Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 267 (2025) 72–81

Procedia Computer Science

www.elsevier.com/locate/procedia

Proceedings of the Third EuroHPC user day

Enabling Ginkgo as Numerics Backend in nekRS Employing A Loosely-Coupled Configuration File Concept

Yu-Hsiang Mike Tsai[a], Mathis Bode[b], Hartwig Anzt[a,c,*]

[a]*Technical University of Munich, Heilbronn, Germany*
[b]*Forschungszentrum Jülich, Germany*
[c]*The Innovative Computing Laboratory, The University of Tennessee, USA*

Abstract

In computational fluid dynamics (CFD), the choice of numerical methods can significantly impact the overall simulation runtime. While it is virtually impossible to know the optimal solver plus preconditioner configuration for every hardware and application setup, it is valuable for CFD engineers to have access to and evaluate different numerical methods to customize the setup for efficient execution. In this paper, we demonstrate how the Ginkgo high-performance numerical linear algebra library is integrated as a math toolbox into the nekRS state-of-the-art computational fluid dynamics simulation library to give CFD engineers access to a plethora of solvers and preconditioners CFD engineers. Using three application test cases, we demonstrate how picking numerical methods from the Ginkgo library can accelerate simulations on supercomputers featuring NVIDIA's Ampere GPUs and Grace Hopper superchips.

Keywords: Spectral Element Methods; GPU; linear solver

1. Introduction

Computational fluid dynamics (CFD) simulations drive a large portion of high-performance research and development. Based on a discretization of the Navier-Stokes equations, CFD simulations attempt to simulate the free-stream flow and the interaction between the flows and gases with applications ranging from airplanes, reactors and combustion to ocean and atmospheric flows as well as astrophysical problems. CFD simulations resolving all relevant physical scales are called direct numerical simulations (DNSs), but they are not possible for all applications due to computational cost. Reduced order methods (ROMs), such as Reynolds-averaged Navier Stokes (RANS) and large-eddy simulation (LES), can be used as cheaper alternatives. However, their accuracy strongly depends on the availability of

* Corresponding author.
E-mail address: hartwig.anzt@tum.de

1877-0509
10.1016/j.procs.2025.08.234

suitable closure models. For handling these models on supercomputers, CFD libraries need access to fast and robust numerical methods that are designed for efficient execution on modern hardware.

Virtually all modern supercomputers are GPU-centric. That means that a significant portion of the computing performance is provided by the GPUs of a system. For example, on the latest TOP500 list ranking the supercomputers according to their HPL performance on FP64 precision, only one of the top ten systems is not GPU-centric. Although the performance and memory bandwidth of GPUs have both grown quickly in recent years driven by applications such as artificial intelligence, the communication between host memory and GPUs' memory is still often a performance bottleneck. Thus, realizing all numerical computations of a CFD simulation on the GPUs of a system without any data movement between the GPUs and the CPUs has become the dominant approach in high-performance computing.

In this paper, we integrate the high-performance numerical linear algebra library Ginkgo as a math toolbox into the state-of-the-art CFD library nekRS. Ginkgo provides a plethora of performance-portable direct and iterative linear solvers, preconditioners, and algebraic multigrid (AMG) that are efficient in the solution of problems arising in LES and RANS turbulence models. After providing some background about the nekRS and Ginkgo libraries and their usage in Section 2, we detail in Section 3 how configuration files can be used to instruct nekRS to employ a specific solver + preconditioner combination from the Ginkgo. We then demonstrate in Section 4 that employing Ginkgo functionality in nekRS simulations can indeed render attractive runtime savings on GPU-centric supercomputers. In Section 5, we summarize the results and how the approach taken in the Ginkgo integration transfers to other simulation software stacks.

