A Look at Performance and Scalability of the GPU Accelerated Sparse Linear System Solver Spliss

Jasmin Mohnke(✉) and Michael Wagner

German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, Dresden, Germany
jasmin.mohnke@dlr.de

Abstract. A significant part in computational fluid dynamics (CFD) simulations is the solving of large sparse systems of linear equations resulting from implicit time integration of the Reynolds-averaged Navier-Stokes (RANS) equations. The sparse linear system solver Spliss aims to provide a linear solver library that, on the one hand, is tailored to these requirements of CFD applications but, on the other hand, independent of the particular CFD solver. Spliss allows leveraging a range of available HPC technologies such as hybrid CPU parallelization and the possibility to offload the computationally intensive linear solver to GPU accelerators, while at the same time hiding this complexity from the CFD solver.

This work highlights the steps taken to establish multi-GPU capabilities for the Spliss solver allowing for efficient and scalable usage of large GPU systems. In addition, this work evaluates performance and scalability on CPU and GPU systems using a representative CODA test case as an example. CODA is the 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. The evaluation examines and compares performance and scalability in a strong scaling approach on Nvidia A100 GPUs and the AMD Rome architecture.

Keywords: sparse linear solver · computational fluid dynamics · CFD solver · high performance computing · heterogeneous computing · GPU

1 Introduction

Computational fluid dynamics (CFD) simulations for aircraft aerodynamics are a non-negotiable part in today's aircraft design process. They allow to reduce cost and time of aircraft development and help accelerating the introduction of progressive technologies and improvements. Moreover, high-precision CFD simulations are inevitable for the assessment of future aircraft designs by providing reliable insight into new aircraft technologies and reach best overall aircraft performance. They allow to design quieter, safer, and more fuel-efficient planes.

J. Cano et al. (Eds.): Euro-Par 2023, LNCS 14100, pp. 637–648, 2023.
https://doi.org/10.1007/978-3-031-39698-4_43

For CFD simulations in the aircraft design process, solving the large systems of linear equations that result from implicit time integration of the Reynolds-averaged Navier-Stokes (RANS) equations plays a significant role. Consequently, the utilized linear solver must be tailored to the requirements of the problems and efficiently complete these computations. The sparse linear system solver Spliss meets these requirements independently of a specific CFD solver while leveraging various available HPC technologies [1].

Many current HPC systems take advantage of GPU compute power, as can be seen in the current Top500 list [2]. Of the first ten systems on the list, six have a heterogeneous architecture with accelerators available on compute nodes. Spliss takes advantage of such architectures with the wide range of parallelization approaches it implements, including a hybrid CPU parallelization and offloading to GPU accelerators. Spliss provides CFD solvers the capabilities to efficiently and transparently execute the computationally intensive linear solver on new architectures and hardware accelerators such as GPUs. This way, the CFD solver can leverage new architectures and hardware accelerators without the necessity of any code adaptation in the CFD solver. One of the CFD solvers that utilize Spliss is CODA. CODA is the 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.

The contribution of this work is, first, a presentation of the improvements made to Spliss to allow an efficient scaling to a large number of GPUs and, second, an evaluation of the achieved performance and scalability using a test case with the CODA CFD software as an example. The evaluation includes a performance assessment of GPU accelerated Spliss with CODA on the JUWELS Booster system with Nvidia A100 GPUs in comparison to a CPU-only execution on German Aerospace Center's CARO HPC system based on AMD Rome CPUs.

This work starts by introducing the software ecosystem in Sect. 2, followed by discussing the improvements to enable acceleration distributed among multiple GPUs in Sect. 3 and their impact on overall performance. In Sect. 4, the evaluated HPC systems and the test case are described and performance and scalability results are presented and compared. Finally, Sect. 5 summarizes the presented work and draws conclusions.

2 Background

At the German Aerospace Center (DLR), the development of computational fluid dynamics software has a long history. Today, the *TAU* CFD package [3] has been in production in the European aircraft industry, research organizations and academia for more than 20 years and was, for instance, used for the Airbus A380 and A350 wing design. As state-of-the-art for its time, TAU implements a classical MPI parallelization to compute steady and unsteady external aerodynamic flows using a second order finite-volumes discretization.

