Performance Analysis and Improvement of FSDM for Large, Highly Parallel Simulations

Jonathan A. Fenske[1], Daniel Vollmer[2]

[1] Institute of Software Methods for Product Virtualization
German Aerospace Center (DLR), Zwickauer Straße 46, 01069 Dresden, Germany
`Jonathan.Fenske@dlr.de`, `https://www.dlr.de/sp`
[2] Institute of Aerodynamics and Flow Technology
German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Brunswick, Germany
`Daniel.Vollmer@dlr.de`, `https://www.dlr.de/as`

Abstract. Modern high-performance computing systems enable larger numerical simulations than previously possible by utilizing more computing power and a larger degree of parallelization. In order for these simulations to run efficiently, it is necessary to have a good load balance among the processes. This paper deals with the problems that occurred during the load balancing with more computing power than previously tested within the `FlowSimulator` framework and their solutions.

Keywords: High–Performance Computing · Performance Analysis · CFD · Mesh Partitioning

1 Introduction

Recent increases in the availability of computational resources have enabled larger computational fluid dynamics (CFD) simulations than ever before. However, such large scale simulations come with new challenges that need to be overcome to be able to efficiently utilize these resources. Some of these challenges will be investigated in this paper and additionally `FlowSimulator`'s current abilities regarding large scale, highly parallelized simulations will be evaluated.

This paper is structured as follows: Section 2 gives overview of the `FlowSimulator` framework. Afterwards, Section 4 discusses problems that arise when running large scale simulations and proposes respective solutions. These large scale simulations are more closely examined in Section 5 in the form of strong scaling benchmark evaluations. This section compares the runtime behavior of the initial code base and an improved code base with solutions for scalability issues applied. Furthermore, the section also discusses current limits of `FlowSimulator` and the best load balancing method currently available in `FlowSimulator`. The paper concludes with a short summary in Section 6.

2 FlowSimulator Framework

`FlowSimulator` is an environment for parallel flow simulations that has been developed and maintained by Airbus and DLR since 2005.

At the heart of this framework lies the `FlowSimulator DataManager` (`FSDM`). As the name suggests, its main purpose is data management for flow simulations. Therefore, `FSDM`'s' core functionalities are the import and export of mesh data in several different mesh formats, data distribution with the purpose of ensuring a good workload balance while minimizing necessary communication between `MPI` processes, and geometry and boundary conditions handling. `FSDM` also provides a range of other functionalities [11][5] but the details of that will be left out here. Even though `FSDM` is implemented in `C++`, most of its functionalities are further wrapped in `Python` using `SWIG` to simplify its usability.

Other software in `FlowSimulator` such as flow solvers (e.g. `CFD for ONERA, DLR, and Airbus` (`CODA`) [9], `DLR TAU` [10], and `TRACE`) is provided as plugins or by providing an interface for `FSDM`. This software can then make use of `FSDM`'s capabilities and access the data stored and managed in `FSDM`.

`FlowSimulator`'s design is illustrated in Figure 1.

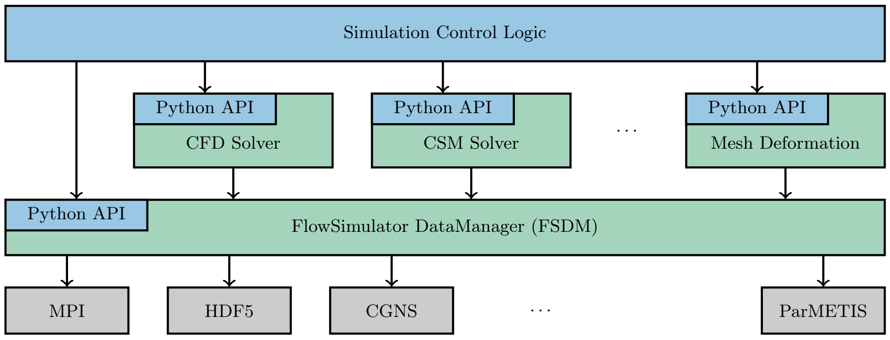

Fig. 1: Components interaction in `FlowSimulator`: `C++` libraries (green), `Python` interfaces (blue), and external libraries (gray boxes) [7].

