implementation of hierarchical partitioning and apply it to a very large test case with more than 1 billion cells.

1 INTRODUCTION

Numerical simulations of coupled computational fluid dynamics (CFD) and computational structural mechanics (CSM), i.e. fluid-structure interaction (FSI), simulations are an essential part of aircraft design development.

Bigger high-performance computing (HPC) systems enable simulations with higher resolution and, in some cases, even scale- resolving simulations providing much more meaningful results. This can speed up the development process of new aircraft significantly and can save costs. Due to climate change, such new developments are also necessary to reduce carbon emissions. The efficient usage of these HPC systems requires a good parallelization of the simulation software and an optimized partitioning of the mesh data, typically using graph partitioning libraries such as `Zoltan` [10] or `ParMETIS` [11].

However, for large unstructured meshes or a higher number of cores, current partitioning software may take too long to partition and distribute the data. The resulting domain decomposition is often not optimized for the system architecture either. In some cases, it is not even possible to compute a mesh partitioning.

One way to improve on this problem is employing hierarchical partitioning [8], i.e., partitioning the data on multiple hierarchy levels with respect to the HPC system architecture. Such an approach uses less resources on each hierarchy level and considers the inter node communication time. This enables numerical flow simulations utilizing more resources efficiently and, hence, also more precise simulations than previously possible.

First, we give an overview of the simulation software within DLR's `FlowSimulator` framework [14] used in the computational studies that are presented in this paper. Then, we present our implementation of a hierarchical partitioner and its effect on the runtimes of CFD and CSM simulations within `FlowSimulator` when compared to using `ParMETIS`' graph partitioning method that is already provided via an interface within `FlowSimulator`. For this demonstration, we use meshes of more than a billion cells and run the simulations on more than $65\,000$ cores. In the end, we draw our conclusions and give a short outlook.

2 FLOWSIMULATOR FRAMEWORK

The `FlowSimulator` framework is jointly developed by DLR, ONERA, and Airbus and aims to enable a common framework for data management in multidisciplinary simulations related to aircraft design. This enables simulation software of different disciplines to exchange data more easily and use all the different functionalities provided by `FlowSimulator`.

At the heart of `FlowSimulator` lies the `FlowSimulator DataManager` (`FSDM`). `FSDM` mainly provides functionalities for mesh import, export, partitioning, and modification. For the partitioning of data, `FSDM` currently includes its own implementation of the Recursive Coordinate Bisection (RCB) method [6] and an interface for the third-party graph partitioning library `ParMETIS`. Additionally, an interface for `Zoltan`, another third-party graph partitioning library, is available via a plugin.

Typically, `ParMETIS` and `Zoltan` are used to compute the partitioning that is later used for the simulations. Meanwhile the RCB partitioner is used as a fast prepartitioner to speed up the following graph partitioning process.

To run the CFD simulations, we used CFD Software by ONERA, DLR, and Airbus (`CODA`) [13], a next generation CFD solver employing finite volume and discontinuous Galerkin methods. It is developed with a focus on scalability, also supporting a hybrid MPI + OpenMP approach to enable efficient large scale simulations. Internally, it uses the sparse linear systems solver `Spliss` [12] for solving the linear equations.

Accurate aircraft simulations also need to account for the deformations, e.g. of the the wings, caused by the airflow and other outside forces. For this, `FlowSimulator` provides the `FSMeshDeformation` plugin, which, among others, implements the elasticity analogy method [15, 16]. `Spliss` or `PETSc` [5] can be used to solve the linear system of equations.

3 HIERARCHICAL PARTITIONING

Hierarchical partitioning is recursive hardware aware partitioning of data on multiple hierarchy levels. The main advantage of hierarchical partitioning is a minimization of communication between processes in different domains on higher hierarchy levels, e.g. between processes on different compute nodes on HPC systems. This potentially reduces the runtime of simulations since communication on higher hierarchy levels is more costly than on lower hierarchy levels. There are two approaches to do this. One works in a top-down and the other in a bottom-up

direction.

