Abstract

In this work, we study the performance-portability of offload lattice Boltzmann kernels and the trade-off between portability and efficiency. The study is based on a proxy application for a lattice Boltzmann method (LBMM). 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 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.

Method

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

Results

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 faster in general 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 it is not as good for AB push pattern on NVIDIA devices. A possible explanation is that the AB push pattern has 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.

Kokkos

Kokkos is a C++ library that aims at uniting 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 Views Kokkos can be constructed on devices by simply managing the memory space according to selected backend. e.g. a double precision view with CUDA backend can be allocated as: Kokkos::View< double, Kokkos::Device< theta_gpu::cuDeviceType >, Kokkos::DefaultMemoryModel > view; Kokkos::View< double, Kokkos::Device< theta_gpu::cuDeviceType >, Kokkos::DefaultMemoryModel > view[0];
- Kokkos host views can be defined as mirrors of device views. Synchronization between host and devices can be done manually. e.g. Kokkos::FamilyOfView< double, Kokkos::Device< theta_gpu::cuDeviceType >, Kokkos::DefaultMemoryModel > view; Kokkos::FamilyOfView< double, Kokkos::Device< theta_gpu::cuDeviceType >, Kokkos::DefaultMemoryModel > view[0];
- Kokkos can be used to offload code to managed devices by Kokkos::runOn< theta_gpu::cuDeviceType >().
- Kokkos can also be used to execute specified execution space. e.g. Kokkos::runOn< theta_gpu::cuDeviceType >(...);
- Kokkos also includes multiple backends: (cuDeviceType, HIPDeviceType, D3Q19DeviceType) blocks on compilation of all outstanding asynchronous Kokkos operations.