Case Study for Performance Portability of GPU Programming Frameworks for Hemodynamic Simulations

ARISTOTLE MARTIN, Duke University, USA
AMANDA RANDLES (ADVISOR), Duke University, USA

Preparing for the deployment of large scientific and engineering codes on GPU-dense exascale systems is made challenging by the unprecedented diversity of vendor hardware and programming model alternatives for offload acceleration. To leverage the exaFLOPS of GPUs from Frontier (AMD) and Aurora (Intel), users of high performance computing (HPC) legacy codes originally written to target NVIDIA GPUs will have to make decisions with implications regarding porting effort, performance, and code maintainability. To facilitate HPC users navigating this space, we have established a pipeline that combines generalized GPU performance models with proxy applications to evaluate the performance portability of a massively parallel computational fluid dynamics (CFD) code in CUDA, SYCL, HIP, and Kokkos with backends on current NVIDIA-based machines as well as testbeds for Aurora (Intel) and Frontier (AMD). We demonstrate the utility of predictive models and proxy applications in gauging performance bounds and guiding hand-tuning efforts.

CCS Concepts: • **Computing methodologies** → **Parallel programming languages**; *Massively parallel and high-performance simulations*; • **Hardware** → **Emerging architectures**.

Additional Key Words and Phrases: Performance portability, Proxy applications, Computational fluid dynamics

1 INTRODUCTION

With the advent of diverse GPU-dense node architectures in exascale platforms, the issues surrounding performance portability of HPC codes across vendor hardware have taken center stage. The transition from predominantly NVIDIA GPU-based systems like Summit (ORNL) and Lassen (LLNL) to platforms consisting of nodes centered around Intel (Aurora, ALCF) and AMD (Frontier, ORNL) GPUs means that HPC users have to translate kernels written in CUDA to alternative offload acceleration languages supported on these devices. Vendors have developed software infrastructure centered around different programming models tailored to their hardware, which includes automated porting assist tools, such as AMD's HIPIFY. Beyond the vendor supported languages, users will also be able to choose from heterogeneous programming frameworks such as Kokkos [1], which includes an ever-growing list of supported backends that includes CUDA, HIP, and SYCL, among others. Selecting a programming model for a given application and hardware specification requires a detailed comparative analysis and will need to consider the relative porting efforts.

Here, we describe our experiences in porting a massively parallel fluid dynamics code, HARVEY [2], from its native CUDA implementation to SYCL, HIP, and Kokkos with CUDA, SYCL, HIP, and OpenACC as backend programming models. We establish a pipeline that uses predictive GPU performance models together with a proxy application to facilitate the porting process and guide additional hand-tuning.

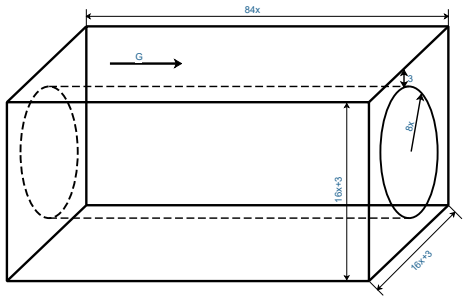

Fig. 1. Geometry setup of the test problem.

2 METHODS

2.1 Description of applications

The main application we use is HARVEY [2], an in-house, highly scalable CFD code based on the lattice Boltzmann method (LBM) developed to simulate blood flow in image-derived vasculature. Alongside HARVEY, we conduct runs using an open source proxy application based on LBM that solves simple tubular flows. To facilitate direct comparisons between the two applications, a cylindrical vessel is used for flow simulations as depicted in Fig. 1.

2.2 Overview of computing systems and porting procedure

Simulations are conducted on Summit (ORNL/NVIDIA V100), Polaris (ALCF/NVIDIA A100), Crusher (ORNL/AMD MI250X), and Sunspot (ALCF/Intel PVC), depicted in Fig. 2. We start with native CUDA versions of both HARVEY and the proxy app, and apply the same porting methods to both applications. SYCL codes are generated through the use of the DPC++ Compatibility Tool (DPCT). HIP ports are generated with HIPIFY. The Kokkos versions are fully manual ports.

