ParCFD2024
35th International Conference on Parallel Computational Fluid Dynamics
Sep. 02-04, 2024, Bonn, Germany

Portable Linear Solvers for High-Order Spectral Element Methods on GPUs

Yu-Hsiang M. Tsai[a], Gregor Olenik[a,b], Andreas Herten[c], Mathis Bode[c] and Hartwig Anzt[a,*]

[a]*Technical University of Munich (TUM), School of Computation, Information and Technology, Bildungscampus 2, 74076 Heilbronn, Germany*
[b]*Karlsruhe Institute of Technology (KIT), Scientific Computing Center (SCC), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany*
[c]*Forschungszentrum Jülich GmbH, Jülich Supercomputing Centre (JSC), Wilhelm-Johnen-Straße, 52428 Jülich, Germany*

ARTICLE INFO[†]

Keywords:
Spectral Element Methods;
Graphics Processing Unit Offloading;
Linear Solver;
Platform Portability

ABSTRACT

The diversification in hardware architectures has become a challenge for computational science: software stacks implemented for a specific hardware architecture often fail to port to other systems. To counter this problem, simulation software stacks increasingly rely on portability layers or software stacks that feature backends for different hardware architectures. We present Ginkgo, a math library that takes platform portability as a central design principle and can be used for numerical calculations in the nekRS CFD code. A runtime and scalability analysis for CFD applications demonstrates that Ginkgo enables platform portability at high performance and scalability.

1. Introduction

In the latest TOP500 list[1] ranking the fastest supercomputers, only one of the fastest ten systems does not feature GPUs. But while the trend incorporating GPU accelerators into the HPC systems is universal, the community still struggles to agree on a universal programming language to implement software for the GPUs of the different vendors. Instead, each vendor supports their own programming ecosystem, often hindering or even blocking portability of software stacks. At the same time, it is impossible for scientific software stacks to develop and maintain multiple versions targeting different hardware architectures. Simulation software stacks currently implement two strategies to tackle the hardware diversification: the use of a portability layer and the use of portable software components. The use of a portability layer requires the rewrite of the simulation software in a portable programming language that can then be compiled for different hardware architectures. Examples for portability layers are the SYCL abstraction[2], Kokkos[3], RAJA[4], and OCCA[5], cf. [1]. Using a portable language enables execution on the architectures that support the portability layer. A disadvantage is that a portability layer has to trade specialization against generalization, and thus can never exploit the latest features of a specific hardware that are not available on other hardware. Hence, a portability layer typically pays some performance penalty to enable the execution on a wide range of hardware. The second approach is to utilize software components that are more limited in functionality, i.e., can not be used for the complete simulation code, but provide high performance on a range of architectures. A viable strategy is to identify the computationally most expensive building blocks in the simulation code, e.g., the numerical computations, and rely on specialized platform-portable libraries for these building blocks. Obviously, the two strategies can also be used in combination: a portability layer enables the execution on a wide range of hardware, a dedicated library ensures high performance for the most expensive building blocks of the simulation code. We here present how the Ginkgo math library can be used to accelerate computational fluid simulations on GPU-accelerated systems. In particular, we demonstrate that the nekRS CFD solver can benefit from using Ginkgo for solving the underlying linear equations by new solver options and platform portability.

[†]This paper is part of the ParCFD 2024 Proceedings. A recording of the presentation is available on YouTube. The DOI of this document is 10.34734/FZJ-2025-02503 and of the Proceedings 10.34734/FZJ-2025-02175.

*Corresponding author
yu-hsiang.tsai@tum.de (Y.M. Tsai); gregor.olenik@tum.de (G. Olenik); a.herten@fz-juelich.de (A. Herten); m.bode@fz-juelich.de (M. Bode); hartwig.anzt@tum.de (H. Anzt)
ORCID(s): 0000-0001-5229-3739 (Y.M. Tsai); 0000-0002-0128-3933 (G. Olenik); 0000-0002-7150-2505 (A. Herten); 0000-0001-9922-9742 (M. Bode); 0000-0003-2177-952X (H. Anzt)

[1]Top500 as of 06/2024 https://www.top500.org/lists/top500/2024/06/

[2]SYCL https://www.khronos.org/sycl/
[3]Kokkos https://kokkos.github.io
[4]RAJA https://raja.readthedocs.io
[5]OCCA https://github.com/libocca/occa