For the top-down approach, the data is first partitioned on the highest hierarchy level which, on HPC systems, typically consists of the compute nodes. Then, the data is recursively partitioned within lower hierarchy levels such as sockets, cores, etc. However, the lowest hierarchy level are the Message Passing Interface (MPI) processes which can consist of multiple cores. This is for example the case if a hybrid approach of MPI and OpenMP is used.

For the bottom-up approach, the data is first partitioned on the lowest hierarchy level, i.e. among all processes. Then, the partitions are recursively redistributed in such a way that connections between these partitions are minimized on higher hierarchy levels.

An advantage of the top-down approach over the bottom-up approach is that the partitioning is computed for fewer partitions at once, increasing the performance and quality of the partitioning. However, the load imbalance will increases with each hierachy level with the top-down approach while it stays constant with the bottom-up approach because the partitions are not changed but only redistributed.

Both approaches were implemented as 2-level hierarchical partitioners in a plugin, named `FSHierarchicalPartitioner`, of `FlowSimulator`. The two considered hierarchy levels are compute nodes and MPI processes on systems where these processes are pinned to specific cores.

In the implementation of the top-down approach, first, a graph representing the mesh is extracted. In this graph, every cell is represented by a graph node and the connections to neighbouring cells are represented by graph edges between both graph nodes. The graph nodes and edges can be weighted to better represent different cell types. Graph partitioners will decompose the graph into a certain number of domains with similar weight while trying to minimize the edge cut. Then, on each compute node, the local graph data is gathered on one process per compute node. These processes then proceed to compute the partitioning among the compute nodes and distribute this data to the other processes within their own compute nodes. This way, the problem that we see in Section 4.3.2 of too many processes simultaneously being involved in these computations and subsequently causing out-of-memory errors (OOM) can be avoided. After the mesh data was redistributed according to the computed partitioning, graphs representing the local mesh are extracted within each compute node. This graph is then used to compute a partitioning among the processes within every compute node.

In the implementation of the bottom-up approach, again, a graph representing the whole mesh is extracted. Like in the top-down approach, next, the local graph data is gathered on one process per compute node to avoid OOM errors for large numbers of processes during the computation of the partitioning. Then, the mesh is partitioned among all processes. In the second hierarchy level, a graph representing the connectivity of the processes is extracted. This is done by having a graph node for every process that is connected to all other graph nodes. In order to represent the number of connections between the processes, each graph edge is weighted with the number of cells having a neighbouring cell on the other respective process. The graph is then partitioned among the compute nodes. This way, partitions with more connections to each other will be on processes on the same compute node.

It is instead also possible to extract graphs representing the mesh nodes and not the mesh cells if so desired. Further, this plugin, just like other graph partitioners within `FlowSimulator`, provides the functionality of keeping cells stacks close to boundary layers together in the same partition. However, this functionality was not used in the computational studies conducted for