2. Background

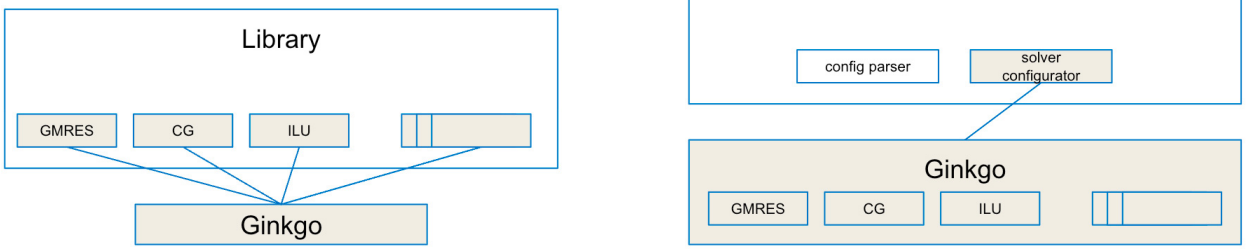
nekRS[15, 6] is a CFD library based on libparanumal[10, 3] and Nek5000[14]. It mainly uses OCCA[11] as a portability layer to support different vendors' hardware. nekRS starts from the spectral element method (SEM), which combines high accuracy with flexibility concerning flow geometry. A high-order SEM approximates the solution and data in terms of locally structured Nth-order tensor product polynomials on a set of globally unstructured elements. Thus, in addition to exponential convergence for smooth solutions with increasing polynomial order, which provides high accuracy at lower computational cost, it offers the flexibility to handle complex geometries via domain decomposition. As a result, nekRS is 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.

nekRS highly optimizes the components for their simulation. They especially apply their domain knowledge to accelerate the solving process based on the problems' properties. nekRS usually uses p-multigrid as a preconditioner, which uses different approximate orders on the problem for different multigrid levels. In multigrid, nekRS needs a solver to solve the coarse problem of p-multigrid. Previously, nekRS used HYPRE[5, 17] and NVIDIA AmgX[13] as the coarse solver options. This paper introduces Ginkgo as another coarse solver option, which brings more solvers into the nekRS stack.

Ginkgo[1] is a software library focusing on the efficient solution of sparse linear systems on GPU-centric supercomputers. The software features multiple backends in hardware-native languages: CUDA for NVIDIA GPUs, HIP for AMD GPUs, and SYCL for Intel GPUs in [4]. Also, Ginkgo considers software sustainability to be of high priority, so all functionality is unit-tested against a sequential reference implementation and employs continuous integration (CI) to ensure the correctness of compilation and execution. Ginkgo contains a plethora of iterative solvers, including Krylov solvers, direct solvers, algebraic multigrid (AMG), and parallel preconditioners like block-Jacobi, incomplete LU factorization (ILU) and parallel variants (ParILU), ILU with thresholding, BDDC preconditioning, and batched routines for efficient data-parallel processing.

3. Leveraging Ginkgo in nekRS

Integrating Ginkgo as a math toolbox into a simulation software stack is one thing, but easing the use of Ginkgo's functionality in simulation runs is a totally different aspect that is equally important to facilitate the hassle-free toolbox usage for domain scientists who are primarily focused on the CFD applications. To facilitate the hassle-free usage of Ginkgo's functionality, we developed a configuration file concept that allows encoding all solver and preconditioner



(a) Library needs to expose the numerical methods available in the math toolbox, then configure the solver in the application code.

(b) A single interface to the toolbox is complemented with a configuration file that encodes the numerical method selection outside the source code.

Fig. 1: Math toolbox integration options.