2.3 Performance evaluation

Performance is measured in millions of fluid lattice updates per second (MFLUPS), which is a representative performance measure for LBM-based codes [2]. We strong scale piece-wise to maintain adequate GPU occupancy over a range of logical GPUs (equivalent to one PVC tile or a single graphics compute die on MI250X) and then increase the problem size proportionately to the increase in the total GPU count, or MPI task count. We scaled the number of GPUs in powers of 2 and doubled the scaling factor x of the geometry shown in Fig. 1 after increasing the number of GPUs by a factor of 8. Specifically, we sweep over $x = 12, 24, 48$, representing cubic increases in the number of fluid points.

Authors' addresses: Aristotle Martin, aristotle.martin@duke.edu, Duke University, 534 Research Dr., Durham, North Carolina, USA, 27705; Amanda Randles (ADVISOR), amanda.randles@duke.edu, Duke University, 534 Research Dr., Durham, North Carolina, USA, 27705.

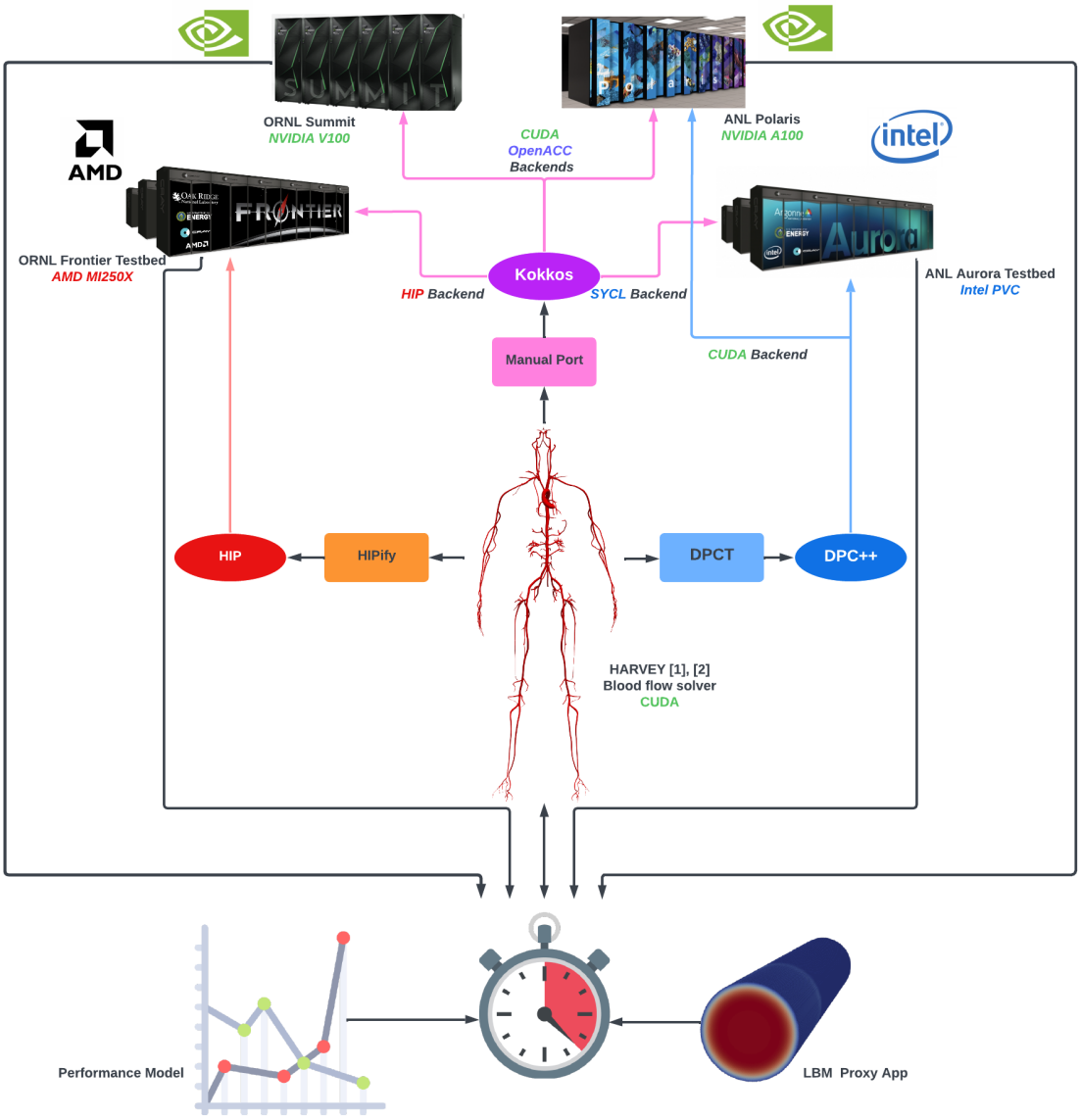

