The 9th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2024
3–7 June 2024, Lisboa, Portugal

ACCELERATING THE FLOWSIMULATOR: MIXED PRECISION LINEAR SOLVERS IN INDUSTRIAL GRADE CFD

Johannes Wendler, Immo Huismann, Olaf Krzikalla and Arne Rempke

German Aerospace Center (DLR)
Institute of Software Methods for Product Virtualization
Zwickauer Str. 46, 01069 Dresden, Germany
https://www.dlr.de/sp/en/
{firstname.surname}@dlr.de

Key words: Mixed Precision, CFD, Sparse Linear Solver, CODA, Spliss

Summary. A common way to speed up linear solvers is to reduce the precision where it is not needed (e.g. in the preconditioner). Therefore, we implemented mixed precision in a sparse linear system solver called Spliss. In Spliss, different solver components can be chained together to form flexible solver stacks with preconditioners. A new solver component casts the preconditioners down to single precision and their results back up to double precision. Where and how to use mixed precision in the solver stack can then be chosen by the user. The internal data structures of Spliss also allow to apply the cast to the system matrix only. In this case, the solution and the right-hand vectors, as well as the actual calculations, are still in double precision. This allows for higher accuracy compared to a full cast to single precision while still benefiting from the largest memory bandwidth savings, the matrix. We benchmarked the CRM test case using the CFD solver CODA, which uses Spliss as linear solver. Depending on the solver configuration and the used computer hardware, we observed a speedup of the simulation runtime in CODA of up to 80 % and even more when dealing with cache effects in some cases.

1 Introduction

The CFD vision 2030 study by Slotnick et. al. [11] describes advancements in high performance computing (HPC) and multidisciplinary analysis and optimization (MDAO) as two of the most important and groundbreaking topics of CFD research of this decade. For aviation to produce less environment damaging exhaust gases and noise pollution in the future, we need bigger, more accurate simulation tools, which perfectly integrate into the design processes of aircrafts. To achieve such large and resource hungry simulations in an ecological manner, we need highly optimized simulation tools that minimize time/energy to solution. In this work we focus on one part of the optimization of our CFD solver CODA.

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. It features finite volume (FV) and discontinuous Galerkin (DG) methods on unstructured grids with various options for partial differential equations (PDEs), time integration methods, and turbulence models [8, 4]. Its core is written in C++ with a Python control layer above, that manages input, output, parameters and the data exchange with other simulation tools over the FlowSimulator framework [10].

Previous studies on the performance of CODA have revealed that for steady state implicit simulations the majority of runtime is spent inside the linear solver, sometimes even over 90% [5]. Therefore we decided to introduce mixed precision into the linear solver of CODA. Anzt et. al. [1] showed that the time and energy to solution can be drastically improved with mixed precision, especially when employing GPUs. Using single precision instead of double precision floating point values is expected to speed up our memory bound linear solver by up to a factor of two, due to half as much data needing to be streamed through the memory interface to the CPU. Walden et. al. [12] show exactly this behavior for usually memory bound block sparse linear systems. Using reduced precision only in specific parts of the code has the benefit of not affecting the accuracy of the obtained solution while gaining substantial speedups, as shown by Brogi et al. [2], Freytag et. al. [3] and Ina et. al. [6]. After implementing single precision into our linear solver preconditioners to obtain a mixed precision solver we proceed with assessing the performance gain while validating the results on a industry representative benchmark case.

2 Implementation in Spliss

As linear solver CODA makes use of the Sparse Linear System Solver (Spliss) library, developed by DLR [7]. Spliss uses sparse matrices consisting of dense blocks (see Figure 1), that can be of fixed or dynamic size and can have varying data types (e.g. complex values in diagonal blocks and real values in the rest). The algorithms and data structures are catered for CFD needs and with its focus on HPC Spliss features hybrid parallelization with GASPI or MPI combined with threading. Different solver components can be combined together almost arbitrarily, forming a so called solver stack, with a top level solver and a chain of preconditioners.