3 Tools

For the investigations of the problems that are discussed in this paper, the profiling tool of the Slurm Workload Manager was used [3]. This profiling tool enables the local collection of energy, filesystem, network, and task data during simulations in periodic intervals. The resulting data can be gathered in a single file afterwards.

For large scale simulations, this is a resource efficient tool for data analysis offering low runtime overhead and limited volume of collected performance data. In particular, the memory consumption data was helpful for investigating the problems.

[3] `https://slurm.schedmd.com/hdf5_profile_user_guide.html`

Moreover, the built-in timing methods of FSDM were used for further performance analysis.

4 Partitioning problems

In order to fully and efficiently utilizing the ever growing computational power of modern high–performance computing systems for large scale simulations, a good workload balance is essential. This is achieved by employing partitioning methods on the mesh data. Even though the mesh data is already distributed among the MPI processes during the mesh import further partitioning is required because this initial data distribution is only concerned with avoiding memory bottlenecks and does not to provide an optimal partitioning.

For this purpose, an implementation of the Recursive Coordinate Bisection (RCB) method [3] as well as interfaces for the two graph partitioning libraries ParMETIS [8] and Zoltan [4] are provided within the FlowSimulator framework. The RCB method is a simple and fast algorithm that recursively bisects the mesh based on the coordinates of the mesh's nodes. The graph partitioners however require a graph representing the mesh. For this reason, FSDM additionally provides a graph extraction method. The graph can then be partitioned according to prescribed weights of the graph vertices. Furthermore, these methods also try to minimize the edge cut resulting in minimal communication between the MPI processes. Hence, the graph partitioning methods lead to better results than the RCB method. However, the RCB method is still used to speed up the graph extraction process and to avoid memory bottlenecks.

When investigating the functionality of FlowSimulator with respect to large scale simulations, two errors occurred. Both were caused by too much memory consumption during communication between the MPI processes. The computational scalability studies that led to these discoveries are described in Section 5.

The first problem was encountered during the graph extraction. There an out-of-memory error occurred during a call to a method establishing the connectivity of local graph vertices to graph vertices on other MPI processes. This error was caused by large communication buffers during the exchange of the relevant data between the MPI processes. Each MPI process stored all data from all other MPI processes at the same time in its receive buffer. Thus, this problem could be solved by only storing the data of only one other MPI process in the local receive buffer at a time.

The other problem emerged during the execution of the RCB method while the cut data was gathered on each of the MPI processes in order to know how the mesh will be distributed. Here another out-of-memory error occurred because the local cut data from all MPI processes was gathered on the root process which distributed the resulting arrays. Therefore, each MPI process stored its local cut data in arrays that are large enough to contain the cut data from all MPI processes. The root process stored all of these arrays in its receive buffer at the same time which led to the out-of-memory error. It was possible to avoid this error by using MPI_Allreduce for the cut data arrays instead [5].

5 Results

In this section, the results from the computational studies that were conducted with the goal of evaluating and improving the functionality of `FlowSimulator` and especially `FSDM` regarding large scale and highly parallelized simulations are presented. This includes the computational studies that inspired the improvements mentioned in Section 4 and a comparison to an `FSDM` version that already contains these improvements.

5.1 Test Case

The shown test case is based on the high–lift configuration of the Common Research Model (CRM-HL) from NASA's 4th High Lift Prediction Workshop [1]. In particular, the computational results presented in the following paragraphs were obtained with the largest mesh of the *103–ANSA–Unstructured–hiA–Yplus1* family (see Figure 2). This mesh consists of about 723 million cells and 629 million nodes. Hence, it is large enough to be partitioned among a very large number of `MPI` processes and to still provide meaningful results and performance improvements in large scale simulations.

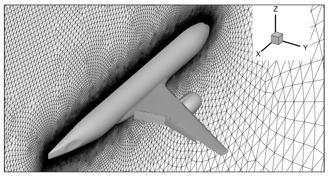

Fig. 2: The CRM-HL grid used for the presented computational studies [5].

