Towards Distributed Linear Solvers on GPUs using Ginkgo

Gregor Olenik, Marcel Koch, Hartwig Anzt

Author details: Gregor Olenik, Ph.D., E-Mail: gregor.olenik@kit.edu

Karlsruhe Institute of Technology, Steinbuch Centre for Computing (SCC),
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen

Introduction and Motivation

To serve the ever-increasing demand for computing resources, many HPC systems are equipped with multiple highly parallel general-purpose GPUs. In most cases, the GPUs are integrated as discrete server-type GPUs that are attached to the nodes as coprocessors and provide the lion's share of the theoretical compute performance. This concept allows to significantly increase the theoretical compute power, however, it poses a challenge to the scientific software developers that need to redesign their software such that it can benefit from the additional resources. An attractive strategy is to use the GPUs for the computationally most expensive part of an application. For computational fluid dynamics (CFD) simulations, a substantial part of the overall effort is spent on the solution of the system of linear equations arising from the discretized transport equations. This paper investigates how the currently ongoing effort within the open-source linear algebra library Ginkgo towards distributed GPU computing can be used as a backend for OpenFOAM to improve the simulation performance by offloading the linear algebra computations to GPUs.

Related Work

In the past several projects, e.g. PETSc4FOAM[1], RapidCFD[2], PARALUTION[3], foamExtend [4] addressed the need to offload OpenFOAMs linear algebra computations to GPUs. However, several of the mentioned projects did not receive significant updates in the last few years. Thus, this work focuses on using the free and open-source library Ginkgo which is under active development and supports NVIDIA GPUs, AMD GPUs, and Intel GPUs Anzt et al. (2020).

[1] `https://develop.openfoam.com/modules/external-solver`
[2] `https://github.com/Atizar/RapidCFD-dev`
[3] `https://www.paralution.com`
[4] `https://sourceforge.net/projects/foam-extend/`

Ginkgo

Ginkgo is a math library for linear algebra written in modern C++. Its main focus lies on sparse linear algebra for GPU architectures by implementing hardware-specific kernels in their native languages, i.e. CUDA (for NVIDIA GPUs), HIP (for AMD GPUs), OpenMP (for general-purpose multicore processors, such as those from Intel, AMD, or Arm), and DPC++ (for Intel GPUs). It supports a variety of high-performance linear algebra solvers, e.g. CG, BiCGStab, GMRES, and AMGx, and preconditioners like IC, ILU, ISAI, and (Block)-Jacobi, which are suitable for CFD simulations.

Backend Methodology

To leverage Ginkgo functionality in OpenFOAM, the *OpenFOAM-Ginkgo-Layer* (OGL) was implemented as a plugin for OpenFOAM. The main steps within OGL are as follows.

At first, the OpenFOAM system matrix is converted from the LDU format to a sparse matrix in CSR format in row-major order. This step yields a permutation matrix and a sparsity pattern.The permutation matrix is used to perform efficient reordering of the matrix values on the GPU device, whereas the sparsity pattern holds the indices of the sparse matrix. Both can be reused for subsequent time-step updates and are stored via smart-pointers within the object registry as *DevicePersistent* data structures to avoid costly recomputation. Furthermore, the right-hand side and the solution vector are copied to the device. While the solution vector is communicated back to OpenFOAM after the solution process for one transport equation is completed, it is also re-used as the initial guess for the next time step. Evaluation of the stopping criterion has been offloaded to the GPU device as well to avoid costly communication during the iterative solver loop.

Performance Evaluation

For a performance evaluation of the OGL interfacing the Ginkgo functionality, we choose the lidDrivenCavity3D[5] example, that was previously used by Bnaa et al. (2020) for benchmarking purposes. The presented test case uses the original, uniform, cubic grid of Bnaa et al. (2020) and evaluates results for various grid resolutions ranging from 100^3 to 500^3 while keeping the CFL number constant.