In 2012 DLR began the development of a new, flexible, unstructured CFD solver called *Flucs* [4] from the ground up. The focus of this new development

was set on, first, a flexible and comprehensive parallelization concept suited for current and future HPC systems and, second, on algorithmic efficiency using strong implicit solvers, higher-order spatial discretization via the Discontinuous Galerkin method featuring hp-adaptation in addition to finite volumes with maximum code share, and seamless integration into Python-based multi-disciplinary process chains via *FlowSimulator* [5,6]. While the development of Flucs had been started at DLR, it since has become part of a larger cooperation that is driven by Airbus, the French aerospace lab ONERA, and DLR. The joint development of the CFD software based on Flucs was named *CODA* (CFD for ONERA, DLR and Airbus) to honor the new collaboration and the involvement of all three partners pursuing the joint effort and co-development.

Similar to TAU, the CODA CFD software uses classical domain decomposition to utilize distributed-memory parallelism via MPI and, additionally, the GASPI [7] implementation GPI-2 as an alternative to MPI, which allows for efficient one-sided communication to reduce network traffic and latency. In addition, CODA supports the overlapping of halo-data communication with computation to hide network latency and further increase scalability. Besides classical domain decomposition, CODA employs a hybrid two-level parallelization to utilize shared-memory parallelism for multi- and many-core architectures [8]. CODA implements sub-domain decomposition, where each domain is further partitioned into sub-domains, each of which being processed by a dedicated software thread that is mapped one-to-one to a hardware thread to maximize data locality. The hybrid approach allows utilizing all layers of parallelism and providing a flexible adaption to different hardware architectures [9,10].

An integral part of the CODA software architecture is the integration of the before mentioned sparse linear system solver *Spliss* [1]. Spliss is used for solving linear equation systems for implicit time integration methods, e.g. for the test case used in this work and is a linear solver library that, on the one hand, is tailored to the requirements of CFD applications but, on the other hand, independent of the particular CFD solver. It is specialized to solve large sparse systems of linear equations, providing a sparse matrix structure with dense blocks of fixed or variable sizes and a range of different iterative solver components and preconditioners that can be stacked as needed. Spliss leverages available compute resources through mechanisms such as one-sided communication, hybrid parallelization and the use of accelerators, i.e. GPUs, to take advantage of heterogenous architectures while hiding the complexity of these hardware-specific optimizations from a CFD solver such as CODA.

3 Porting Spliss to GPU

This section takes a look at GPU acceleration for the sparse linear system solver Spliss. This can be taken advantage of from CPU-only codes such as CODA by simply linking against a version of Spliss compiled for GPU.

Porting the linear solver components to GPU consists of both the initial implementation enabling single GPU usage through computation kernels as well

as ensuring that performance scales to multiple GPUs. The process was aided by the Nvidia performance analysis tools [11] NSight Systems (profiling the timeline of GPU usage) and NSight Compute (analysis on a GPU kernel level) for multiple iterations of improvements on general GPU usage and specific computation kernels. The focus of this work is to enable efficient multi-GPU usage including good scalability. This ensures that future improvements made on computation kernel level can also benefit at scale for distributed execution.

A baseline for all following changes and measurements is established as the initial GPU port with multi-GPU usage enabled in code. The computational load is expected to be close to balanced across all processes since this is provided by the according CFD solver. In addition, we assume that the targeted compute architectures are comprised of multiple of the same type of GPU, which is the case for most or all available systems. As a result, Spliss multi-GPU means each process is offloading to a single GPU, which stays consistent throughout the entire runtime. This baseline is displayed as the gray set of bars in Fig. 1, where the solid bar represents the runtime of the entire iteration phase and the shaded bar the time spent within the linear solver (including host to device and vice versa data transfers), which makes up about 75 % of the runtime.

3.1 Implementation Changes

By performing an initial analysis we found that the following steps need to be implemented to establish efficient multi-GPU capability and to enable the acceleration that can be achieved by offloading computation to a single GPU also to scale to a distributed use-case using multiple GPUs.