5.2 Computational setup

The scalability studies were conducted on DLR's high–performance computing system CARO. CARO consists of $1,364$ compute nodes, each composed of two AMD EPYC 7702 CPUs and 256 GB DDR4 RAM except for 20 compute nodes with 1 TB DDR4 RAM. These CPUs comprise 64 CPU cores that are organized in 16 dies with shared L3 caches [2]. The frequency was fixed to the CPU's base frequency of 2 GHz [4]. The used `FSDM` versions were the development versions stages from 07/2022 and 03/2023. The used `CODA` version was the development version from 03/2023. Both `FSDM` and `CODA` were compiled with `Open MPI` 4.1.1, using the compiler flags `-O3` and `-int-size=64` and the compiler was `GCC` 10.3.0.

[4] `https://www.amd.com/en/product/8766`

5.3 Computational studies

The computational studies were conducted as strong scaling benchmarks, i.e., the experiments were first carried out with the minimum number of necessary compute nodes on CARO and then the number of used compute nodes is subsequently doubled while keeping the problem size constant. First the CRM-HL mesh was imported into `FSDM`. During this mesh import the mesh data was already distributed among the `MPI` processes to avoid bottlenecks during this phase. This initial distribution, however, still necessitated further partitioning in order to have a good workload balance and minimized communication between the processes. To do this, the mesh data was prepartitioned with `FSDM`'s RCB method and subsequently a graph representing the mesh was extracted. Finally, the mesh data is repartitioned using either `ParMETIS`' `ParMETIS_V3_PartKway` k-way graph partitioning method [12] or `Zoltan`'s Parallel Hypergraph Partitioning (PHG) method [6]. After this, a test run with the next generation CFD solver `CODA` is optionally performed to assess the quality of the partitioning. Since `CODA` is capable of multi-threading, the investigations were first done with 4 threads per `MPI` process to exploit the shared L3 caches in CARO's architecture.

Partitioning with ParMETIS (4 threads per MPI process) First the experiments were conducted with `ParMETIS` leading to multiple out-of-memory errors as described in Section 4 and showcased in Figure 3. It can be seen that employing `FSDM`'s graph extraction method was not possible for 16,384 or more cores whereas `FSDM`'s RCB method ran into out-of-memory errors for 65,536 or more cores.

After applying the changes explained in Section 4, the benchmarks were performed again and corresponding results are displayed in Figure 3 as well. With these changes it is now possible to run simulations with at least 131,072 cores on CARO when using 4 threads per process. Furthermore, there is a slight runtime decrease for `FSDM`'s RCB method (see also [5]). Additionally, the load imbalance factor is consistently within the desired tolerance of 1.05.

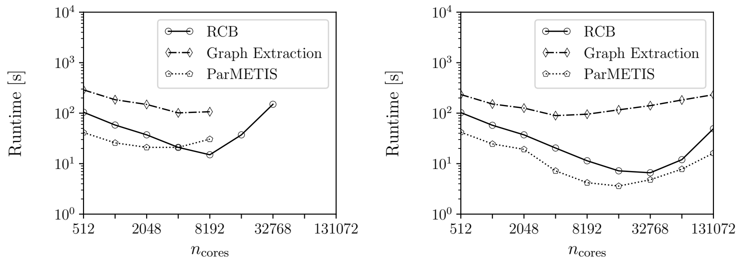

Fig. 3: Comparison of partitioning runtime in the old (left) and improved (right) `FlowSimulator` versions while scaling from 512 cores to $131,072$ cores, using 4 threads per `MPI` process. Missing data points represent failed runs.

The resulting partitioning was then used for CODA to see how it deals with this high number of processes. For this purpose, not a complete simulation was run but just 100 iterations to see if CODA can deal with so many MPI processes. The simulations were conducted using a finite volume discretization, the linearized implicit Euler method for time integration and the Jacobi method as the iterative solver. The outcome of this strong scaling benchmark is shown in Figure 4. This figure illustrates a comparison of the total runtimes, the runtime of the iterative solver, and the preprocessing runtime. Preprocessing aggregates everything that happens before the iterative solver.