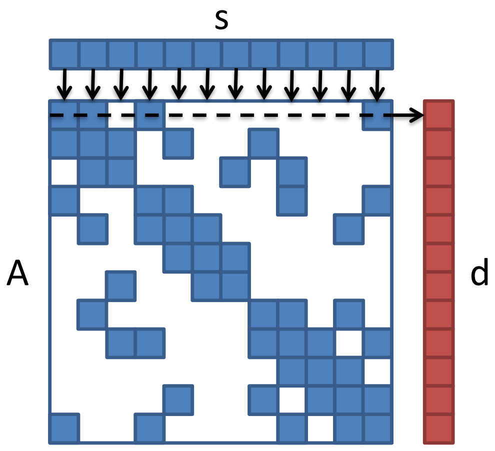


Figure 1: Visual example of a matrix vector multiplication operation with a sparse matrix and vectors consisting of dense blocks.

The core idea for mixed precision for Spliss is to introduce a new solver component which can connect a high precision solver with a low precision preconditioner. This preserves the flexibility of solver combinations and allows for the low precision to be used anywhere in the stack (see Figure 2).

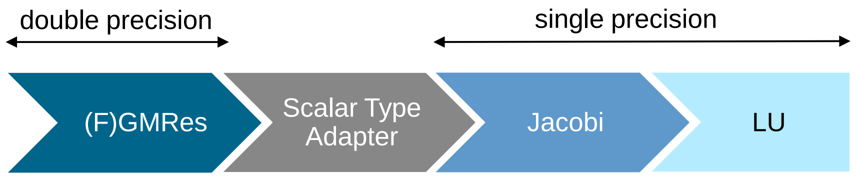


Figure 2: Example solver stack containing a scalar type adapter which connects a single precision Jacobi - LU preconditioner to a double precision (F)GMRes solver.

On Solver entry, where every solver component does a preparation step, this new scalar type adapter component copies and casts the matrix down to single precision for it to be used by the successors (preconditioners). During each time step of the predecessor the scalar type adapter then copies and casts the right hand side (RHS) vector down to single precision and hands it to the successor. The result of the preconditioner is the copy casted back to double precision and returned to the predecessor to be used in its iteration.

Making use of the Spliss feature for different underlying data types for the blocks inside a matrix, we also can implement a special kind of mixed precision. For this we keep the interface data type of all solvers, matrices and vectors as double precision and only change the data types of all blocks inside the matrix to single precision. With the interface data type being unchanged, all computations are carried out in double precision and the data of the single precision blocks are casted up just when they are used in the low precision solvers. Figure 3 illustrates the underlying data structure. With this so called "matrix only mixed precision", the scalar type adapter then does not need to cast the vectors back and forth between the predecessor and successor solver component and all calculations are carried out in higher precision than for full mixed precision. The main performance gain, which is less matrix data to be streamed through the memory interface, is kept in place all the while.