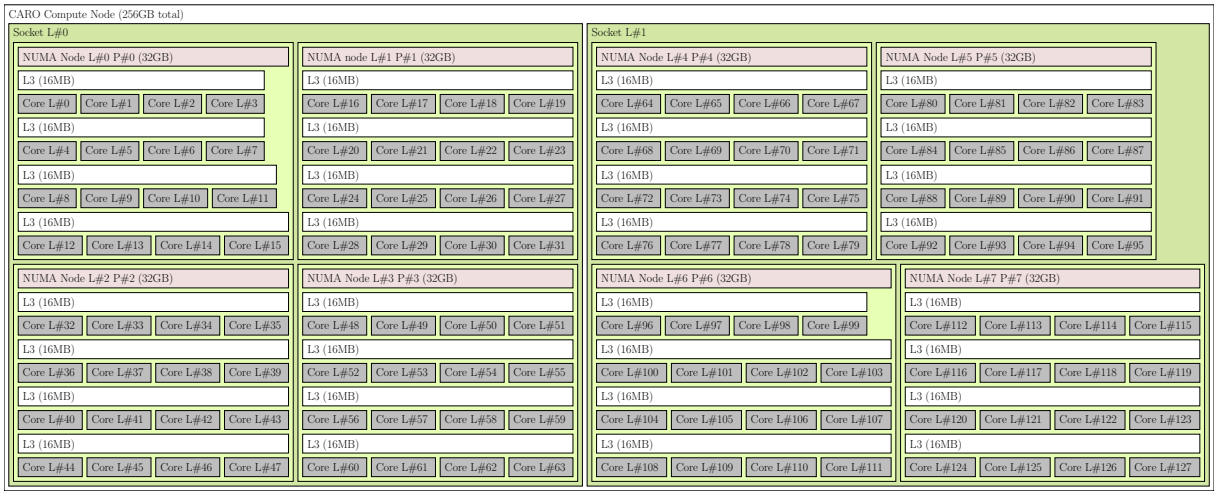

Figure 1: Image represeting the topology of a CARO compute node including sockets, NUMA domains, L3-caches, and cores (created using `lstopo` from the framework `hwloc` [7]).

this paper.

One important improvement when compared to the partitioners already integrated into `FlowSimulator` is optimizing the communication pattern in the graph extraction. When finding neighbouring cells on other processes, an all-to-all communication pattern was used to exchange the information of cells for which we want to find neighbours on other processes. This has been replaced by a more hierarchical communication pattern where each process sends its relevant information to one process on each compute node. This process then sends this information to all other processes within its own compute node. The impact of this is demonstrated in Section 4.3.1.

To compute the partitionings within the hierarchy levels, `FSHierarchicalPartitioner` still uses third-party graph partitioning libraries. Currently, `ParMETIS` and `Zoltan` are supported.

4 COMPUTATIONAL STUDIES

In this section, we will present the computational studies we conducted in order to investigate the actual impact of hierarchical partitioning on our simulations. For this, we made strong scaling benchmarks of `CODA` and `FSMeshDeformation` runtimes after the mesh was first prepartitioned by `FSDM`'s implementation of the RCB method and then repartitioned by either just `ParMETIS` or the top-down or bottom-up approach in `FSHierarchicalPartitioner`, internally also using `ParMETIS` and `METIS`. Specifically, all partitioners use the `ParMETIS_V3_PartKway` or `METIS_PartGraphKway` routines.

4.1 Computational setup

The tests were conducted on DLR's HPC system CARO [4]. CARO has 1364 compute nodes with 2 AMD EPYC 7702 [2] processors each. Every processor has 64 cores, 8 Non-Uniform Memory Access (NUMA) domains, 16 dies with a shared L3 cache, and a base clock frequency of $2\,\mathrm{GHz}$. The topology of a CARO compute node is shown in Figure 1.

The versions of `CODA`, `FSDM`, `FSHierarchicalPartitioner`, and `FSMeshDefor-`

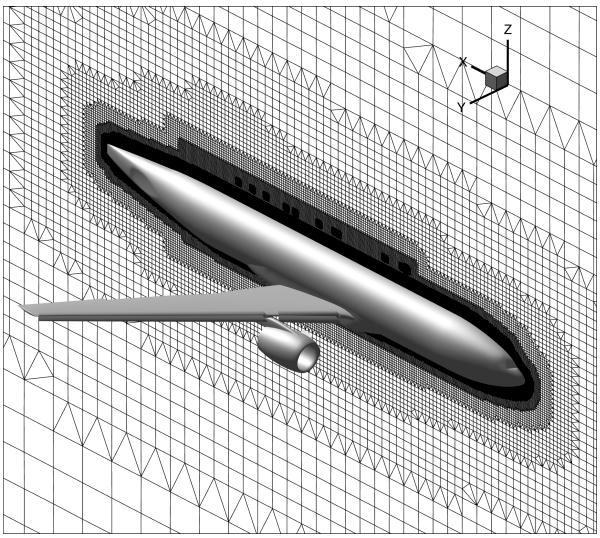

Figure 2: Image of the CRM-HL mesh used for the computational studies. The cells representing the surface of the airplane are higher resolved than other cells.

`mation` used here are all from February 2025 and use 64 bit integers.