Listing 1: CG solver from 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}
  ]
}
```

settings in a file. Prior to introducing this concept, the simulation software needed to expose all of Ginkgo's solver and preconditioner options internally to the users, see Figure 1a. The configuration file concept removes this burden by outsourcing all solver and preconditioner configurations into a file. The simulation software only needs to include one interface to the math toolbox (in this case: Ginkgo) and extracts the specific numerical method configuration from the configuration file, see Figure 1b. This concept has several advantages: 1) It dramatically eases the maintenance of a library interface as new methods do not need to be exposed in the simulation software stack but simply become available as an option in the file configuration, and 2) it eases the toolbox usage for the domain scientists as they can easily swap solvers and preconditioners and options in the configuration file without even touching the code or re-compilation.

To provide an example for the configuration file usage, we configure the JSON in Listing 1 to select a Conjugate Gradient (CG) solver with an iteration limit of four iterations and a relative residual stopping criterion of 10^{-4}. Not significantly more complicated is the configuration of an AMG solver based on Parallel Graph Match (PGM[13]) as the coarsening method and a CG coarse grid solver for the bottom-level system of size smaller than 512 in Listing 2.

The nekRS domain scientist instructs nekRS to use Ginkgo by modifying the nekRS run file, see Listing 3: Specifying `ginkgo` in `coarseSolver` instructs nekRS to the generic interface to Ginkgo in the coarse solver of the nekRS multigrid, the path specified in the `[GINKGO]` section of the runfile provides the path to a Ginkgo configuration file like Listing 2. Domain scientists can quickly change the Ginkgo solver configuration by modifying the Ginkgo configuration file or changing the path to the configuration file in the nekRS run file.

4. Experiments Result

With Ginkgo being integrated into nekRS and the configuration file enabling the hassle-free use of Ginkgo's numerical methods in nekRS, the remaining question is whether domain scientists can benefit from utilizing Ginkgo functionality in nekRS simulations. To demonstrate the practical benefits the Ginkgo toolbox brings to nekRS, we focus on three scientifically relevant benchmark applications and their solution on the latest GPU-centric clusters at the Jülich Supercomputing Centre (JSC). The two supercomputers we run on are the JURECA-DC GPU system, which

Listing 2: AMG solver from configuration file for selecting the numerical methods and configuring the parameters.

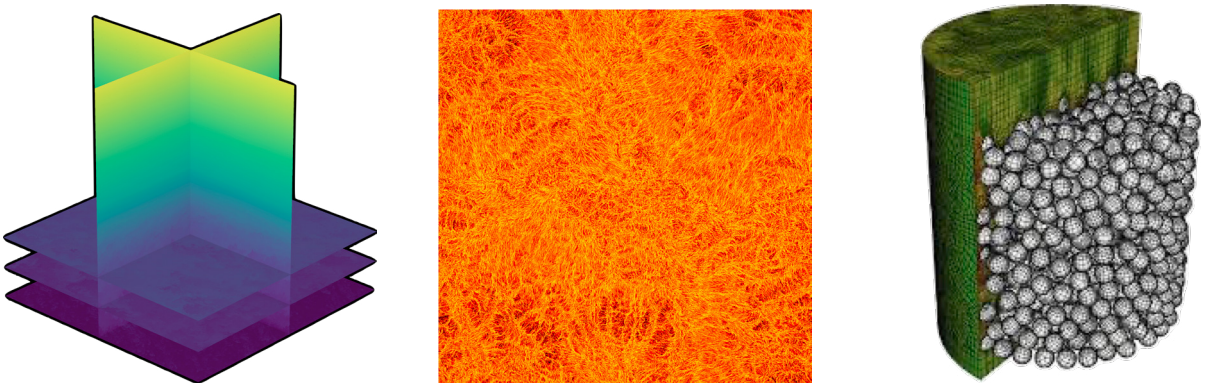
```
{
  "type": "solver::Multigrid",
  "min_coarse_rows": 512,
  "mg_level": [
    {
    "type": "multigrid::Pgm", "deterministic": true
    }
    ],
  "coarsest_solver": {
    "type": "solver::Cg",
    "criteria": [
      {"type": "Iteration", "max_iters": 4},
      {"type": "ImplicitResidualNorm",
       "reduction_factor": 1e-4}
    ]
  },
  "criteria": [
        {"type": "Iteration", "max_iters": 1}
  ]
}
```

Listing 3: Using Ginkgo as the coarse solver and set the ginkgo solver from the file.