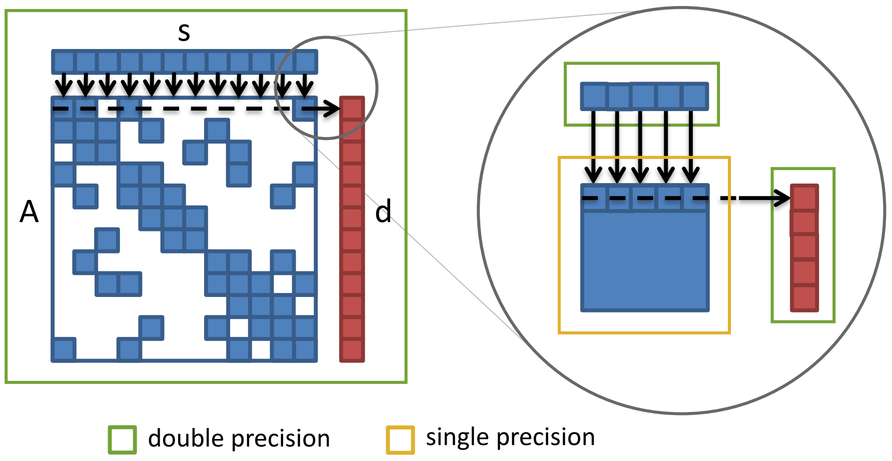


Figure 3: For matrix only mixed precision the whole matrix vector operation looks like it is performed in double precision to the outside interface (indicated by the green encapsulating square on the left). Inside the dense matrix blocks are in single precision though and their content gets cast up to double for each computation inside the block operation just in time (indicated by the yellow encapsulating square in the zoom in on the right).

3 Test Case and Test System

As a representative testcase for industry relevant applications we choose the Common Research Model (CRM) by NASA [9]. It features a simplified half body of an airplane originally as a block structured mesh. For the purpose of testing CODA, an unstructured solver, we convert the mesh into two different meshes, to also test if the element types have an effect on the mixed precision. One mesh consists completely of prism elements, while the other one is a hybrid mesh of prisms and a majority of tetrahedrons. We choose Finite Volumes (FV) of second order as discretization in space with RANS Spallart-Allamaras negative turbulence model and the linearized implicit Euler time stepping method. As linear solver we choose a GMRES-(scalar type adapter)-Jacobi-LU solver stack. We perform runtime measurements of a fixed number of iterations, only depending on the problem size. As in all preliminary tests, no impact of mixed precision to overall convergence behavior of CODA was observed, this suffices.

The system we run our measurements on is the DLR HPC cluster CARO. It consists of 1364 nodes, each with two AMD EPYC 7702 CPUs (2x64 Zen2 cores), reaching a peak performance of 3.46 PetaFlop/s. In terms of L3 cache, the EPYC 7702 has 16 MB per 4 cores, resulting in 512 MB for one whole node. For all runs we use the preinstalled CODA version 2024.02.0.

Due to resource constraints, we are only able to analyze the two smallest mesh versions of the CRM test case here ("tiny" and "coarse"). To estimate at which number of used nodes the whole linear problem of Spliss starts to fit into the L3 cache, we need an approximate size of the matrix, RHS vector and solution vector. With one line of the sparse linear equation system matrix corresponding to one cell it contains one block for the cell itself and one block for each neighbor. With the two vectors containing one Block per cell we arrive at the formula for the problem size:

$$\text{Problemsize} = n_{\text{elements}} \times (2 \times n_{\text{statevariables}} + (1 + n_{\text{neighbors}}) \times n^2_{\text{statevariables}}). \qquad (1)$$

For our chosen equation and turbulence model the state contains 6 variables, resulting in a matrix block size of 6×6 floating point values. This results in an approximate cache cutoff, the minimal number of nodes at which the whole linear system fits into the L3 cache (see Table 1). For simplicity we ignore the negligible effects of halo elements, sparsity indices and the size of the vectors not being halved for matrix only mixed precision here.

Table 1: Numbers of cells, size of the linear problem in memory and the subsequent minimal number of nodes for the problem to fit into the L3 cache for the different meshes.

Mesh	# prisms	# tetrahedrons	Size [GB] Regular / Mixed Precision	L3 cache cutoff [# nodes] Regular / Mixed Precision
Tiny - prism	1.3M	0	2.4 / 1.2	4 / 2
Tiny - hybrid	0.4M	2.6M	4.75 / 2.375	9 / 4
Coarse - prism	4.3M	0	7.8 / 3.9	15 / 7
Coarse - hybrid	1.4M	8.6M	16.8 / 8.4	33 / 16

4 Results

The first indicator of performance is the runtime of the fixed number of timesteps the CRM simulation runs. In Figure 4 the differences in runtime between mixed precision and the regular solver are depicted. Both mesh sizes and and the hybrid and prism versions of them seem to qualitatively behave the same way. Without mixed precision there is a clear change in scaling behavior at about the node number that we established as the L3 cache cutoff before (see Table 1), which is not present for mixed precision. Before the problem fits into the L3 cache, the mixed precision computation seems to scale better than the standard one, but when both versions fit completely, the runtime advantage of mixed precision becomes less relevant. This is expected, as the main advantage of lower precision arithmetic, less memory traffic, is mostly lost when working on data inside the cache.

