Taking GPU Programming Models to Task for Performance Portability

Joshua H. Davis, Pranav Sivaraman, Joy Kitson, Konstantinos Parasyris, Harshitha Menon, Isaac Minn, Giorgis Georgakoudis, Abhinav Bhatele

PARALLEL SOFTWARE AND SYSTEMS GROUP

ICS

UNIVERSITY OF MARYLAND

The Team

UMD


Abhinav Bhatele


Joshua H. Davis


Pranav Sivaraman


Joy Kitson


Isaac Minn

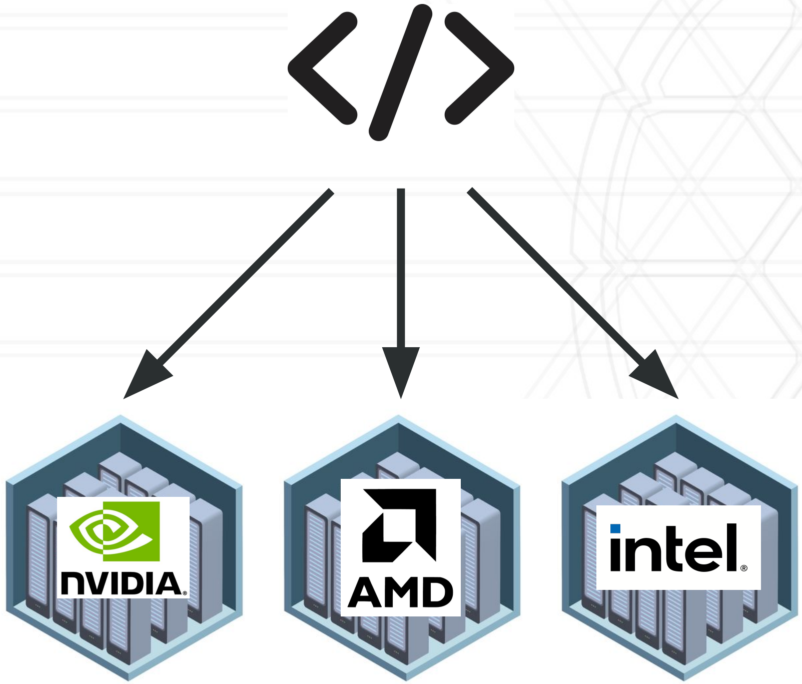
LLNL


Harshitha Menon


Giorgis Georgakoudis


Konstantinos Parasyris

The Performance Portability Problem



- One code, multiple systems
- Perlmutter, Frontier, and Aurora supercomputers use three different GPU vendors
- **How to maintain a single codebase that achieves good performance on all platforms?**

Portable Programming Models

- GPU programming models can provide a solution through portable abstractions of hardware capabilities

Research Question: How well do each of the available GPU programming models enable performance portability in practice?

OpenACC
More Science, Less Programming

RAJA

AMD ROCm

NVIDIA CUDA

Methodology: Proxy Apps

- We surveyed a large number of proxy applications
  - Selected five from a range of domains, which all had some existing implementations
- Implemented missing versions where needed, verified with test cases
- Ensured each implementation of an app does the same thing
- Ensured that the timers were consistent
  - We measured data movement time but don't show that here
- Selected largest input problems that would fit in DRAM of all devices
- Picked five different GPU-based platforms

Proxy Apps Used

1. **BabelStream**: memory bandwidth benchmark with five kernels
2. **XSBench**: cross-section lookup kernel with irregular memory access
   - Proxy for OpenMC, (Monte Carlo neutron transport)
3. **CloverLeaf**: 2D structured compressible Euler equations solver
   - From Mantevo Suite
4. **su3_bench**: implements SU(3) a complex number matmul
   - Proxy for MILC (Lattice QCD)
5. **miniBUDE**: molecular docking kernel with compute-bound characteristics
   - Proxy for BUDE (Bristol University Docking Engine)

New Ports and Development Efforts

Proxy Application	Scientific Domain	Method(s)	Suite	CUDA	HIP	SYCL	Kokkos	RAJA	OpenMP	OpenACC	Spack Pkg.
BabelStream	N/A	Bandwidth benchmark	N/A	E	E	E	E	M	E	E	M
XSBench	Nuclear physics	Monte Carlo	ECP	E	E	M	C	C	E	C	M
CloverLeaf	Hydrodynamics	Structured grid	Mantevo	E	E	M	E	C	E	C	M
su3_bench	Particle physics	Structured grid, dense lin. alg.	NERSC	E	E	E	E	C	E	E	C
miniBUDE	Molecular dynamics	N-body	N/A	E	E	M	E	M	E	E	C

