Optimizing an HPC LBM Application using CUDA Graphs

Milena Veneva, Jayesh Badwaik, Andreas Herten
Forschungzentrum Jülich

Introduction

With increasing focus on scalability and performance of high performance computing applications, it has become important for the simulation softwares to be able to utilize the underlying hardware as comprehensively to its maximum performance.
waLBerla [2] is a multiphysics software framework that has achieved high scalability and performance. It achieves this excellent performance due to architecture specific code generation algorithms [1] combined with efficient communication and parallel data structures like BlockForest.
In this work, we attempt to improve the GPU utilization of an Lattice-Boltzmann Method (LBM) software.

Objectives and Motivation

Objective:

Exploit CUDA Graphs to improve GPU utilization in Walberla for simulations with small block sizes.

Motivation

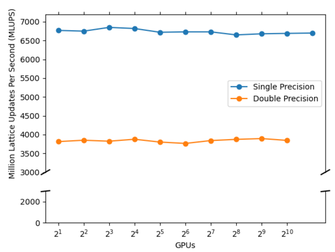

Fig. 3: Weak Scaling for Walberla on Juwels Booster

Fig. 4: Domain Partitioning in Walberla. The whole domain is divided into blocks of possibly different sizes. The blocks are then distributed among the available processes.

Walberla achieves high scalability in general. However, for regions with AMR as shown in Figure 4, small block sizes lead to suboptimal utilization of GPUs

- Small block sizes lead to smaller kernel run times
- Kernel launch times become comparable to kernel run times
- Leading to suboptimal utilization of GPUs
- CUDA Graphs
  - Define work as graph rather than single operation
  - Requires dependency graphs between kernels
- Exploit the fixed dependency graphs in WALBERLA

Method

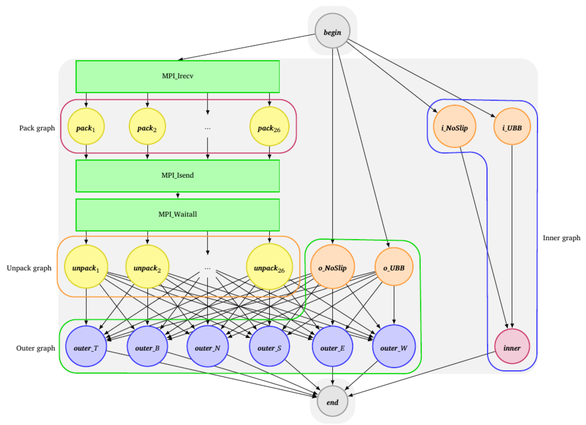

Fig. 5: Dependency Between Kernels in Walberla

(Yellow : Communication Kernels, Green: MPI Communiation API Calls, Others: Compute Kernels)

- Port UniformGridGPU Benchmark to CUDA Graphs using 4 Graphs
  - All pack kernels
  - All unpack kernels
  - Inner Domain Computation Kernels
  - Outer Domain Computation Kernels
- CUDA Graphs can only consist of GPU Kernels
  - Exclude CUDA-Aware MPI (launched from CPU) from CUDA Graphs
- Use `CUDAStreamSynchronize` to synchronize graph and non-graph components
- Empty nodes added at beginning and end to help the graph scheduler.

Results

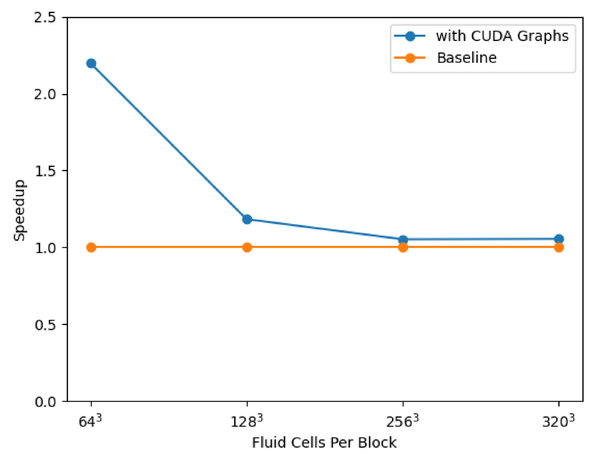

Fig. 6: Results on Juwels Booster

Conclusions

- More than 100% improvement in performance for smaller block sizes.
- Between 6% and 23% improvement for large block sizes.
- Traditional MPI Calls not integrable into CUDA Task Graphs
- Possibility of further performance benefits by using communication frameworks which can be integrated into graphs (NCCL instead of MPI).

Next Steps

- Port More Complex Use Cases to CUDA Task Graphs
- Port MPI Communications to use NCCL

References

[1] Martin Bauer, Harald Köstler, and Ulrich Rüde. “lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods”. In: *Journal of Computational Science* 49, 101269 (Feb. 2021). DOI: 10.1016/j.jocs.2020.101269.

[2] Martin Bauer et al. “waLBerla: A block-structured high-performance framework for multiphysics simulations”. In: *Computers and Mathematics with Applications* 81 (2021). Development and Application of Open-source Software for Problems with Numerical PDEs, pp. 478–501. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2020.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0898122120300146.

Acknowledgements

- This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 956000
- The JU receives support from the European Union’s Horizon 2020 research and innovation programme, and the French National Research Agency, the Federal Ministry of Education and Research of Germany, the Ministry of Education, Youth and Sports of the Czech Republic.