4.2 Test case

The test case used for the computational studies is grid D of the mesh family 3.R.01 with $\mathrm{Re} = 3 \cdot 10^7$ from the 5th AIAA CFD High Lift Prediction Workshop (HLPW-5) [1]. This mesh is based on the high lift configuration of NASA's Common Research Model (CRM-HL) [3]. It has 1.23 billion cells and 973 million nodes and is displayed in Figure 2.

4.3 Results

Now, we can discuss the results obtained by running `CODA` and `FSMeshDeformation` simulations after partitioning the mesh with either `FSHierarchicalPartitioner` or just `ParMETIS`.

4.3.1 Results with CODA

For the `CODA` tests, we used the linearized implicit Euler time integration method with a linear Jacobi solver with LU preconditioning from the `Spliss` and 100 time iterations.

Since `CODA` supports hybrid MPI + OpenMP, all test runs were done with 4 threads per process. We did this without using simultaneous multithreading and thus also used 4 cores per process.

It was necessary to use at least 8192 cores to not run into out-of-memory (OOM) erros. The maximum number of used cores is $65\,536$ because this was the user limit on CARO.

As can be seen in Figure 3, the `CODA` runtimes scale quite well for all different partiong approaches. However, in the end, `FSHierarchicalPartitioner` scales better than `ParMETIS` with the top-down approach leading to the best result. For the top-down approach,

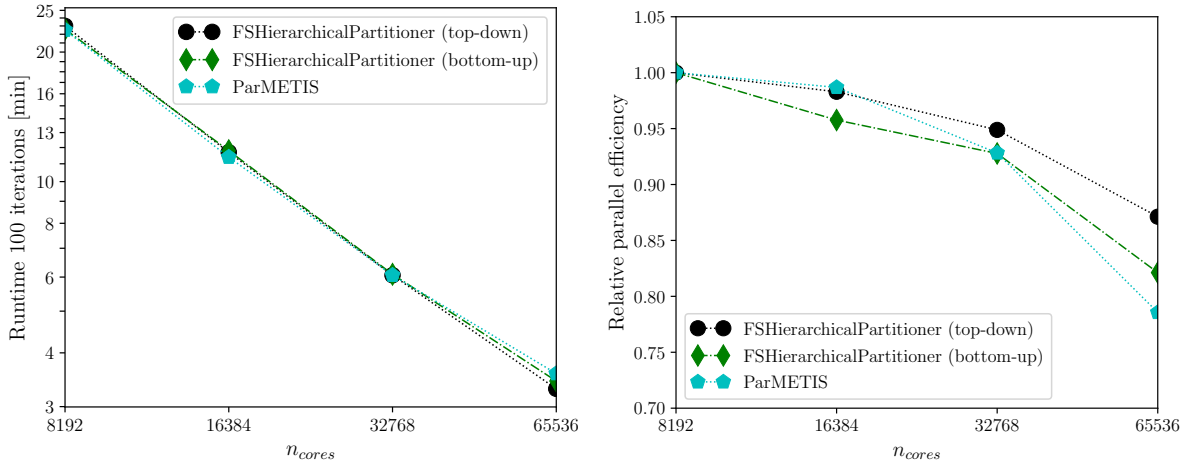

Figure 3: Strong scaling benchmarks of `CODA` runtimes and relative parallel efficiency after being partitioned by the respective partitioning scheme.

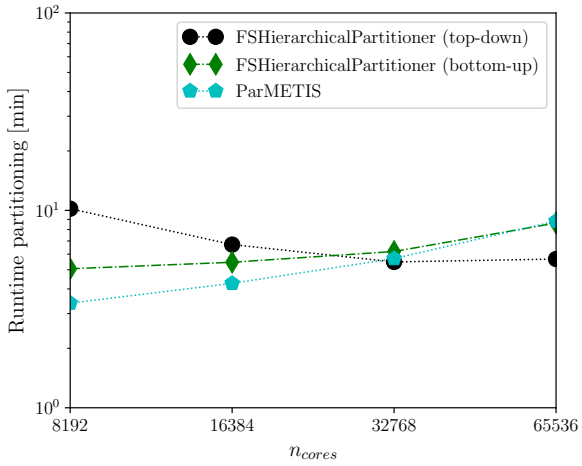

Figure 4: Runtimes of the partitioning processes in the `CODA` benchmarks, i.e. with 4 threads per process.