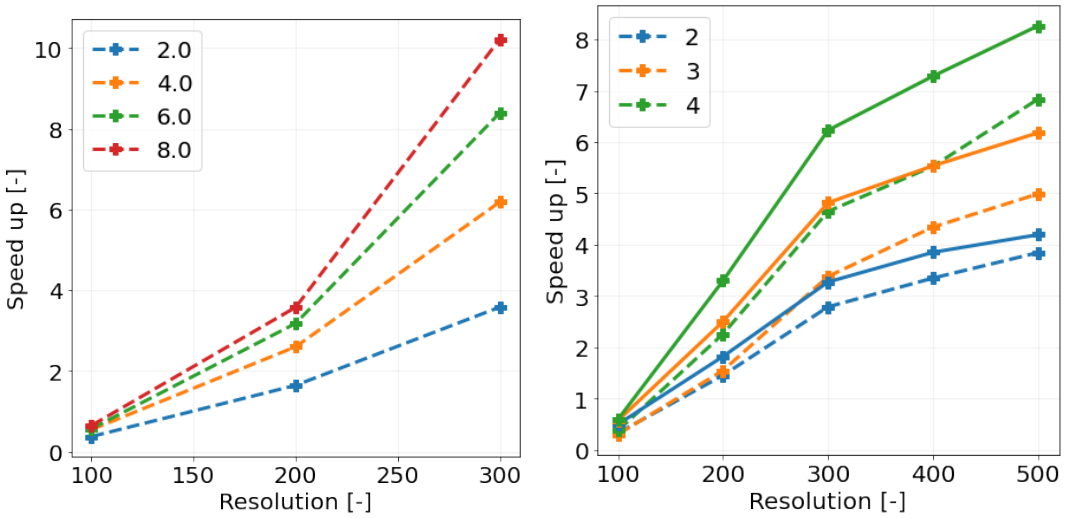
In the experimental evaluation, we use two hardware systems: For the MPI-parallel OpenFOAM execution, without GPU acceleration, and the OGL execution using Ginkgo's HIP backend, we run on a private machine composed of two AMD EPYC 7302 16 Core CPU and eight AMD MI100 GPUs. Additionally, for the OpenFOAM runs using Ginkgo's CUDA backend, we run on the HoreKa[6] cluster, which nodes are equipped with two Intel Xeon Platinum 8368 CPUs and four NVIDIA A100-40 GPUs. The MPI-parallel OpenFOAM runs without GPU acceleration employ as many MPI ranks as physical cores are available on the respective machines, i.e. 32 on the private AMD machine and 76 on the HoreKa cluster. For the OpenFOAM runs using either Ginkgo's CUDA backend or Ginkgo's HIP backend, the case is decomposed into as many subdomains as GPUs are used. The *simple*

[5] `https://develop.openfoam.com/committees/hpc.git`

[6] `https://www.scc.kit.edu/dienste/horeka.php`

Table 1: Number of subdivisions in each direction for the corresponding domain decompositons

Total number of subdomains	2	3	4	6	8	32	76
Divisions in x-dimension	2	3	2	3	2	4	19
Divisions in y-dimension	1	1	2	2	2	4	2
Divisions in z-dimension	1	1	1	2	2	2	2



(a) HIP (b) CUDA

Figure 1: Speed-up of the linear solve of pressure equation on the HoreKa system over different grid resolutions and different numbers of GPUs employed. Solid lines indicate results using GPU-direct and dashed lines transferring via host memory.

scheme was taken as a decomposition method, with the in Tab. 1 displayed number of subdomains in the respective dimensions.

Figure 1 shows the speed-up of the linear solver in comparison to the non-accelerated runs over different grid resolutions and for different numbers of GPUs. Benchmark runs with the HIP backend, shown in Fig 1(a) are conducted up to a problem size of 300^3. The speed-up increases monotonically with the problem size from approximately 0.5 for the 100^3 up to a factor of ten for the largest case on eight GPUs. Benchmark runs with the CUDA backend, shown in Fig 1(a) are conducted up to a problem size of 500^3. Additionally, on the CUDA machines, benchmark runs with (solid lines) and without (dashed lines)*NVIDIAs GPUDirect* are performed. Here again, a monotonically increasing speedup up to a factor of approximately eight for the *GPUDirect* version can be observed. Generally, using *GPUDirect* technology increases the performance by another 10-20%, since it avoids costly communication to the host by transferring data between GPUs directly.

Future Work

We have demonstrated that by using the OpenFOAM Ginkgo Layer (OGL) to interface Ginkgo's high-performance linear algebra, we can accelerate OpenFOAM applications on

multiple NVIDIA GPUs and AMD GPUs. For the lidDrivenCavity3D example, we reported speed-up factors of up to 8× over the non-accelerated execution using 76 and 32 MPI ranks, respectively. Thus, OGL allows users to speed up their single-node simulations. Future work will extend the scaling of OGL across multiple nodes.

References

Anzt, H., Cojean T., Flegar G., Göbel F., Grützmacher T., Nayak P., and Ribizel T., Tsai Y. M., and Quintana-Ortí, E. S. (2020). Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing *arXiv cs.MS/2006.16852*.

Bnà, S., Spisso, I., Olesen, M., and Rossi, G. (2020). PETSc4FOAM: A Library to plug in PETSc into the OpenFOAM *PRACE White paper*.