Fig. 2. Overview of the study, which evaluate the performance portability of different programming models and hardware configurations between HARVEY and an LBM proxy app. This diagram illustrates the diverse range of hardware and programming platforms investigated

.

2.4 Description of the GPU performance model

To help evaluate the ported codes, we extended a performance model previously developed by our group in [3]. The model uses the fact that LBM is memory bandwidth bound and approximates the time required for a GPU to process

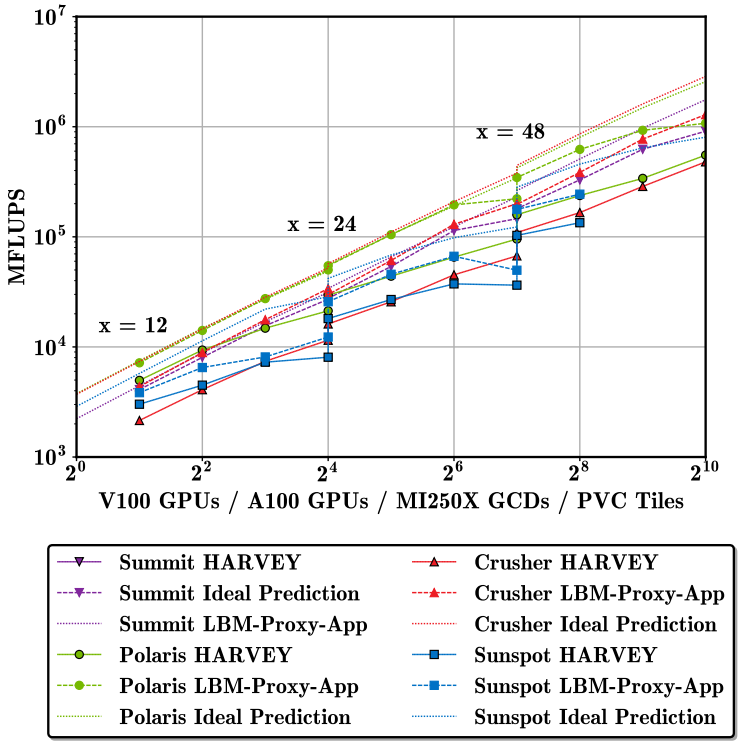

Fig. 3. Hardware comparison of HARVEY and LBM Proxy-App piecewise scaling performance (with LBM Proxy-App simulationSize value "X" labeled on the graph) to prediction using the native backend to each system on Summit (CUDA), Crusher (HIP), Polaris (CUDA), and Sunspot (SYCL).

fluid points as a function of the bandwidth measured with the Babel-STREAM benchmark [4]. The model also considers the latency of on-node and inter-node MPI message passing directly between GPUs, as well as host-device memory transfers, using a custom GPU PingPong benchmark. The model was used to predict the upper bound of piece-wise strong scaling performance in MFLUPS on each system of interest.