the relative parallel efficiency, here defined as

$$E_{\text{rel}} = \frac{T_{n_0}}{\frac{n_k}{n_0} \cdot T_{n_k}},$$

where T_{n_k} is the runtime with n_k cores used and n_0 is the smallest number of used cores in a data point, is still at 0.87 for the top-down approach in `FSHierarchicalPartitioner` with $65\,536$ cores or 512 compute nodes, whereas it is only 0.82 for the bottom-up approach and 0.78 for just `ParMETIS` in this case. An ideal scaling would provide a relative parallel efficiency of 1.

Furthermore, we can see a comparison of the runtimes of the partitioning processes of these benchmarks in Figure 4. Besides computing the partitioning, this also includes the graph extraction and redistribution of the mesh to the different ranks. It should be noted that `FSHierarchicalPartitioner` and `ParMETIS` do not support multithreading. Thus, only 1 thread per process was active for the partitioning phase despite having 4 threads per process available.

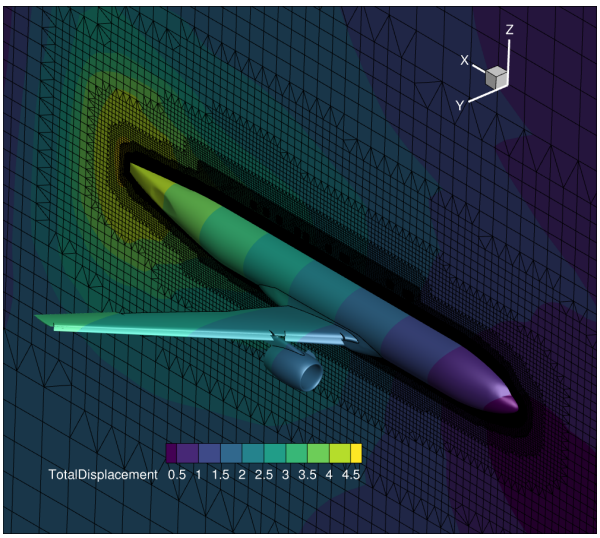

Figure 5: The CRM-HL mesh with an applied deformation of $0.1°$ rotation around the y-axis. The color scale represents the local values of the resulting total displacement.

It can be observed that the top-down approach of `FSHierarchicalPartitioner` takes the longest time when fewer cores are used but gradually gets faster with more cores except in the end because it is slightly faster with $32\,768$ than with $65\,536$ cores. With fewer cores, it is slower because, in the first hierarchy level, a large number of cells is moved to and, in the second hierarchy level, from one process per compute node. However, with more cores, it is the fastest approach because in the second hierarchy level, the number of cells for which a partitioning is computed is continously decreasing. This accelerates the time needed to compute the partitioning as well as to redistritbute the cells. For $32\,768$ cores or more, it is even the fastest of the compared approaches here.

Both other approaches take increasingly more time for the partitioning phase. This is mainly due to the graph extraction taking more time for more processes. With fewer cores, `ParMETIS` is still faster than `FSHierarchicalPartitioner`'s bottom-up approach because they both compute same partitioning at some point but `FSHierarchicalPartitioner`'s bottom-up approach computes an additional partitioning on the second hierarchy level. Nonetheless, `ParMETIS` is slightly slower than `FSHierarchicalPartitioner`'s bottom-up approach with $65\,536$ due to improvements in the graph extraction in `FSHierarchicalPartitioner`.