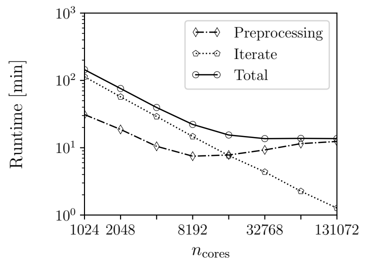

Fig. 4: Strong scaling benchmark of CODA scaling from 1, 024 cores to 131, 072 cores and using 4 threads per MPI process.

It should be noted that in contrast to FSDM itself CODA does not run with this test case for 4 or less compute nodes on CARO due to out-of-memory errors. However, this benchmark demonstrates that CODA is able to run with 131,072 or more cores, almost utilizing the entire CARO cluster. The relative parallel efficiency, i.e., comparing the runtime with the least possible amount of cores with the runtime of the largest executed test run through the formula

$$E_{\text{rel}} = \frac{T_{n_1}}{\frac{n_2}{n_1} \cdot T_{n_2}} = \frac{T_{1,024}}{128 \cdot T_{131,072}},$$

where E is the relative parallel efficiency and T_n is the runtime of the iterative solver for n cores, is approximately 0.7.

Furthermore, Figure 4 also shows that, while the iterative solver takes less and less runtime with each increase in the number of cores, the preprocessing runtime is only reduced until 8,192 cores and increases afterwards.

ParMETIS vs Zoltan In order to see whether there is an advantage in using Zoltan rather than ParMETIS, the same tests were conducted again with Zoltan employing the same workflow. Figure 5 shows a runtime comparison of ParMETIS and Zoltan. Since the same prepartitioning and graph extraction routines were used before both graph partitioners, the runtimes of the prepartitioning and graph extraction routines were the same and are thus excluded here.

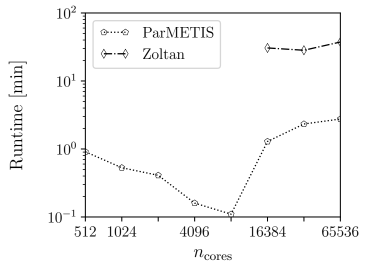

Fig. 5: Comparison of partitioning runtime of `ParMETIS`' `ParMETIS_V3_PartKway` and `Zoltan`'s `PHG` algorithms while scaling from 512 cores to 65, 536 cores, using 4 threads per `MPI` process. Missing data points represent failed runs.

It can be observed that `Zoltan` ran into out-of-memory errors when using 8,192 or less cores. Furthermore, in cases where `Zoltan` worked, it was considerably slower than `ParMETIS`. Thus, for the current implementation of the interfaces for `ParMETIS` and `Zoltan` within the `FlowSimulator` framework it can be recommended to use `ParMETIS`.

Partitioning with ParMETIS (pure MPI) Even though it is not the optimal configuration, the tests were performed again with pure `MPI` to have an indication how well `FSDM` can deal with an even greater number of partitions. As shown in Figure 6, `ParMETIS` fails for 32,768 or more cores whereas `FSDM`'s RCB method fails for 65,536 or more cores. These failed runs were again caused by out-of-memory errors. Further investigations have shown that these errors also occur for much smaller meshes with the same number of `MPI` processes. This indicates that the reason for the out-of-memory errors is the number of `MPI` processes.

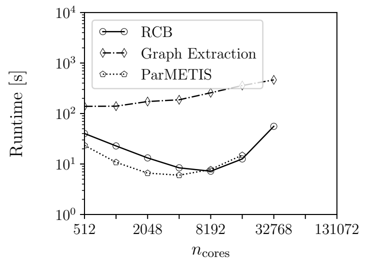

Fig. 6: Comparison of partitioning runtime while scaling from 512 cores to 131, 072 cores, using pure `MPI`. Missing data points represent failed runs.