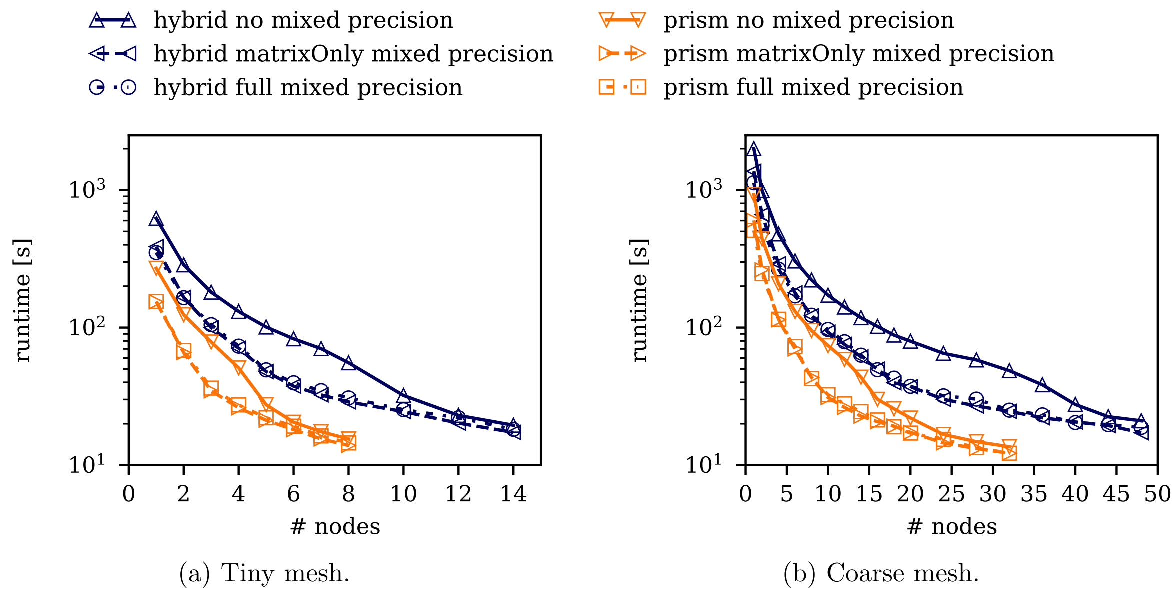
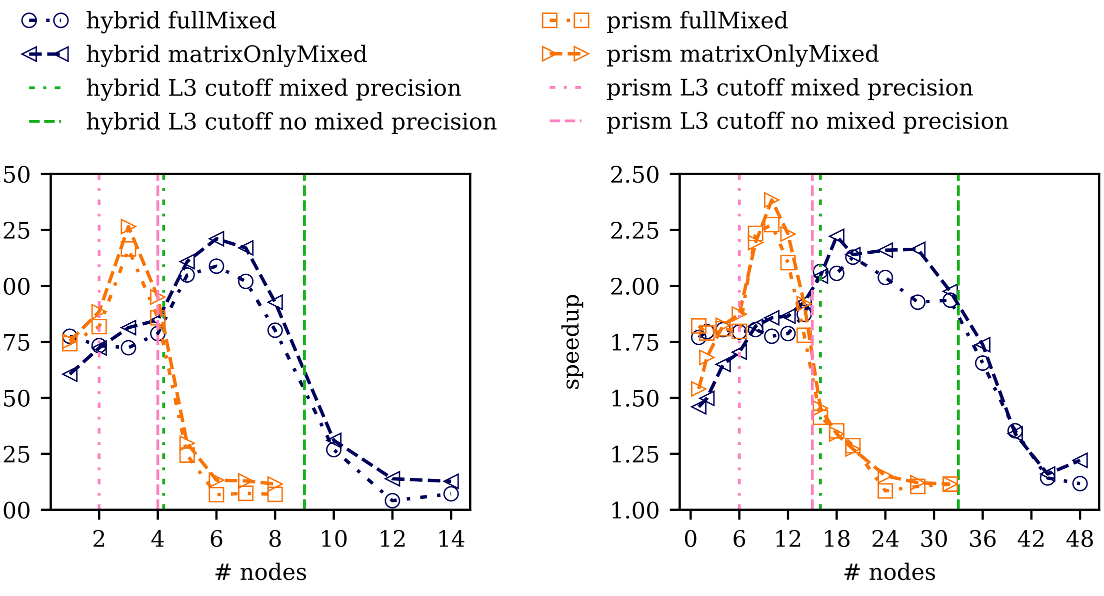