4.3.2 Results with FSMeshDeformation

For the `FSMeshDeformation` tests, we computed and applied a $0.1°$ rotation around the y-axis to the aircraft in the mesh using linear elasticity. The result is illustrated in Figure 5. To solve the linear system of equations, we used `Spliss`' and `PETSc`'s implementations of the biconjugate gradient stabilized (BiCGStab) method, in `PETSc` preconditioned with the successive over-relaxation (SOR) method, and, in `Spliss` preconditioned by an algebraic multigrid (AMG) preconditioner with Gauß-Seidel smoothing on every level. Both solvers were set to an error tolerance of 10^{-9}. Since `PETSc` does not support a hybrid MPI + OpenMP approaches, tests with `PETSc` were run with pure MPI while the `Spliss` tests were run with 4 threads per

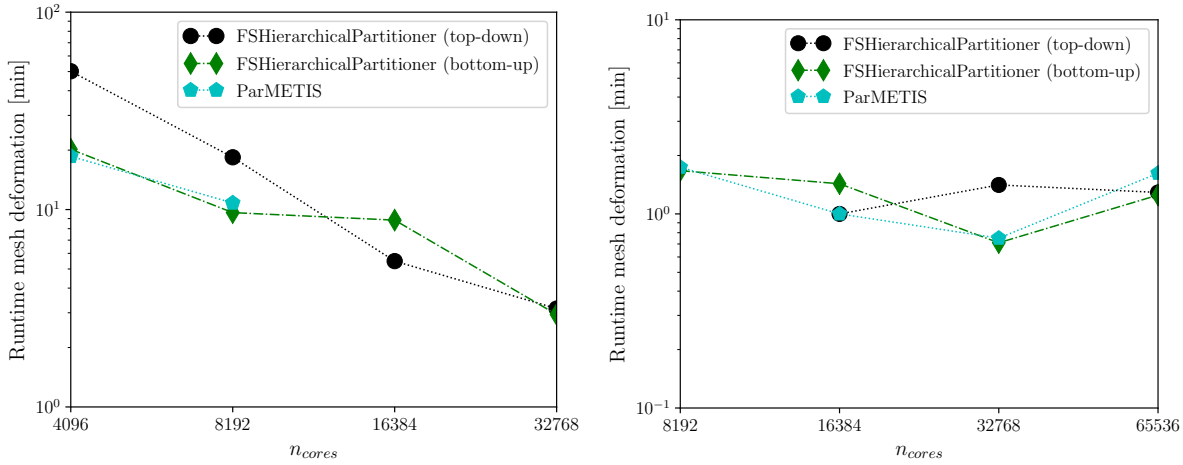

Figure 6: Strong scaling benchmarks of `FSMeshDeformation` runtimes with `PETSc` (on the left) with pure MPI and `Spliss` (on the right) with 4 threads per MPI process after being partitioned by the respective partitioning scheme. Missing data points represent OOM errors.

process. A minimum of 4096 cores was necessary to avoid running into OOM errors. Furthermore, it was only possible to use up to 32 768 cores because the RCB prepartitioner does not work anymore in those cases.

As can be observed in Figure 6, the runtime with `PETSc` depends more on the partitioning than in the `CODA` benchmarks, which is expected due to the choice of the partitioning-dependent solver components.

`ParMETIS` only runs with up to 8192 cores before running into an OOM error. Until that point the `FSMeshDeformation` runtimes after partitioning the mesh with `ParMETIS` are similar to those after partitioning with the bottom-up approach of `FSHierarchicalPartitioner`.

After partitioning the mesh with the top-down approach of `FSHierarchicalPartitioner`, the `FSMeshDeformation` runtimes are by far the longest with fewer cores. However, with 16 384 cores, the runtime is significantly faster than after partitioning with the bottom-up approach. Eventually, both runtimes are similar again with 32 768 cores.

Because the runtimes are so different, comparing the relative parallel efficiencies would not be meaningful here. The reason for the differences in runtime is that the number of iterations necessary to calculate the mesh deformation varies significantly. An explanation for this behavior is the partition dependency of the chosen solution method.

The runtime comparison of the partitioning phases looks similar to that of the `CODA` benchmarks even though the runtimes are a bit longer due to the higher number of processes in pure MPI, as shown in Figure 7. With fewer cores, the top-down approach of `FSHierarchicalPartitioner` is the slowest but it becomes the fastest when using 16 384 or more cores. The runtimes of `ParMETIS` and the bottom-up approach of `FSHierarchicalPartitioner` continously increase with more cores.