3 RESULTS

3.1 Comparison of native programming models on different hardware

We evaluated the relative performance of native programming models to each system for HARVEY and the LBM proxy app, along with corresponding performance predictions, shown in Fig. 3. Overall, native CUDA on NVIDIA A100s of Polaris outperformed the other native models, seen with both HARVEY and the proxy app. For the smallest GPU counts, HIP performs the worst, but then becomes competitive for multi-node runs. The SYCL code exhibited the largest performance gains during the weak scaling portions of the graph.

3.2 Comparison of programming model backends

Fig. 4 shows the relative performance of programming models versus the native model on each system used in this study. Overall, the native programming models generally outperform the “off-brand" models.

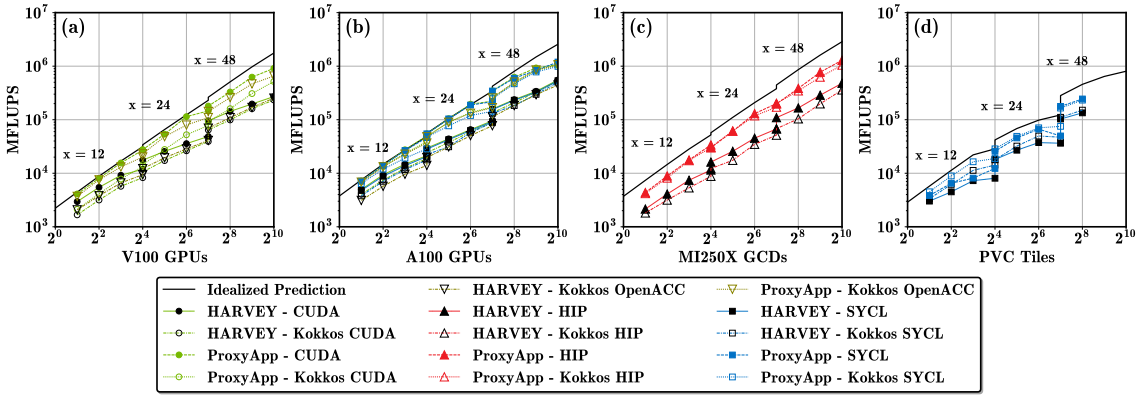

Fig. 4. Backend comparison of HARVEY and LBM proxy app piece-wise strong scaling performance (with LBM Proxy-App simulation-Size value "X" labeled on the graph) to performance predictions using a variety of backends across 4 different systems: (a) Summit, (b) Polaris, (c) Crusher, (d) Sunspot.

4 CONCLUSION AND FUTURE WORK

This study has established a framework for systematically evaluating the performance portability of heterogenous programming models on modern supercomputers, using a real-world application. We demonstrated the utility of a performance model and proxy application in facilitating the portability analysis. With a large number of backends, Kokkos was the most portable of the codes and could be run on every system used here, but required the most work upfront in manual porting. HIP, on the other end, was the least portable but required the least porting effort. In future work, we will evaluate performance portability of programming models in the context of fluid-structure simulations of biological cells.

ACKNOWLEDGMENTS

Computing support for this work came from the Argonne National Laboratory (ANL) Aurora Early Science program. An award of compute time was provided by the INCITE Program. This research also used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

REFERENCES

[1] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, 2022. doi: 10.1109/TPDS.2021.3097283.

[2] Amanda Peters Randles, Vivek Kale, Jeff Hammond, William Gropp, and Efthimios Kaxiras. Performance analysis of the lattice boltzmann model beyond navier-stokes. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 1063–1074. IEEE, 2013.

[3] William Ladd, Christopher Jensen, Madhurima Vardhan, Jeff Ames, Jeff R Hammond, Erik W Draeger, and Amanda Randles. Optimizing cloud computing resource usage for hemodynamic simulation. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 568–578. IEEE, 2023.

[4] Tom Deakin, James Price, Matt Martineau, and Simon McIntosh-Smith. Evaluating attainable memory bandwidth of parallel programming models via babelstream. *International Journal of Computational Science and Engineering*, 17(3):247–262, 2018.