Case Study for Performance-Portability of Lattice Boltzmann Kernels

A comparison of Kokkos and native CUDA on ThetaGPU

Geng Liu[1], Joseph Insley[1,2], Saumil Patel[1], Silvio Rizzi[1], Victor Mateevitsi[1] & Amanda Randles[3]

[1]Argonne National Laboratory, [2]Northern Illinois University, [3]Duke University

Abstract

In this work, we study the performance-portability of offloaded lattice Boltzmann kernels and the trade-off between portability and efficiency. The study is based on a proxy application for the lattice Boltzmann method (LBM). The performance portability programming framework of Kokkos (with CUDA or SYCL backend) is used and compared with programming models of native CUDA and native SYCL. The Kokkos library supports the mainstream GPU products in the market. The performance of the code can vary with accelerating models, number of GPUs, scale of the problem, propagation patterns and architectures. Both Kokkos library and CUDA toolkit are studied on the supercomputer of ThetaGPU (Argonne Leadership Computing Facility). It is found that Kokkos (CUDA) has almost the same performance as native CUDA. The automatic data and kernel management in Kokkos may sacrifice the efficiency, but the parallelization parameters can also be tuned by Kokkos to optimize the performances.

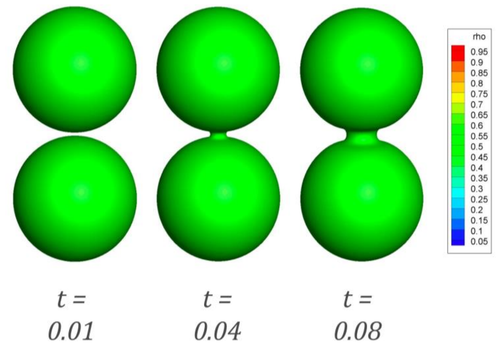

Fig. 1 (Left) Images shows the evolution (non-dimensional time) of the interfacial motion for two droplets of liquid water in coalescence in their ambient gas phase. Results from the IMEXLB-F code [2].

Motivation

- The lattice Boltzmann method (LBM) [1] is widely used in many industrial, engineering and environmental processes. (See LBM example in Fig. 1.)
- Based on the discrete velocity model (DVM) proposed by Qian et al. [3] (Fig. 3) LBM simulates the fluid by applying collision and propagation algorithms to the particle probability distribution functions (PDFs).
- The portability of the lattice Boltzmann codes needs to be considered to support different architectures. The trade-off between portability and efficiency is therefore related to whether the kernels are computationally intensive, and whether the memory accessing is indirect and complicated.

Method

- The proxy application is originally developed by John Gounley. [4]
- The Code is written with Kokkos, native CUDA and native SYCL. It also includes the LBM propagations patterns of AA, AB push and AB pull.
- The test problem is a 3D pressure driven cylinder channel flow. The geometry can be depicted by Fig. 2.
- The DVM model is D3Q19 (Fig. 3).
- The propagation patterns are explained in Fig. 4 and Fig. 5 based on a D2Q9 model.

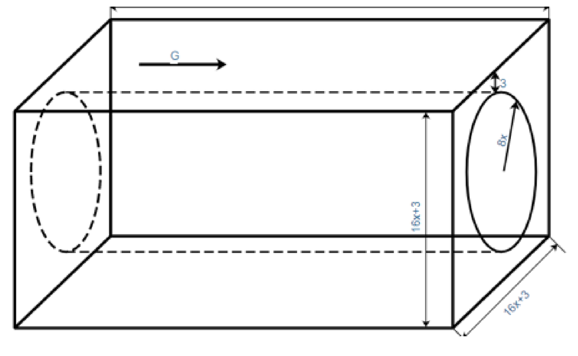

Figure 2. Geometry setup.

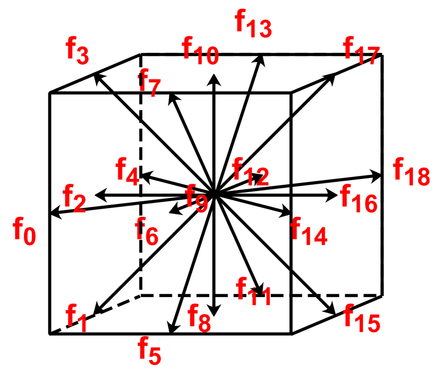

Figure 3. D3Q19 model.