Figure 4: Runtimes for different numbers of nodes of both the tiny and the coarse mesh.

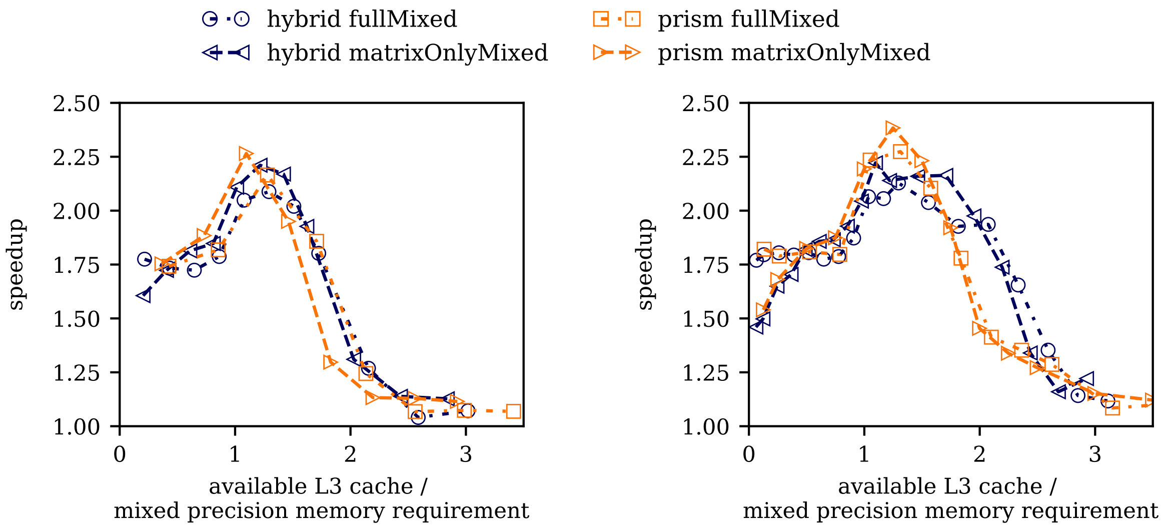
Figure 5 shows the speedup of mixed precision compared to the same problem without it. Here we clearly see the big increase in speedup, as soon as the mixed precision problem fits into the L3 cache, which in turn almost vanishes again when the regular problem also fits. But the curve never goes below 1, so even when the new bottleneck for both versions takes effect, mixed precision still achieves a speedup of 10-20 %. Also worthy to note is that in the pure main memory region to the left, full mixed precision has a stable speedup near 80 % until the cache cutoff, while matrix only mixed precision starts in the region of 40-60 % speedup and rises to reach the value of the other method at the cache cutoff.

When normalizing the graph by the available L3 cache compared to the problem size, all speedup graphs line up with each other (see Figure 6). With the single precision problem size being the reference, the over proportional speedup of mixed precision is between 1.0 and 2.0. The slightly later loss of speedup for the coarse hybrid mesh might be due to some inaccuracies in estimating the problem size, or because cache effects are usually not a clean cutoff and more gradual, especially for the L3.