Data Movement. At the start, measurements showed an increased runtime for distributed GPU usage. An analysis of the runtime behavior with the Nvidia tools revealed redundant data copy operations from host to device as well as within host memory. We resolved all redundant transfers from host to device and all redundant copy operations in the host memory. The impact of these improvements is highlighted with the blue set of bars in Fig. 1. The improved version achieves a performance gain of about 20 % for the entire iteration phase and about 30 % for linear solver.

CUDA-aware MPI. In addition, we identified that when offloading data and computation to GPU the amount of time spent in point-to-point MPI communication needed for halo updates during the key computation of the matrix-vector multiplication was significant. By taking advantage of CUDA-aware MPI capabilities [12], we avoid the need to transfer notable amounts of data between host and device when the MPI communication is executed via the CPU host. We used this to improve the halo exchange by directly passing a pointer to GPU memory to MPI. As a result, an explicit data transfer from device to host on the sender and an explicit data transfer from host to device on the receiver is no longer needed. In order to facilitate this halo exchange we pack the non-contiguous

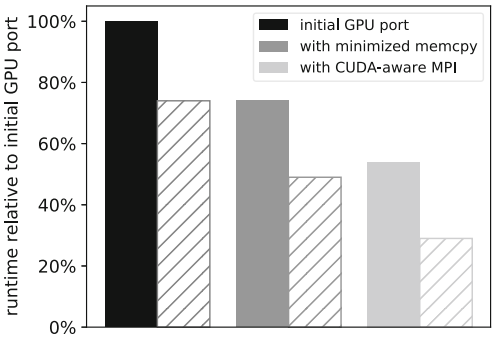

Fig. 1. Runtime of different improvements in relation to initial GPU port for a small test case on four A100 GPUs with the entire time integration iterate phase in solid colors and the linear solver within that in shaded colors.

halo data on the device to a contiguous GPU buffer that can be passed directly to the CUDA-aware MPI installation. This allows the MPI library and underlying frameworks to make the decision if and when to copy relevant chunks of data to host memory or communicate directly from and to device memory for best performance. Figure 1 shows that the improvements using CUDA-aware MPI reduce the runtime of the iteration phase to about 50 % of the baseline. In this case, the time spent in the linear solver, i.e. the part running on GPUs, is reduced to about half of the iteration time.

3.2 Adjustments at Runtime

Next to the above changes made to enable multi-GPU usage, we can take advantage of optimizations at runtime: GPUDirect Accelerations [12,13] and Nvidia Multi Process Service (MPS) [14]. While the former is automatically applied as deemed appropriate by the CUDA-aware MPI software stack, the latter can be enabled by the user when considered necessary.

Nvidia Multi Process Service. Generally, the goal for GPU acceleration is using the hardware as efficiently as algorithmically possible while maintaining little overhead. This favors having only one MPI process offload computation to one distinct GPU each. However, other components of the software framework for the CFD solver (except the linear solver) may benefit from not constricting the number of processes by the number of available GPU accelerators. For instance, with CODA when no GPU acceleration is used the time spent in the non-linear part of the iteration phase is about 5–10 % and the time in the linear solver about 90–95 %. With enabled GPU acceleration the ratio is closer to half and half. As a result, all computation outside of the accelerated linear solver needs to also be executed as efficiently as possible. As will be discussed in Sect. 4, for example, on the AMD Naples and Rome architecture best hybrid performance

can be achieved using only four OpenMP threads per MPI rank, i.e. using 16 or 32 MPI processes, respectively. This suggests that a further performance gain could be achieved through using Nvidia's MPS to mitigate a restriction on the number of MPI processes given by the number of available GPUs. MPS enables multiple processes to simultaneously offload to the same GPU as efficiently as possible. While this predominately benefits the parts of computation that take place on CPU by being able to use the best hybrid parallel configuration, there is also a minimal benefit for the Spliss linear solver as long as there are only a few processes submitting to each GPU. The observation can be traced back to the host to device copies necessary at the start of the linear solver execution and is negated by overhead when more processes offload work to the same GPU.