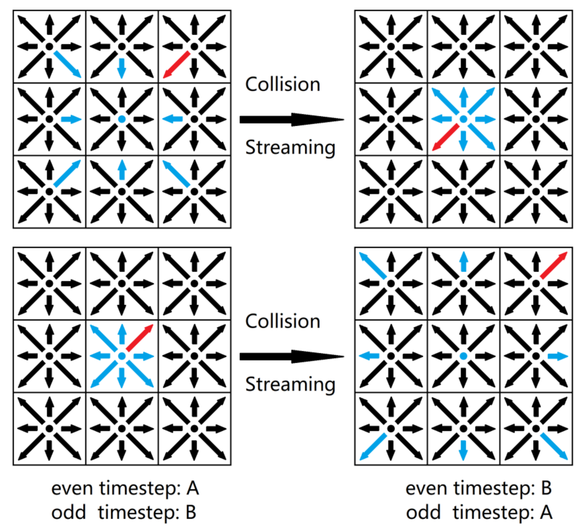

Figure 4. AB pull pattern (top) and AB push pattern (bottom).

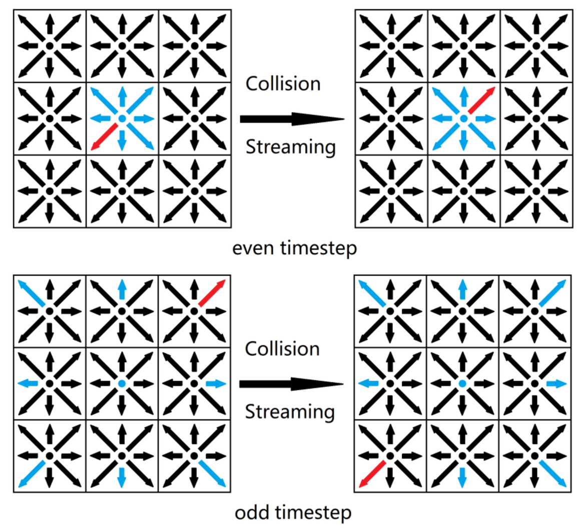

Figure 5. AA pattern even step (top) and odd step (bottom).

Kokkos

- Kokkos is a C++ library that aims at unifying different low-level parallel programming models such as OpenMP, CUDA, SYCL, HIP, etc. [5,6] This framework can support building cross-platform applications and is said to achieve performance-portability with a single codebase.
- The arrays managed by Kokkos are in the form of Views. The Kokkos Views can be constructed on devices by simply specifying the memory space according to selected backend. e.g. a double precision device view with CUDA backend can be allocated as:

```
Kokkos :: View < int * , Kokkos :: CudaSpace > myView = Kokkos :: View < int * , Kokkos :: CudaSpace > ("myView", mySize);
```

- Kokkos host views can be defined as mirrors of device views. Synchronization between host and devices can be done manually. e.g.

```
Kokkos :: View < int * , Kokkos :: CudaSpace >:: HostMirror myHostView;
myHostView = kokkos :: create_mirror_view ( myView );
Kokkos :: deep_copy ( myHostView, myView );
```

- Kokkos kernels are also executed in specified execution space. e.g.

```
Kokkos :: parallel_for (Kokkos :: RangePolicy < Kokkos :: Cuda :: execution_space > ( start , start + count ) , KOKKOS_LAMBDA ( const int index ){...});
```

- `Kokkos :: fence ();` blocks on completion of all outstanding asynchronous Kokkos operations.