(a) Tiny mesh. (b) Coarse mesh.

Figure 5: Speedups of mixed precision for different numbers of nodes of both the tiny and the coarse mesh.



(a) Tiny mesh. (b) Coarse mesh.

Figure 6: Speedups of mixed precision, normalized by the L3 cache size, for different numbers of nodes of both the tiny and the coarse mesh.

5 Conclusion

Mixed precision has become the default for CODA users, as it always achieves a speedup with little to no drawbacks. For industry relevant cases specifically, which mostly reside in main memory and do not fit into the L3 cache, we achieve speedup values of up to 80 %. Though many cases do not spend as much time in the linear solver as the CRM test case shown here, users still report speedup values near 30 %. As full mixed precision seems to perform better when having a big linear system compared to the available cache, it is the recommended mode for most use cases. Other software tools that use Spliss as their linear solver sometimes require more accuracy than full mixed precision can provide. Those then converge well in matrix only mode, so both implementations have their place.

Going forward we still want to make some more experiments with more different solver combinations to analyze how they react to mixed precision. Also with the rather new GPU support of Spliss it would also be interesting to incorporate FP16 floating point values for mixed precision and see if it will provide even more speedup and can still produces adequate results for industrial CFD applications.

Acknowledgments

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".

REFERENCES

[1] H. Anzt, V. Heuveline, B. Rocker, M. Castillo, J. C. Fernández, R. Mayo, and E. S. Quintana-Orti. Power consumption of mixed precision in the iterative solution of sparse linear systems. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 829–836, 2011.

[2] F. Brogi, S. Bnà, G. Boga, G. Amati, T. Esposti Ongaro, and M. Cerminara. On floating point precision in computational fluid dynamics using OpenFOAM. *Future Generation Computer Systems*, 152:1–16, 2024.

[3] G. Freytag, J. V. F. Lima, P. Rech, and P. O. A. Navaux. Impact of reduced and mixed-precision on the efficiency of a multi-GPU platform on CFD applications. In O. Gervasi, B. Murgante, S. Misra, A. M. A. C. Rocha, and C. Garau, editors, *Computational Science and Its Applications – ICCSA 2022 Workshops*, pages 570–587, Cham, 2022. Springer International Publishing.

[4] I. Huismann, S. Fechter, and T. Leicht. HyperCODA – extension of flow solver CODA towards hypersonic flows. In A. Dillmann, G. Heller, E. Krämer, and C. Wagner, editors, *New Results in Numerical and Experimental Fluid Mechanics XIII*, pages 99–109, Cham, 2021. Springer International Publishing.

[5] I. Huismann, J. Gericke, and R. Tschüter. Accelerating the FlowSimulator: Performance analysis & optimization of DLR high-performance solvers using Score-P. In *14th International Parallel Tools Workshop (IPTW)*, in press.

[6] T. Ina, Y. Idomura, T. Imamura, S. Yamashita, and N. Onodera. Iterative methods with mixed-precision preconditioning for ill-conditioned linear systems in multiphase cfd simu-

lations. In *2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 1–8, 2021.

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

[8] T. Leicht, J. Jägersküpper, D. Vollmer, A. Schwöppe, R. Hartmann, J. Fiedler, and T. Schlauch. DLR-project Digital-X – next generation CFD solver 'Flucs'. 2016.

[9] NASA. Common Research Model. `http://commonresearchmodel.larc.nasa.gov`, 2012.

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

[11] J. P. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. J. Mavriplis. CFD vision 2030 study: A path to revolutionary computational aerosciences. Technical report, NASA, 2014.

[12] A. Walden, E. Nielsen, B. Diskin, and M. Zubair. A mixed precision multicolor point-implicit solver for unstructured grids on GPUs. In *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 23–30, 2019.