4 Evaluation

This section, first, introduces the test systems and the test case, second, provides an assessment of the scalability of Spliss with CODA on the German Aerospace Center's CARO HPC production system and, third, compares the performance and scalability of CODA with Spliss executed on Nvidia A100 GPUs on the JUWELS Booster module at Jülich Supercomputing Center.

4.1 The Test Systems

The *Cluster for Advanced Research in Aerospace* CARO is one of the two German Aerospace Center's main HPC systems. It was ranked at 135 in the Top500 list of 11/2021 providing 3.5 TFlop/s out of 5.6 TFlop/s theoretical peak performance [2]. The system offers 1364 compute nodes, whereas each compute node consists of two AMD EPYC 7702 (64 cores at 2.0 GHz). In total, the system offers 174,592 compute cores.

Similar to the AMD Naples architecture, the AMD Rome architecture within this system includes 16 NUMA (non-uniform memory access) domains and three NUMA distances: first, to the memory of the seven other cores on the same die, second, to the memory on the 7 other dies on the same chiplet (socket) and, third, to the memory located on the other chiplet. In addition, only four of the eight cores on each die share a last level cache (L3 cache), which presents an additional difference in memory access latency depending on the locality of the data; whether it is in the shared L3 cache of the according core or in the adjoining L3 cache on the same die.

The second test system is the JUWELS Booster module at Jülich Supercomputing Center. The JUWELS Booster module was ranked at 7 in the Top500 list of 11/2020 providing 44.1 PFlop/s out of 71.0 PFlop/s theoretical peak performance; making it the most powerful system in Europe at that time. The system offers 936 compute nodes, whereas each compute node consists of two AMD EPYC 7402 and four Nvidia A100 GPUs with four-times InfiniBand HDR (Connect-X) interconnect. In total, the system offers 3744 GPUs.

4.2 The Test Case

The test case for the evaluation is based on the NASA 3D Onera M6 wing test case [15], which simulates the external airflow at transonic speed and computes typical characteristics like air velocity and direction, pressure and turbulence via a turbulence model. The NASA 3D Onera M6 wing test case is well studied and provides experimental data as well as numerical solutions by other CFD applications for comparison. For the test case, CODA solves the Reynolds-averaged Navier-Stokes equations (RANS) with a Spalart-Allmaras one-equation turbulence model in its negative form (SAneg). It uses a second-order finite-volume spatial discretization with an implicit Euler pseudo-time integration based on local (pseudo) time steps scaled via an up-ramping CFL number starting at 5.0. For the linear problem, a Block Inversion preconditioned Block-Jacobi Solver is applied to solve the linear system. The flow conditions are outlined by the following parameters: the Mach number is set to 0.84, the Reynolds number to 14.6e6, and a fixed 3.06° angle of attack is set.

For this case, the vast majority of the iteration phase is spent in the linear solver, which makes it ideal to offload the computationally intensive linear system solving to GPUs. Measuring the iteration time of CODA provides a very close estimation of the performance and scalability of Spliss within a real-world example. In addition, it highlights the performance of the entire simulation, i.e. it includes all time spent for data transfer between CPU and GPU, CPU-only sections as well as communication and synchronization; not just the time for the GPU kernels. While results may be biased by CODA, the measurements show the combined performance and scalability of CODA with Spliss since performance degrading effects accumulate. In this sense, CODA and Spliss may achieve better performance and scalability individually.

The test case operates with a medium-sized, unstructured mesh with 69.2 million volume elements. This way, it is large enough to achieve good performance per GPU with the chosen linear solver components but still small enough to allow for a reasonable strong scaling evaluation.

4.3 Measurement Setup

As a reference, we evaluated the scalability of CODA with the above test case on the CARO HPC system. For the scalability evaluation all software threads are bound to a hardware thread to ensure thread affinity and using one hardware thread per core. For the reference, we compared different hybrid-parallel setups suitable for the specific memory and NUMA layout of the AMD Rome architecture. The comparison showed that best hybrid-parallel performance is reached when using only four OpenMP threads per MPI process, so that these threads share the same last level cache. This is consistent with its predecessor, the AMD Naples architecture, and stands in contrast to other architectures, for instance, the Intel Cascade Lake architecture, where comparable performance for all hybrid setups was obtained [10,16]. Consequently, we chose the best hybrid setup, i.e. with 32 MPI ranks and 4 OpenMP threads per node, as a reference.