```
[PRESSURE]
...
preconditioner = multigrid
coarseSolver = ginkgo
...
[GINKGO]
configFile = <path to cg.json or mg.json>
```

is based on 4 A100 GPUs hosted by 2 AMD EPYC 7742 CPUs per node, and the JEDI system, which is composed of 4 Grace Hopper superchips per node. Each Superchip integrates a 72-core Grace CPU and a Hopper GPU (96 GB HBM3) connected via NVLink-C2C (900 GB/s). JEDI is the development system for the first European Exascale supercomputer JUPITER. In fact, it features one future JUPITER rack of hardware. We compile the nekRS and Ginkgo codes with GCC 12.3.0, CUDA 12.2, and OpenMPI 4.1.6 with GPU-aware features on these machines.

Different CFD problems can be numerically very different. E.g. the pressure solve, which is usually the most critical part, can be simpler or more difficult to solve from different problems. The benchmark applications are 1) GABLS in Figure 2a - GEWEX Atmospheric Boundary Layer Study[8, 7] focusing on boundary layer turbulence and near surface process on the stable boundary layers over ground and under clear skies; 2) RBC in Figure 2b - Rayleigh-Bénard convection[16, 2] focusing on the natural convection in the container when heating the container; and 3) PB in Figure 2c - Pebble Bed[9] focusing on the flow past non-contacting, randomly-packed, equally sized spheres staked in cylindrical or annular container. Also, we collect the problem size and corresponding polynomial order in Table 1. Additionally, we collect the global size of the coarse problem from nekRS's multigrid which is solved by Ginkgo in the last column of Table 1. nekRS almost distributes the problem equally to GPUs, so each GPU handles $\approx$ {the total number of elements} / N elements when we run the problem on N GPUs. Because the problem in the pressure solve can be varied in terms of difficulty, we pick Ginkgo's CG and AMG solvers with the configurations listed in Listing 1 and Listing 2, respectively for the comparison against the nekRS-internal solvers. nekRS specifics single precision for the coarse solver, so Ginkgo's solver also uses single precision to set the solver.



(a) GABLS (b) RBC (c) PB from the paper[12]

Fig. 2: The application visualization.

	polynomial order	the total number of elements	global problem size from nekRS to Ginkgo
GABLS_64	8	262,144	266,240
GABLS_128	8	2,097,152	2,113,536
RBC_40	7	2,160,000	2,182,500
RBC_80	9	2,160,000	2,182,500
ann3344	7	1,080,003	1,238,974
cyl11k	8	3,575,076	4,096,583