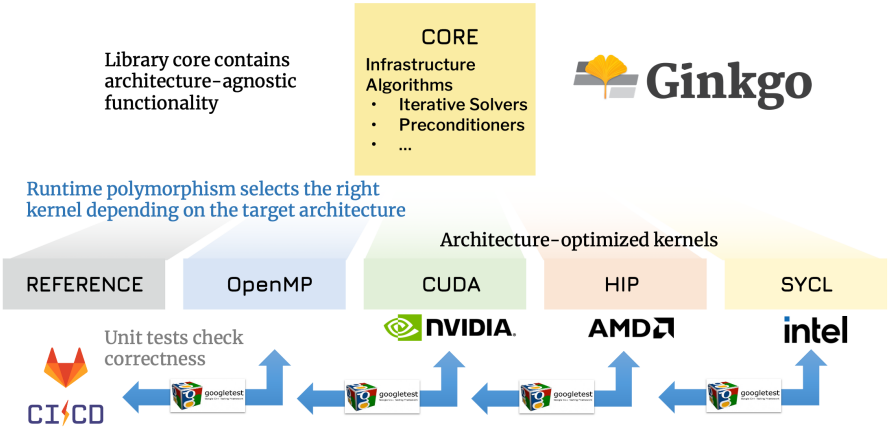


Figure 1: Overview of the Ginkgo library design using the backend model for platform portability [2].

Listing 1: Using Ginkgo as the coarse solver and set the ginkgo solver from the config.json file.

```
[PRESSURE]
preconditioner = multigrid
coarseSolver = ginkgo
[GINKGO]
configFile = config.json
```

Listing 2: Configuration file for selecting the numerical methods and configuring the parameters.

```
{
  "type": "solver::Cg",
  "criteria": [
    {"type": "Iteration", "max_iters": 4},
    {"type": "ImplicitResidualNorm",
     "reduction_factor": 1e-4,
     "baseline": "initial_resnorm"}
    ]
}
```

2. The nekRS and Ginkgo software packages

The nekRS framework[6] [3] is a CFD solver based on libParanumal[7] [4] and Nek5000[8]. It is implemented in C and C++ including Fortran bindings for Nek5000 and uses an MPI + OCCA approach for parallelization, where OCCA acts like a language translator without overhead associated to it during runtime. Numerically, it relies on the spectral element method (SEM), which makes it well suited for the efficient simulation of turbulence, where the number of grid points grows faster than quadratically with the REYNOLDS number when all flow features must be resolved.

Ginkgo is a software library developed under the US Exascale Computing Project (ECP) that focuses on the efficient handling of sparse linear systems on GPUs. Ginkgo is implemented in modern C++ to accommodate a large number of scientific application codes. The software features multiple backends in hardware-native languages: CUDA for NVIDIA GPUs, HIP for AMD GPUs, and SYCL for Intel GPUs. Ginkgo contains a set of iterative solvers, including Krylov solvers, algebraic multigrid (AMG), and parallel preconditioners that serve as a valuable toolbox for application codes. Ginkgo aims to provide not only functional portability but also performance portability [2]. To achieve this, Ginkgo uses a backend model that lifts portability to the software design level. The idea is to rely on a set of kernels implemented using vendor-native programming models, for each hardware target [2]. These kernels are then used to implement the high-level algorithms. This is reflected in Fig. 1 visualizing the backend model used in the Ginkgo library design[9].

Ginkgo provides a wide range of numerical methods for the solution of sparse linear systems, see [2] for an overview of functionality supported by Ginkgo on different hardware. While there are some solvers and preconditioners known to provide good performance for a wide range of nekRS simulations, it is of interest to test a large variety of methods and configuration parameters to identify the best method for a specific simulation. To enable the easy and fast analysis of a wide range of numerical methods, we enabled the use of configuration files that use JSON to encode solver and parameter configurations, The example configuration in Lst. 1

[6] nekRS https://github.com/Nek5000/nekRS
[7] libParanumal https://github.com/paranumal/libparanumal
[8] Nek5000 https://github.com/Nek5000/Nek5000

[9] Ginkgo uses SYCL as one of its backends, which is a portability layer in itself.

Solver	CG(4 iter)	BiCGstab(4 iter)	BiCGstab(20 iter)	GMRES(20 iter)
#coarseSolver	44	41	37	38
runtime[s]	2.03469e-01	3.02990e-01	3.80737e-01	6.32582e-01

Table 1
Performance of different solver settings of GABLS test case with 32 elements per each axis.