For the GPU measurements all software threads are bound to a hardware thread to ensure thread affinity and using one hardware thread per core, too. Similarly, we compared different hybrid-parallel setups that matched well with specific memory and NUMA layout of the AMD Rome architecture on the host as well as the number of installed GPUs, namely 4 MPI processes with 12 OpenMP threads each, 8 process with 6 threads, 16 processes with 3 threads and 48 processes MPI-only. Out of these, the setup with 8 MPI processes and 6 OpenMP threads each achieved the best overall performance and, as a result, was selected to represent the GPU measurements.

For all GPU measurements the linear systems solving via Spliss is offloaded to the Nvidia A100 GPUs, while the non-linear part in CODA is executed on the host CPU. The offloading is achieved by simply linking against the GPU-version of Spliss; without any modifications to the CODA source code or installation.

4.4 Comparing CPU and GPU Performance and Scalability

The CPU reference measurement runs the above described test case as strong scaling setup, i.e. the problem size is fixed for increasing core counts, and contains measurements from 2,048 to 12,288 cores or 16 to 96 nodes, respectively. This represents an appropriate range for the given mesh size, with on average about 5600 elements per thread at the largest core count. In this range, the test case achieves a near ideal speedup and a compute performance that matches experiences from previous measurements, which makes it a valid and strong reference to compare the GPU measurements against.

The GPU measurement runs the same strong-scaling test case from 8 to 128 GPUs or 2 to 32 nodes, respectively. This represents an appropriate range for the given mesh size, where two nodes is the minimum number of nodes to fit the simulation data into main memory and at 32 nodes the individual GPU utilization starts to decline. At 32 nodes and 128 GPUs, respectively, the given test case can theoretically achieve about 85 % of the maximum single GPU performance since there is simply not enough computational load to meet the massive demand of parallel load for the A100 GPUs. Since there is an additional decrement for running multiple processes via MPS (two in this case) the resulting utilization is about 70 % of the maximum single GPU performance. For further increasing numbers of GPUs the resulting individual GPU utilization declines faster than the parallel efficiency within Spliss or CODA, i.e. increasing the number of GPUs would necessitate larger input data to match the GPUs demand for computational load.

On the GPU system the test case achieves a scaling efficiency of 82 %, i.e. a speedup of 13.1 of ideally 16 for 128 GPUs, for the entire CODA iteration phase including the linear part in Spliss running on the GPUs, the non-linear part in CODA running on the CPUs and all transfers between host and device. Whereas, the linear part in Spliss makes up about 60 % for 8 GPUS up to 75 % for 128 GPUs of the iteration phase. The linear part in Spliss running on the GPUs on its own, achieves a scaling efficiency of 66 %, i.e. a speedup of 10.5, which is mainly due to the above described decreasing individual GPU utilization to about 70 %