- Several new ports created (C), some apps modified to align timed regions (M)
- Spack packages enable reproducible environment – build apps with one command
- Working on upstreaming all changes for usage by the wider community

Hardware Platforms

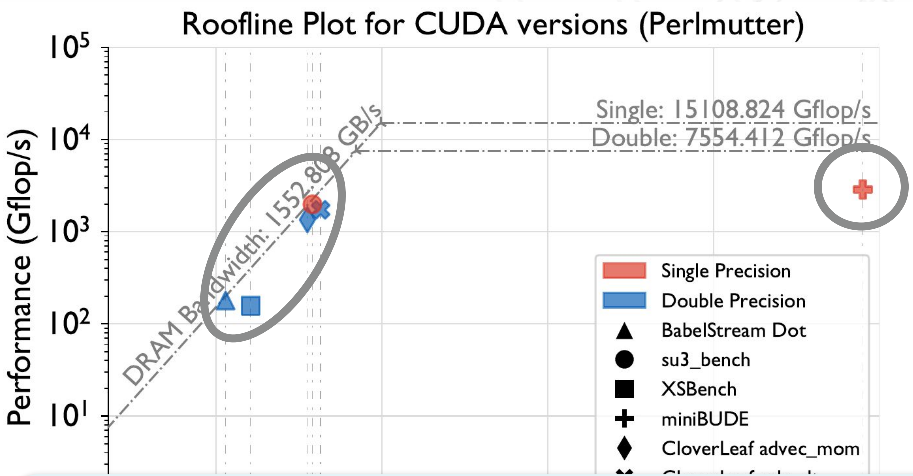
System	GPU Model	DRAM Size
Summit (ORNL)	NVIDIA **V100** GPU	32 GB
Perlmutter (NERSC)	NVIDIA **A100** GPU	40 GB
Zaratan (UMD)	NVIDIA **H100** GPU	80 GB
Corona (LLNL)	AMD **MI50** GPU	32 GB
Frontier (ORNL)	AMD **MI250X** GPU	64 GB

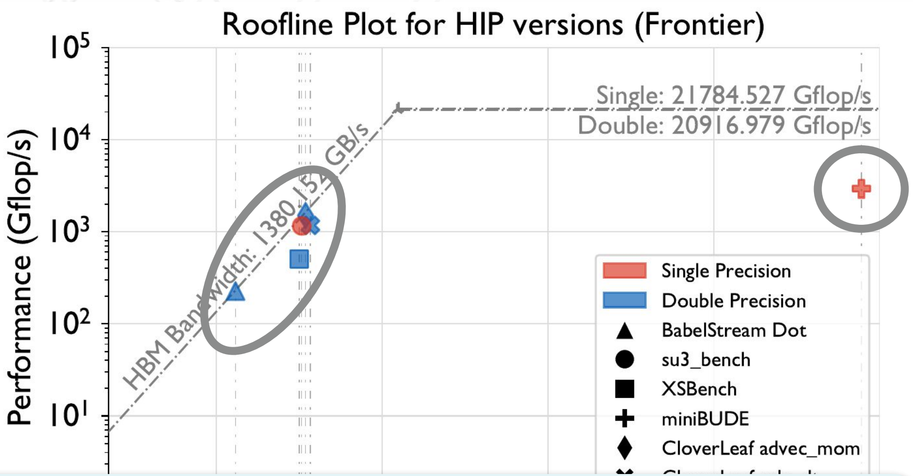
Other Experimental Considerations

- Keep compilers the same as much as possible unless >5% improvement from using another compiler
- Data points represent mean of three trials, variation was very small
- Use developer's choices for kernel parameters – no tuning!

Prog. Model	NVIDIA	AMD
CUDA	GCC 12.2.0	N/A
HIP	N/A	ROCmCC 5.7.0
SYCL*	DPC++ 2024.01.20	DPC++ 2024.01.20
Kokkos	GCC 12.2.0	ROCmCC 5.7.0
RAJA	GCC 12.2.0	ROCmCC 5.7.0
OpenMP*	NVHPC 24.1	LLVM 17.0.6
OpenACC	NVHPC 24.1	Clacc 2023-08-15