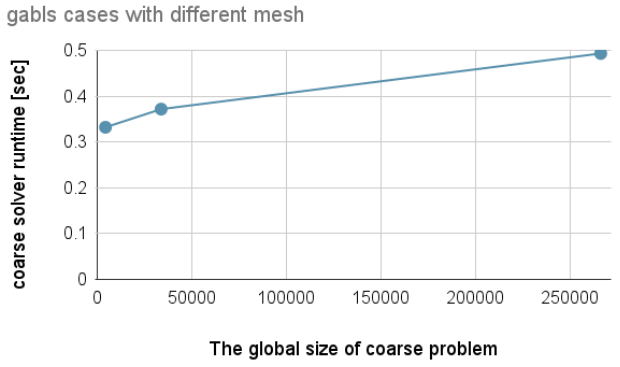
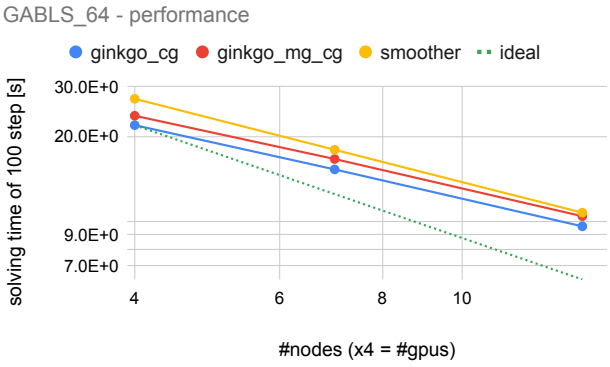
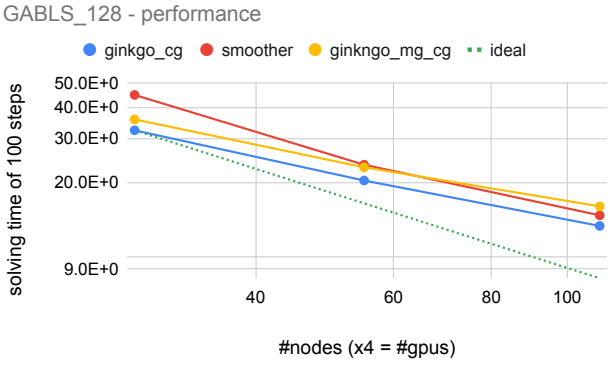


Figure 2: Runtime of the coarse solver on the GABLS test case using different discretizations (16, 32, and 64 elements in each dimension) on 2 machine nodes (8 A100 GPUs/MPI nodes).



(a) The performance of GABLS with 64 elements in each dimension on 4, 7, and 14 physical nodes.



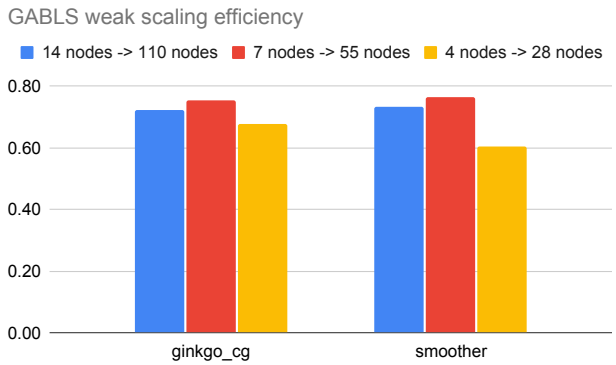
(b) The performance of GABLS with 128 elements in each dimension on 28, 55, and 110 physical nodes.

Figure 3: Performance of GABL: solving time.

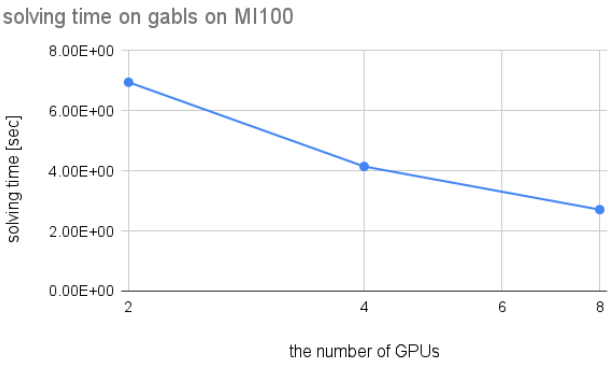
enables ginkgo in nekRS with the configuration file. This concept avoids any compilation of the code, but exclusively works with already compiled libraries. An example for a configuration file is given in Lst. 2. We can try different kinds of solver like CG, BiCGstab, and GMRES or use more iterations in Tab. 1.