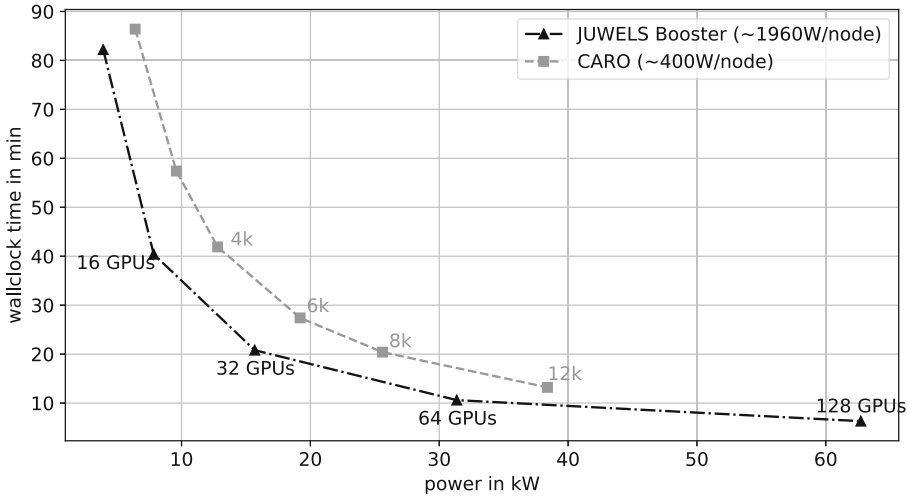

Fig. 2. Runtime comparison on CARO (AMD Rome) and JUWELS Booster (4x Nvidia A100) with the M6 wing testcase in relation to power consumption.

on 128 GPUs. The remaining efficiency decrease of about 4 % is due to increasing ratio of synchronization to computation, whereas the synchronization includes MPI communication between the GPUs and transfers between host and device.

When comparing the relative performance per node, i.e. 16 CARO nodes (128 cores AMD Rome each) versus 16 JUWELS Booster nodes (48 cores AMD Rome and 4 GPUs each), the GPU nodes outperform the CPU nodes by a factor of up to 8.4, whereas with increasing scale the factor declines to 6.7 due to the above described reduced GPU utilization for the given test case. While this node-wise comparison matches two high-end nodes that were both state-of-the-art for CPU and GPU systems at their similar installation time, it must be considered that the GPU nodes are significantly more costly in both acquisition and operation, the later due to their much higher power consumption.

To allow for a fairer assessment of both systems, we compare the runtime in relation to estimated power consumption using the Thermal Design Power (TDP) value, as well. A single CARO compute node has a power consumption of about 400 W, which is composed of the 2× 200 W of the AMD Epyc 7702 CPUs. In comparison, a single JUWELS Booster node has a power consumption of about 1960 W, which is made up of the 2× 180 W of the AMD Epyc 7402 CPUs and the 4× 400 W of the Nvidia A100 GPUs. Since both systems are production systems it is infeasible to retrieve the proportionate power consumption of further involved components such as network, storage or cooling. Nonetheless, their impact can be assumed to be insignificant in comparison to the nodes themselves.

Figure 2 shows the runtime comparison for the test case on CARO and JUWELS Booster in relation to power consumption. It depicts the runtime for

2,048 to 12,288 cores on CARO and 8 to 128 GPUs on JUWELS Booster on the vertical axis and the power consumption on the horizontal axis, which is obtained by multiplying the number of nodes for each data point with the power consumption of the according node. For the test case, Spliss achieves a significant speedup of 1.6 to 1.9 on the GPU system even when equated for power consumption.

Key Results. The evaluation presents three key results: First, the above described improvements enable CFD solvers such as CODA to leverage the benefits of offloading the computationally intensive linear equation solver to GPU accelerators without any modifications to the CFD solver itself and achieve a speedup of up 8.4 in a node-wise comparison and a speedup up to 1.9 in a power-equated comparison. Second, Spliss' GPU version allows to achieve a similar performance on significantly less compute nodes, which can provide better scalability, particularly, towards exascale systems since fewer nodes allow for less MPI processes, less MPI communication and, thus, less communication overhead. Third, due to the significant acceleration of the linear part on GPUs, the non-linear part that is executed on the CPU becomes more prominent: where it is typically about 5–10 % of the iteration phase it increases to about 40 %. Since the main purpose of the usage of Spliss is to hide the complexity of hardware-specific optimizations from the CFD solver, the non-linear part in the CFD solver might remain exclusive to CPUs by design. To further increase the performance in this case would require a) larger workloads at the given scale, which would be quite typical for industrial applications or b) move to systems that have more performant CPUs in the GPU nodes. For instance, a hypothetical node that replaces the CPU in the JUWELS Booster system with the state-of-the-art CPU from the CARO system would provide an additional speedup of about 20 %, i.e. a power-equated speedup of about 2.3 for the GPU version over the CPU version on CARO.