The scalings of the test runs using `Spliss` are almost constant. This may be due to the generally lower runtime which is prone to network fluctuations on the HPC cluster and because of the stronger preconditioner for the multigrid solver in `Spliss` which constitutes a higher proportion of the runtime. Moreover, the multigrid solver needs much fewer iterations in order

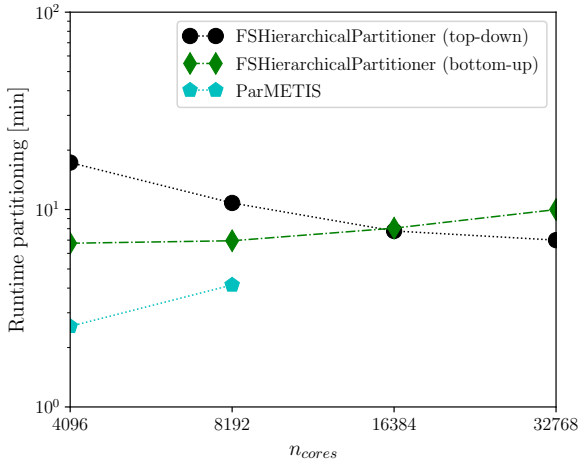

Figure 7: Runtimes of the partitioning processes in the `FSMeshDeformation` + `PETSc` benchmarks, i.e. with pure MPI. Missing data points represent OOM errors.

to get a converged solution than `PETSc`. This was also already observed in [9].

Like in the `CODA` benchmarks, we needed at least 8192 cores for `Spliss` to run but could again go to up to 65 536 cores. Curiously, `Spliss` ran into an OOM error after the mesh was partitioned with the top-down approach of `FSHierarchicalPartitioner` on 8192 cores. This did not happen with the other partitioning approaches.

Since we used the same grid and the same MPI + OpenMP approach for the `FSMeshDeformation` with `Spliss` benchmarks as for the `CODA` benchmarks, the partitioning runtimes are the same for both cases (see Figure 4).

5 CONCLUSIONS

In this paper, we describe an implementation of a hierarchical graph partitioner for unstructured meshes and demonstrated the effect it can have on CFD and CSM simulations. Especially for highly parallel simulations with large meshes, this effect is observable and in some cases, these simulations are only enabled by hierarchical partitioning. Considering future increases in available computational resources, this will probably become even more important.

Furthermore, we could see that `CODA` scales very well with a relative parallel efficiency of 0.87 with 65 536 cores after applying top-down hierarchical partitioning.

On the other hand, while the runtimes of `FSMeshDeformation` with `PETSc` decreased with more used cores, the runtimes did not scale with a clear pattern, suggesting a strong sensitivity to the partitioning. However, `FSMeshDeformation` with `Spliss` does not scale at all for this test case. This can be attributed to the small runtimes due to the application of the AMG preconditioner but, because of these small runtimes, it is not a critical issue. Still, further investigation will be required to explain these curious scaling behaviours to potentially improve it.

The top-down approach of `FSHierarchicalPartitioner` is the fastest of the discussed partitioning approaches in this paper for a large number of used cores while being the slowest for fewer cores. Meanwhile, the bottom-up approach of `FSHierarchicalPartitioner` takes longer with more cores. It is also slower than `FSDM`'s integrated `ParMETIS` interface for fewer cores but gets faster in comparison for more cores thanks to the improve-

ments made in the graph extraction. Moreover, both the top-down and the bottom-up approach still work for 16 384 cores and pure MPI where `FSDM`'s integrated `ParMETIS` interface fails.

`FSHierarchicalPartitioner`'s development will be continued in the future to enable even better partitionings and better scalability of the partitionings itself. Additionally, we will consider integrating other third-party graph partitioning libraries into `FSHierarchicalPartitioner` to see if they can improve the scalability of numerical simulations even further.

Acknowledgements

The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARO is partially funded by "Ministry of Science and Culture of Lower Saxony" and "Federal Ministry for Economic Affairs and Climate Action".

`CODA` is the computational fluid dynamics (CFD) software being developed as part of a collaboration between the French Aerospace Lab ONERA, the German Aerospace Center (DLR), Airbus, and their European research partners. `CODA` is jointly owned by ONERA, DLR and Airbus.