Table 1: The application size

4.1. GABLS cases

We report the performance of GABLS_64 (64^3 discretization) and GABLS_128 (128^3 discretization) on JURECA-DC GPU in Figure 3 and Figure 4. The node count is chosen to fulfill the weak scaling requirement of the local problem size remaining constant for both node counts. The parallel efficiency for the weak scaling experiment is visualized in Figure 5. The GABLS benchmark cases are relatively easy to solve, and Ginkgo CG outperforms both Ginkgo AMG and nekRS smoother. For the largest node count, Ginkgo CG achieves a 1.1× speedup against nekRS smoother for GABLS_64 in Figure 3 and 1.12× speedup for GABLS_128 in Figure 4. In terms of parallel efficiency, Ginkgo and nekRS are comparable, reaching about 70% parallel efficiency, see Figure 5. We collect the memory consumption per GPU from the report of nekRS smoother settings:

- GABLS_64: 17.11 GB per GPU when using 16 A100, 9.83 GB per GPU when using 28 A100, and 4.93 GB per GPU when using 56 A100.
- GABLS_128: 19.61 GB per GPU when using 112 A100, 9.97 GB per GPU when using 220 A100, and 5.02 GB per GPU when using 440 A100.

Besides the performance results on A100 on JURECA, we also perform the same experiments on JEDI to assess the performance gains coming from NVIDIA's Grace Hopper architecture. We collect the performance data of GABLS_128 on both machines in Figure 6. Overall, we see a 2× speedup when moving from the A100-powered supercomputer to the Grace Hopper superchips system. This speedup matches the experience from the other applications on JEDI. Grace Hopper superchips equip more memory than A100, so we only use a half the number of machine nodes of JURECA-DC GPU for the experiments on Grace Hopper superchips. For example, we set 14 machine nodes (56 Grace Hopper superchips) on JEDI from 28 machine nodes (112 A100) on JURECA-DC GPU. Because we set the number based on the number of machine nodes, we set 27 machine nodes by rounding down from the number $\frac{55}{2}$. We set up 28 machine nodes on JEDI, which contains 112 Grace Hopper superchips, rather than 55 machine nodes

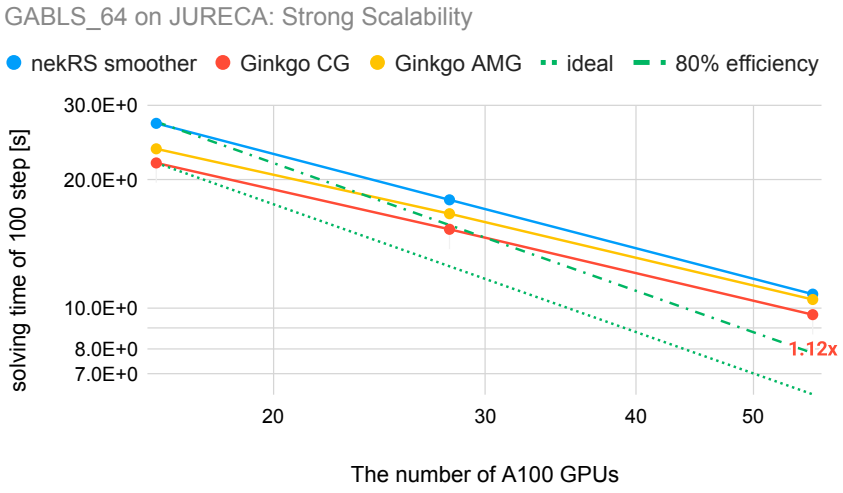


Fig. 3: The performance of GABLS_64 (64^3 discretization) on JURECA-DC GPU.

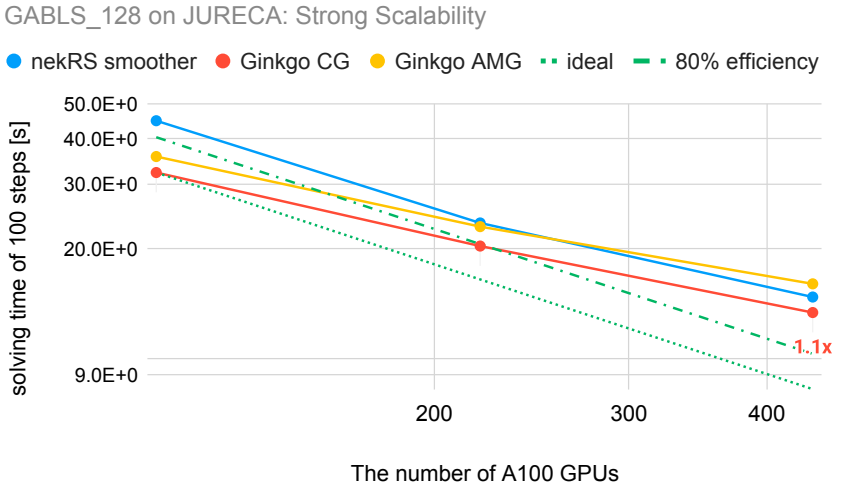


Fig. 4: The performance of GABLS_128 (128^3 discretization) on JURECA-DC GPU.

by reducing half from 110 machine nodes on JURECA-DC GPU because it is the maximum number of nodes we can request. We collect the memory consumption per GPU from the report of nekRS smoother settings:

- GABLS_128: 38.97 GB per GPU when using 56 Grace Hopper superchips, 20.30 GB per GPU when using 108 Grace Hopper superchips, and 19.61 GB per GPU when using 112 Grace Hopper superchips.

4.2. RBC cases

We next use the same hardware configurations for the RBC benchmark applications. The suffix number of RBC indicates the minimal machine node requirement on JURECA-DC GPU. RBC_40 uses polynomial order 7 and a Rayleigh number to 10^8, but RBC_80 uses polynomial order 9 and a Rayleigh number to 10^9. In this performance

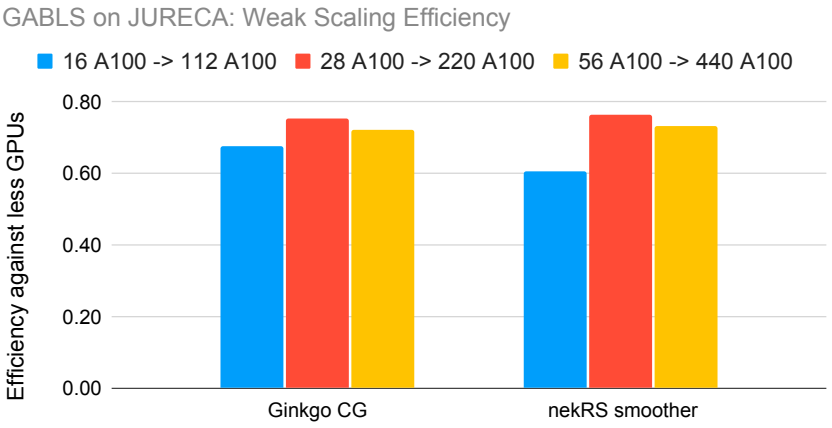


Fig. 5: The weak scaling efficiency of GABLS (from 64^3 to 128^3) on JURECA-DC GPU.

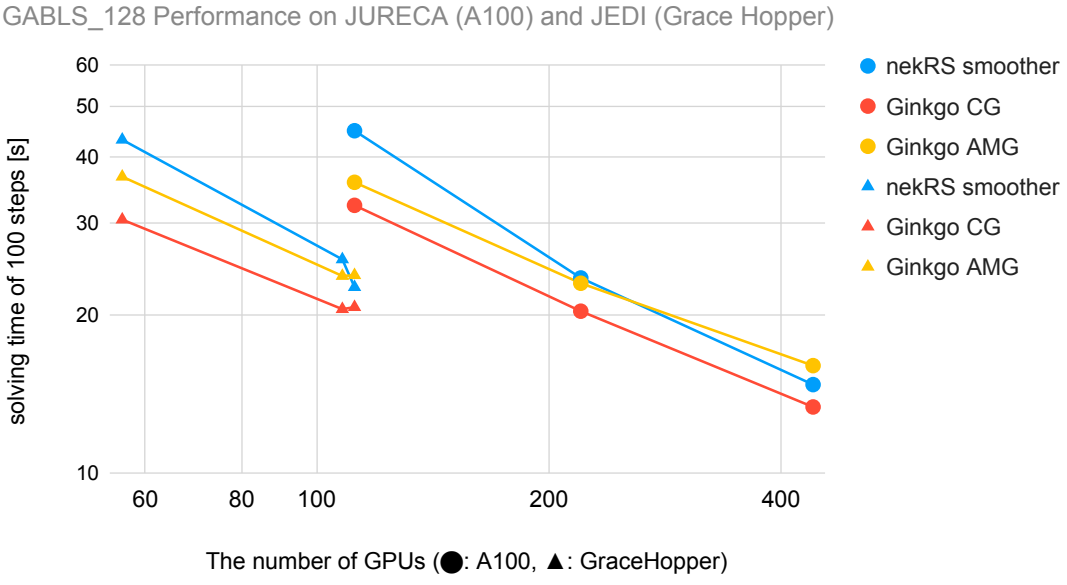


Fig. 6: The performance of GABLS_128 (128^3 discretization) on JURECA-DC GPU and JEDI.

experiment, we increment the node count by steps of 40 until we run on the full JURECA-DC GPU machine. In Figure 7 we see that all solvers perform similarly for the RBC_40 case. Ginkgo CG shows 1.15× speedup over nekRS smoother on 160 nodes (640 A100) on JURECA-DC GPU in Figure 8. For GABLS(128^3) Figure 4, nekRS smoother outperforms Ginkgo AMG for large GPU counts, but Ginkgo AMG starts outperforming nekRS smoother for large node counts for the RBC_80 benchmark application, see Figure 8. We collect the memory consumption per GPU from the report of nekRS smoother settings:

- RBC_40: 11.59 GB per GPU when using 160 A100, 5.89 GB per GPU when using 320 A100, 3.89 GB per GPU when using 480 A100, and 2.92 GB per GPU when using 640 A100.
- RBC_80: 11.34 GB per GPU when using 320 A100, 7.58 GB per GPU when using 480 A100, and 5.70 GB per GPU when using 640 A100.

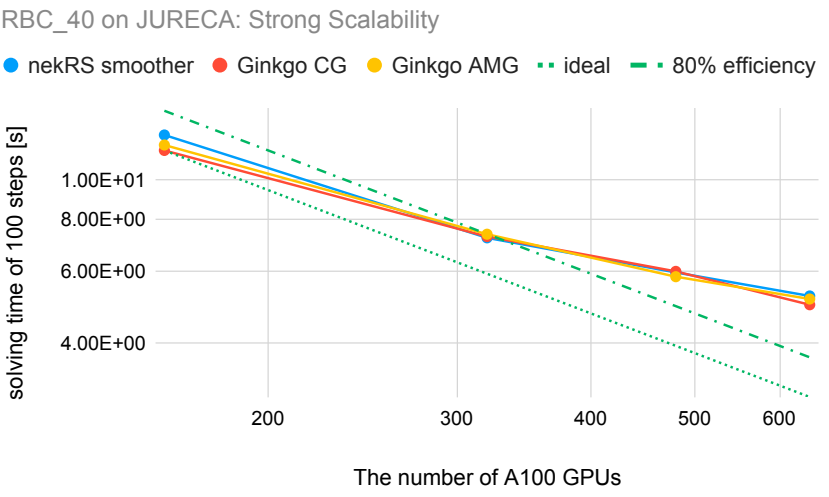


Fig. 7: The performance of RBC_40 on JURECA-DC GPU.

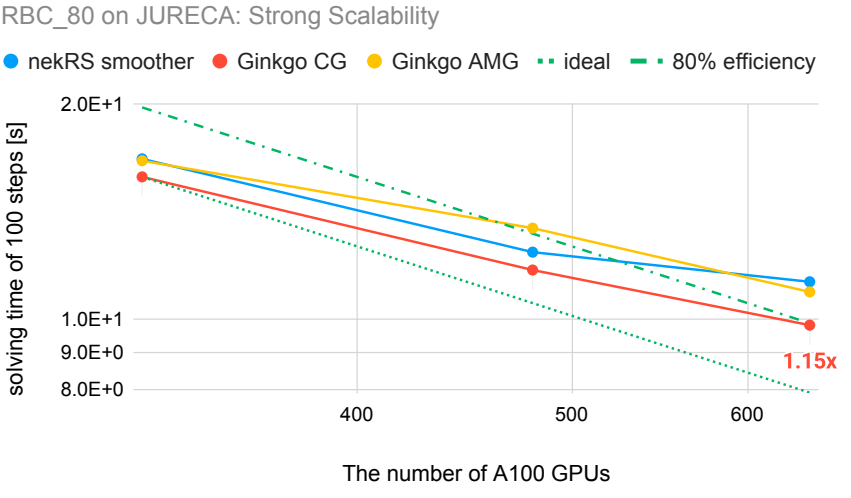


Fig. 8: The performance of RBC_80 on JURECA-DC GPU.

4.3. PB cases

We now address a numerically challenging problem coming from the PebbleBed benchmark application. The ann3344 problem contains 3,344 spheres in an annular container, and the cyl11k problem contains 11,145 spheres in a cylindrical container. The solver runtime visualization for the pebble bed series problem reveals that Ginkgo AMG largely outperforms Ginkgo CG and nekRS smoother: Ginkgo AMG outperforms nekRS smoother by 3× for the ann3344 test case and 8× for the larger cyl11k test case in Figure 9. These speedups stem from the