3. Preliminary experiments on integration

For testing the correctness, performance and scalability of the integration of the Ginkgo backend, we focus on the GABLS test case that was established by the atmospheric boundary layer community and is an acronym for the GEWEX (Global Energy and Water Cycle Experiment) Atmospheric Boundary Layer Study (GABLS). It is used to quantify the effects of numerical modeling and discretization choices. We initially fix the computational resources to 8 NVIDIA A100 GPUs in two nodes of the Jureca supercomputer and vary the discretization of the unit cube from 16^3 to 64^3. For the different discretizations, we visualize in Fig. 2 the Ginkgo coarse grid solver runtime used in a four level multigrid. We observe an only mild increase in the coarse grid solver runtime for the increasing coarse grid problem sizes. We next investigate the strong and weak scalability of the nekRS simulation using Ginkgo for the numerical computations. We analyze the solving time from 3,000-th time step to 3,100-th time step of the nekRS simulation using the 64^3 discretization on 4, 7, and 14 physical nodes (16, 28, and 56 GPUs respectively) in Fig. 3a and 128^3 discretization on 28, 55, and 110 physical nodes (112, 220, and 440 GPUs respectively) in Fig. 3b with Ginkgo's CG, algebraic Multigrid with CG as coarse solver, and OCCA smoother. In these two cases, CG from ginkgo can get 1.1x speedup against smoother from nekRS. We choose the number of nodes in Fig. 3a and Fig. 3b such that the averaged local sizes are the same between two cases and we can have weak scalability plot in Fig. 4a. We focus the weak scaling efficiency between Ginkgo CG and nekRS smoother in Fig. 4a. Ginkgo CG only performs slightly better weak scalability from GABLS (64^3) on 4 nodes to GABLS (128^3) on 28 nodes than nekRS



(a) The weak scaling efficiency of Ginkgo CG and nekRS smoother between GABLS with 64 and 128 elements in each dimension. We choose the number of nodes between the two cases to make the averaged local size the same.



(b) Solving time of GABLS (16 elements in each dimension) on 2, 4, and 8 AMD MI100 in one machine. It is our self-host machine, so the performance and scaling behavior may be changed in the supercomputer.

Figure 4: Performance of GABLS: weak scaling and solving time on AMD MI100.

smoother, but other cases are quite similar. This preliminary experiments shows that the new ginkgo integration keeps the scalability of nekRS and brings the new solver options to nekRS. Moreover, we run the GABLS with 16 elements in each dimensions on 2, 4, and 8 AMD MI100 GPUs to show the portability of Ginkgo with nekRS in Fig. 4b.

Acknowledgements

This research is supported by the Inno4Scale project under Inno4scale-202301-099. Inno4Scale has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101118139. The JU receives support from the European Union's Horizon Europe Programme. The authors acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputers at Jülich Supercomputing Centre (JSC).

References

[1] W. F. Godoy, P. Valero-Lara, T. E. Dettling, C. Trefftz, I. Jorquera, T. Sheehy, R. G. Miller, M. Gonzalez-Tallada, J. S. Vetter, V. Churavy, Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes (2023). doi:10.48550/arXiv.2303.06195.

[2] T. Cojean, Y.-H. M. Tsai, H. Anzt, Ginkgo—A math library designed for platform portability, Parallel Computing 111 (2022) 102902. doi:10.1016/j.parco.2022.102902.

[3] P. Fischer, S. Kerkemeier, M. Min, Y.-H. Lan, M. Phillips, T. Rathnayake, E. Merzari, A. Tomboulides, A. Karakus, N. Chalmers, T. Warburton, NekRS, a GPU-accelerated spectral element Navier–Stokes solver, Parallel Computing 114 (2022) 102982. doi:10.1016/j.parco.2022.102982.

[4] N. Chalmers, A. Karakus, A. P. Austin, K. Swirydowicz, T. Warburton, libParanumal: a performance portable high-order finite element library, release 0.5.0 (2022). doi:10.5281/zenodo.4004744.