5 Conclusion

The sparse linear system solver Spliss efficiently solves the large sparse systems of linear equations that result from the time integration of the Reynolds-averaged Navier Stokes (RANS) equations. It takes advantage of various current HPC technologies while hiding the resulting complexity from the CFD solver. The heterogenous compute node architecture consisting of CPUs and GPUs that can be found on many current top HPC systems is one of them. In combination with an efficient, hybrid CPU parallelization, Spliss and the improvements of this work allow the performance gain achieved with a single GPU to scale to large distributed systems consisting of hundreds of GPUs. We outlined the steps taken to enable efficient multi-GPU usage for the Spliss linear solver reducing the runtime in a distributed set-up on Nvidia A100 GPUs by up to 50 %. Additionally, using the NASA 3D Onera M6 wing test case for Spliss with the CODA CFD software, we looked at performance in a strong scaling scenario on current

HPC systems. We showed that GPU acceleration of Spliss can yield a up to 8.6 times speedup over state-of-the-art CPU systems or a up to 1.9 times speedup when equated for power consumption.

Acknowledgements. Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Italy, Slovenia, Spain, Sweden, and France under grant agreement No 101092621.

References

1. Krzikalla, O., Rempke, A., Bleh, A., Wagner, M., 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., Wagner, C. (eds.) STAB/DGLR Symposium 2020. NNFMMD, vol. 151, pp. 635–645. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79561-0_60
2. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: The 60th Top500 list (2022). https://www.top500.org/lists/top500/2022/11/ Accessed 23 Feb 2023
3. Schwamborn, D., Gerhold, T., Heinrich, R.: The DLR TAU Code: recent applications in research and industry. In: Proceedings of the European Conference on Computational Fluid Dynamics, ECCOMAS CFD (2006). https://elib.dlr.de/22421
4. Leicht, T., et al.: DLR-project digital-X – next generation CFD solver 'Flucs'. Deutscher Luft- und Raumfahrtkongress (2016). https://elib.dlr.de/111205
5. Meinel, M., Einarsson, G.: The FlowSimulator Framework for Massively Parallel CFD Applications. In: PARA 2010 (2010).https://elib.dlr.de/67768
6. Huismann, I., et al.: 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) (2021). https://doi.org/10.23967/coupled.2021.008
7. Alrutz, T., et al.: GASPI - a partitioned global address space programming interface. Facing Multicore-Challenge III, LNCS **7686**, 135–136 (2013). https://doi.org/10.1007/978-3-642-35893-7_18
8. Jägersküpper, J., Vollmer, D.: On highly scalable 2-level-parallel unstructured CFD. In: 8th European Congress on Computational Methods in Applied Sciences and Engineering (2022) https://doi.org/10.23967/eccomas.2022.208
9. Wagner, M., Jägersküpper, J., Molka, D., Gerhold, T.: Performance analysis of complex engineering frameworks. In: Mix, H., Niethammer, C., Zhou, H., Nagel, W.E., Resch, M.M. (eds.) Tools for High Performance Computing 2018 / 2019, pp. 123–138. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-66057-4_6
10. Wagner, M.: Scalability evaluation of the CFD solver CODA on the AMD Naples architecture. In: Sustained Simulation Performance 2021, pp. 95–106 (2023). https://doi.org/10.1007/978-3-031-18046-0_7
11. Nvidia: Performance Analysis Tools. https://developer.nvidia.com/performance-analysis-tools. Accessed 20 Feb 2023
12. Kraus, J.: An introduction to CUDA-aware MPI (2013). https://developer.nvidia.com/blog/introduction-cuda-aware-mpi. Accessed 17 Feb 2023
13. Nvidia: GPUDirect. https://developer.nvidia.com/gpudirect. Accessed 20 Feb 2023

14. Nvidia: Multi-Process Service (2021). https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf
15. Daniel Destarac, Antoine Dumont: ONERA M6 Wing Test-Case, Original and TMR (2016). https://turbmodels.larc.nasa.gov/onerawingnumerics_val.html. Accessed 17 Feb 2023
16. Wagner, M., Mohnke, J., Krzikalla, O., Rempke, A.: Evaluating performance and scalability of the sparse linear systems solver Spliss. In: Methods, Tools and Technologies for Design in Aviation (to be published)