significant convergence acceleration: the AMG reduces the iteration count by more than 8×, see Figure 10. Ginkgo CG seems superior to Gingko AMG and nekRS smoother for GABLS and RBC cases because the problems are relatively easy to solve. For PB cases, Ginkgo AMG shows clear advantages against Ginkgo CG and nekRS smoother. We collect the memory consumption per GPU from the report of nekRS smoother settings:

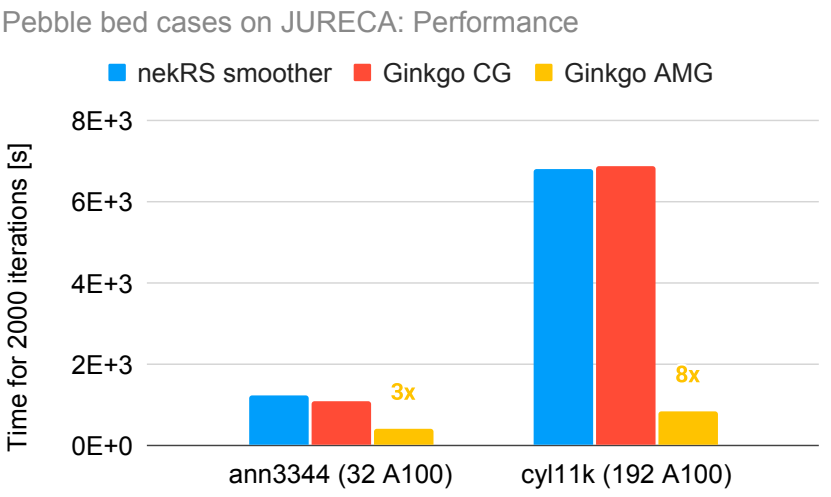


Fig. 9: The performance of pebble bed cases on JURECA-DC GPU.

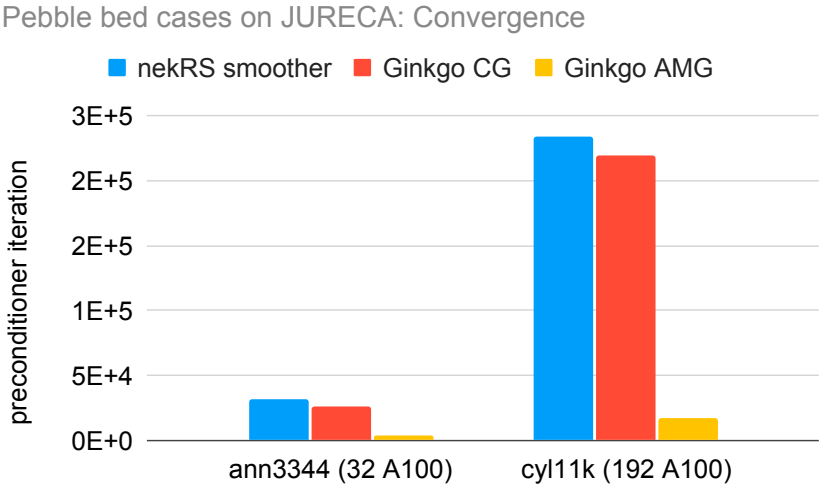


Fig. 10: The convergence of pebble bed cases on JURECA-DC GPU.

- ann3344: 36.37 GB per GPU when using 32 A100.
- cyl11k: 32.80 GB per GPU when using 192 A100.

5. Conclusion

Using the Ginkgo library as an example, we demonstrated how a lightweight integration into the nekRS library can give domain scientists easy access to a plethora of numerical methods in the Ginkgo toolbox. The use of a configuration file workflow enables quick testing of different methods and their suitability for a given application and hardware setup. We furthermore demonstrated that the CG and AMG methods available in Ginkgo through this lightweight interface can offer attractive runtime savings, particularly for numerically challenging problems. While we

do not claim that the performance we demonstrate on the latest GPU architecture from NVIDIA cannot be matched by other solvers or libraries, we are convinced that the integration strategy is a blueprint for other math toolbox integrations as it combines sustainability through low maintenance and productivity through the hassle-free usage by domain scientists.

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) and the JUPITER Research and Early Access Program (JUREAP).

References

[1] Anzt, H., Cojean, T., Flegar, G., Göbel, F., Grützmacher, T., Nayak, P., Ribizel, T., Tsai, Y.M., Quintana-Ortí, E.S., 2022. Ginkgo: A modern linear operator algebra framework for high performance computing. ACM Transactions on Mathematical Software (TOMS) 48, 1–33.

[2] Bode, M., Alvarez, D., Fischer, P., Frouzakis, C.E., Göbbert, J.H., Insley, J.A., Lan, Y.H., Mateevitsi, V.A., Min, M., Papka, M.E., et al., 2025. Deciphering boundary layer dynamics in high-rayleigh-number convection using 3360 gpus and a high-scaling in-situ workflow. arXiv preprint arXiv:2501.13240 .