6 Conclusion

In this paper new challenges and problems arising with the opportunity to utilize more computational power than before were investigated. These problems prevented `FlowSimulator` from running large scale simulations with more than a few thousand `MPI` processes. However, new developments within `FSDM` now enable simulations with at least 32,768 `MPI` processes on 131,072 cores, potentially even on the entire CARO cluster, and through recent `FSDM` releases they have been made available to the entire `FlowSimulator` community. It became clear that optimized communication patterns and memory management are becoming even more crucial when trying to run large scale simulations.

It was further demonstrated that the usage of `ParMETIS` should be preferred to using `Zoltan` within the `FlowSimulator` framework in order to run highly parallelized applications efficiently.

Running simulations with even more `MPI` processes poses further challenges as the currently implemented partitioning methods still run into out-of-memory errors. Hence, further improvements will have to be made in order to enable simulations with an even greater degree of parallelization, possibly by using hierarchical partitioning, i.e., first partitioning the mesh data among the compute nodes and then within the compute nodes for example.

Since such a high degree of parallelization is now possible, more investigations regarding the parallel efficiency of `CODA` or other CFD solvers within `FlowSimulator` for large scale and highly parallelized simulations can now be subject of further study.

Acknowledgments. The authors thank BETA CAE Systems SA for providing the CRM mesh.

This research was funded in parts via the DLR–internal project HighPoint.

This research used resources of the DLR high-performance computing cluster CARO in Göttingen.

`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.

References

1. 4th AIAA CFD High Lift Prediction Workshop (HLPW-4) (2022), `https://hiliftpw.larc.nasa.gov/index-workshop4.html`
2. CARO (2024), `https://gwdg.de/hpc/systems/caro/`
3. Berger, M.J., Bokhari, S.H.: A partitioning strategy for nonuniform problems on multiprocessors. IEEE Transactions on Computers **36**(05), 570–580 (1987)
4. Boman, E.G., Catalyurek, U.V., Chevalier, C., Devine, K.D.: The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. Scientific Programming **20**(2), 129–150 (2012)

5. Cristofaro, M., Fenske, J.A., Huismann, I., Rempke, A., 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), `https://elib.dlr.de/196157/`
6. Devine, K.D., Boman, E.G., Heaphy, R.T., Bisseling, R.H., Catalyurek, U.V.: Parallel hypergraph partitioning for scientific computing. IEEE (2006)
7. Huismann, I., Reimer, L., Strobl, S., Eichstädt, J., Tschüter, R., Rempke, A., Einarsson, G.: Accelerating the FlowSimulator: Profiling and scalability analysis of an industrial-grade CFD-CSM toolchain. In: 9th edition of the International Conference on Computational Methods for Coupled Problems in Science and Engineering (COUPLED PROBLEMS 2021). `https://doi.org/10.23967/coupled.2021.008`
8. Karypis, G.: METIS and ParMETIS pp. 1117–1124 (2011). `https://doi.org/10.1007/978-0-387-09766-4_500`
9. Leicht, T., Jägersküpper, J., Vollmer, D., Schwöppe, A., Hartmann, R., Fiedler, J., Schlauch, T.: DLR-project Digital-X – next generation CFD solver 'Flucs' (2016)
10. Reimer, L.: The FlowSimulator – a software framework for CFD-related multidisciplinary simulations. In: NAFEMS European Conference: Computational Fluid Dynamics (CFD) – Beyond the Solve (2015)
11. Reimer, L., Heinrich, R., Geisbauer, S., Leicht, T., Görtz, S., Ritter, M.R., Krumbein, A.: Virtual aircraft technology integration platform: Ingredients for multidisciplinary simulation and virtual flight testing. In: AIAA SciTech Forum (Januar 2021), `https://elib.dlr.de/140244/`
12. Schloegel, K., Karypis, G., Kumar, V.: Parallel multilevel algorithms for multi-constraint graph partitioning. In: Bode, A., Ludwig, T., Karl, W., Wismüller, R. (eds.) Euro-Par 2000 Parallel Processing. pp. 296–310. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)