This research was funded in parts via the DLR-internal project HighPoint which was funded by "Ministry of Science and Culture of Lower Saxony".

REFERENCES

[1] 5th AIAA CFD high lift prediction workshop (HLPW-5). `https://hiliftpw.larc.nasa.gov/`. (Last accessed: 23.09.2025).

[2] AMD EPYC™ 7002 series processors. `https://www.amd.com/en/products/processors/server/epyc/7002-series.html`. (Last accessed: 23.09.2025).

[3] CRM-HL. `https://commonresearchmodel.larc.nasa.gov/high-lift-crm/`. (Last accessed: 23.09.2025).

[4] HPC DLR CARO. `https://gwdg.de/en/hpc/systems/caro/`. (Last accessed: 23.09.2025).

[5] Balay, S., Gropp, W.D., McInnes, L.C., and Smith, B.F. Efficient management of parallelism in object oriented numerical software libraries. In Arge, E., Bruaset, A.M., and Langtangen, H.P., editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997. doi:10.1007/978-1-4612-1986-6_8.

[6] Berger, M.J. and Bokhari, S.H. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 36(05):570–580, 1987. doi:10.1109/TC.1987.1676942.

[7] Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. hwloc: a generic framework for managing hardware affinities in HPC applications. In IEEE, editor, *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, Pisa, Italy, Feb. 2010. doi:10.1109/PDP.2010.67.

[8] Chevan, A. and Sutherland, M. Hierarchical partitioning. *The American Statistician*, 45(2):90–96, 1991. doi:10.1080/00031305.1991.10475776.

[9] Cristofaro, M., Fenske, J.A., Huismann, I., Rempke, A., and Reimer, L. Accelerating the FlowSimulator: improvements in FSI simulations for the HPC exploitation at industrial level. In *10th International Conference on Computational Methods for Coupled Problems in Science and Engineering (COUPLED PROBLEMS 2023)*, August 2023. doi:10.23967/c.coupled.2023.001.

[10] Devine, K., Hendrickson, B., Boman, E., St. John, M., and Vaughan, C. Design of dynamic load-balancing tools for parallel applications. In *Proceedings of the 14th International Conference on Supercomputing*, ICS '00, page 110–118, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335231.335242.

[11] Karypis, G. *METIS and ParMETIS*, pages 1117–1124. Springer US, Boston, MA, USA. doi:10.1007/978-0-387-09766-4_500, 2011.

[12] Krzikalla, O., Rempke, A., Bleh, A., Wagner, M., and Gerhold, T. Spliss: A sparse linear system solver for transparent integration of emerging HPC technologies into CFD solvers and applications. In Dillmann, A., Heller, G., Krämer, E., and Wagner, C., editors, *New Results in Numerical and Experimental Fluid Mechanics XIII*, pages 635–645, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-79561-0_60.

[13] Leicht, T., Jägersküpper, J., Vollmer, D., Schwöppe, A., Hartmann, R., Fiedler, J., and Schlauch, T. DLR-project Digital-X – next generation CFD solver 'Flucs'. 2016. doi:10.1007/s13272-015-0179-7.

[14] Reimer, L. The FlowSimulator—a software framework for CFD-related multidisciplinary simulations. In *European NAFEMS Conference Computational Fluid Dynamics (CFD) – Beyond the Solve*, Dezember 2015.

[15] Rempke, A. Deformation of CFD meshes with anisotropic cells in a viscous boundary layer using line-implicit methods. In Dillmann, A., Heller, G., Krämer, E., Wagner, C., and Weiss, J., editors, *23rd STAB/DGLR Symposium on New Results in Numerical and Experimental Fluid Mechanics XIV*, volume 154 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 273–283. Springer Nature Switzerland AG, September 2023. doi:10.1007/978-3-031-40482-5_26.

[16] Stein, K., Tezduyar, T., and Benney, R. Mesh moving techniques for fluid-structure interactions with large displacements. *Journal of Applied Mechanics*, 70:58–63, 01 2003. doi:10.1115/1.1530635.