[3] Chalmers, N., Karakus, A., Austin, A.P., Swirydowicz, K., Warburton, T., 2022. libParanumal: a performance portable high-order finite element library. URL: https://github.com/paranumal/libparanumal, doi:10.5281/zenodo.4004744. release 0.5.0.

[4] Cojean, T., Tsai, Y.H.M., Anzt, H., 2022. Ginkgo—a math library designed for platform portability. Parallel Computing 111, 102902.

[5] Falgout, R.D., Yang, U.M., 2002. hypre: A library of high performance preconditioners, in: International Conference on Computational Science, Springer. pp. 632–641.

[6] Fischer, P., Kerkemeier, S., Min, M., Lan, Y.H., Phillips, M., Rathnayake, T., Merzari, E., Tomboulides, A., Karakus, A., Chalmers, N., Warburton, T., 2022. Nekrs, a gpu-accelerated spectral element navier–stokes solver. Parallel Computing 114, 102982. URL: https://www.sciencedirect.com/science/article/pii/S0167819122000710, doi:https://doi.org/10.1016/j.parco.2022.102982.

[7] Holtslag, A., 2006. Gewex atmospheric boundary-layer study (gabls) on stable boundary layers. Boundary-Layer Meteorol 118, 234–246.

[8] Kosović, B., Curry, J.A., 2000. A large eddy simulation study of a quasi-steady, stably stratified atmospheric boundary layer. Journal of the atmospheric sciences 57, 1052–1068.

[9] Lan, Y.H., Fischer, P., Merzari, E., Min, M., 2021. All-hex meshing strategies for densely packed spheres. arXiv preprint arXiv:2106.00196 .

[10] libparanumal, . libparanumal. [Online] https://github.com/paranumal/libparanumal.

[11] Medina, D.S., St-Cyr, A., Warburton, T., 2014. Occa: A unified approach to multi-threading languages. arXiv preprint arXiv:1403.0968 .

[12] Merzari, E., Yuan, H., Min, M., Shaver, D., Rahaman, R., Shriwise, P., Romano, P., Talamo, A., Lan, Y.H., Gaston, D., et al., 2021. Cardinal: A lower-length-scale multiphysics simulator for pebble-bed reactors. Nuclear Technology 207, 1118–1141.

[13] Naumov, M., Arsaev, M., Castonguay, P., Cohen, J., Demouth, J., Eaton, J., Layton, S., Markovskiy, N., Reguly, I., Sakharnykh, N., et al., 2015. AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods. SIAM Journal on Scientific Computing 37, S602–S626.

[14] nek5000, . nek5000. [Online] https://github.com/Nek5000/Nek5000.

[15] nekRS, . nekRS. [Online] https://github.com/Nek5000/nekRS.

[16] Samuel, R.J., Bode, M., Scheel, J.D., Sreenivasan, K.R., Schumacher, J., 2024. No sustained mean velocity in the boundary region of plane thermal convection. Journal of Fluid Mechanics 996, A49.

[17] Yang, U.M., et al., 2002. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics 41, 155–177.