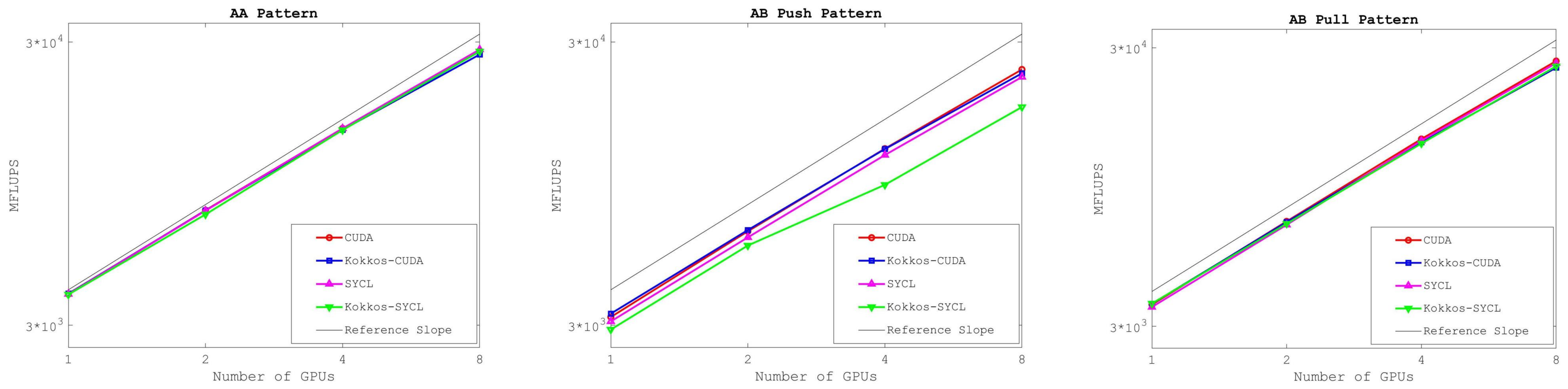

Figure 6. Performance comparison of native CUDA, native SYCL and Kokkos with CUDA/SYCL backend on ThetaGPU. Subfigures from left to right correspond to propagation patterns AA, AB push and AB pull respectively. (The reference lines represent theoretical strong scaling speedups.)

Results

The performances are evaluated by Mega Fluid Lattice Updates per Second (MFLUPS). This value m can be expressed by:

$$m = \frac{lattice\ points\ in\ the\ domain \times iterations}{computing\ time \times 10^6},$$

The performance data for different propagation patterns and different programming models are shown in Fig. 6, where the scale factor x is 16 and the implementation is on a single node (8 NVIDIA A100 GPUs with 40 GB memory space) of ThetaGPU. The factor x is chosen to make full use of the device memory.

- From the results we can see that AA pattern is in general faster than AB patterns.
- Within the same propagation pattern, the performance of Kokkos with CUDA backend, native CUDA and native SYCL are almost identical.
- The performance of Kokkos with SYCL backend has similar behavior for the AA pattern and AB pull pattern, but is not as good for AB push pattern on NVIDIA devices. A possible explanation is that the AB push pattern has a more complicated memory accessing mechanism, which makes the Kokkos optimization for SYCL backend work unexpectedly.
- The performance of Kokkos with SYCL backend for AB push pattern can be as minimal as 70% of native CUDA performance.

Next Steps

- Roofline plots and other profiling data will be collected to further explain the behaviors of the tested programming models on multiple platforms. Native SYCL and Kokkos (SYCL), Kokkos (OpenMP Target) are also going to be applied to the proxy application on non-NVIDIA devices.
- The profiling data and the comparisons will contribute to migrating stencil applications, whose algorithms are similar to LBM, to different parallel computing platforms.

Acknowledgement

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This research is also a part of the Early Science Program of Argonne (AESP).

Reference

[1] X. He and L.-S. Luo, "Theory of the lattice boltzmann method: From the boltzmann equation to the lattice boltzmann equation," Phys. Rev. E, vol. 56, pp. 6811–6817, 1997.

[2] https://github.com/lucaso19891019/IMEXLB-1.0

[3] Y. H. Qian, D. d'Humieres, and P. Lallemand, "Lattice BGK Models for Navier-Stokes Equation," Europhys. Lett., vol. 17, no. 6, pp. 479–484, 1992.

[4] https://code.ornl.gov/j8g/lbm-proxy-app

[5] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," J. Parallel Distributed Comput., vol. 74, pp. 3202–3216, 2014.

[6] C. R. Trott et al., "Kokkos 3: Programming Model Extensions for the Exascale Era," in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 4, pp. 805-817, 1 April 2022, doi: 10.1109/TPDS.2021.3097283.