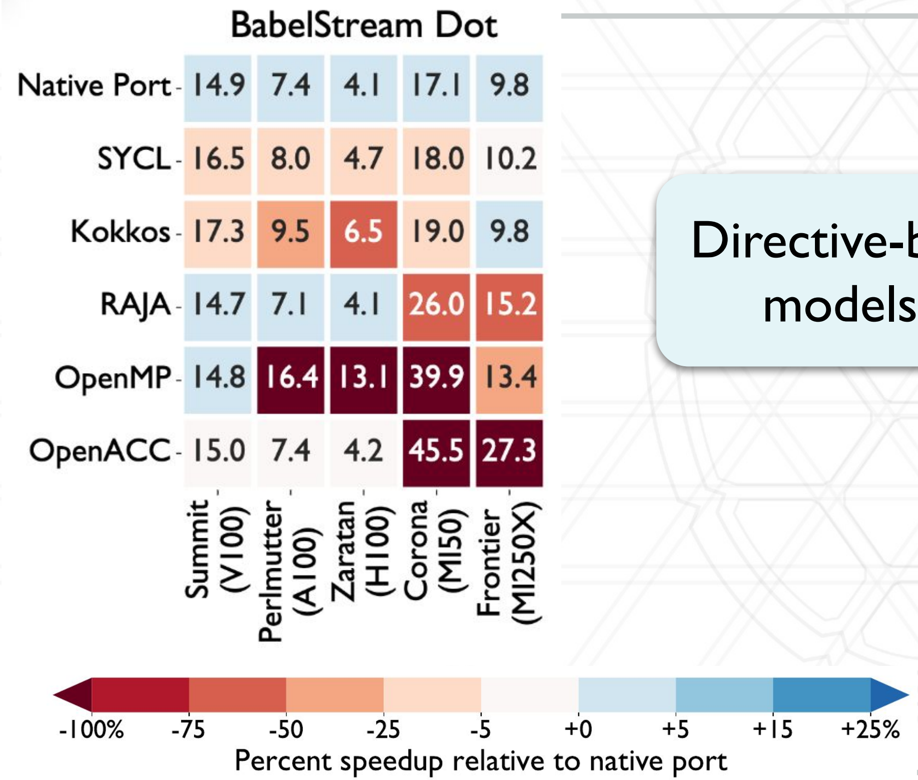
A100 and MI250X Roofline Plots





Most kernels are memory-bound in this work, except miniBUDE

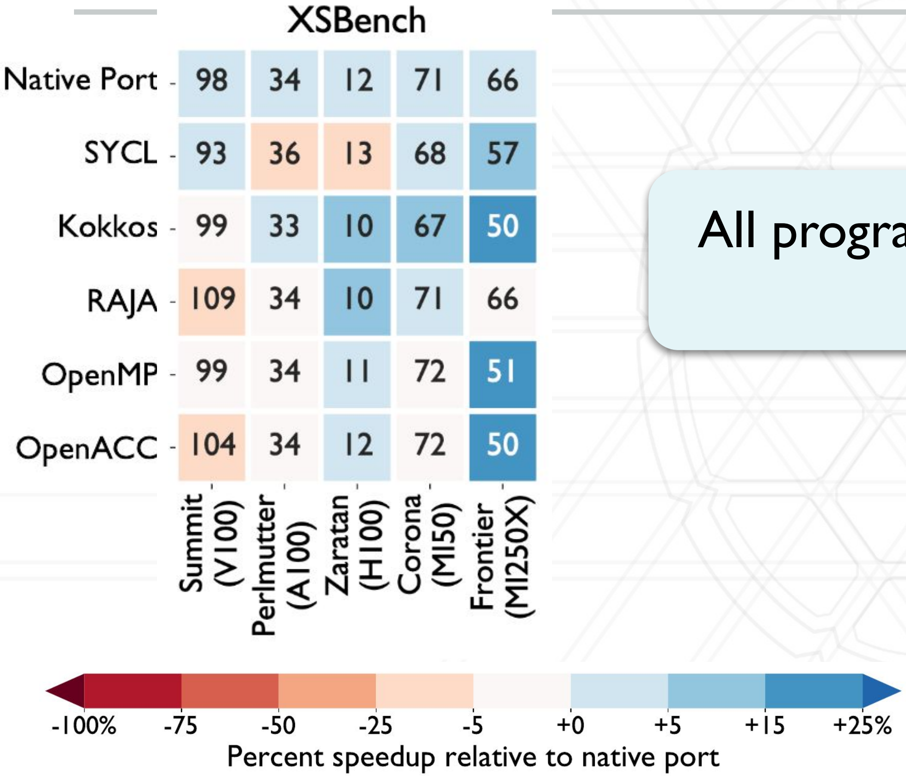
BabelStream dot Reduction Kernel



Directive-based models struggle more than other models with dot, a simple reduction kernel

Number in cell is runtime (seconds)
Blue means faster than native, red means slower

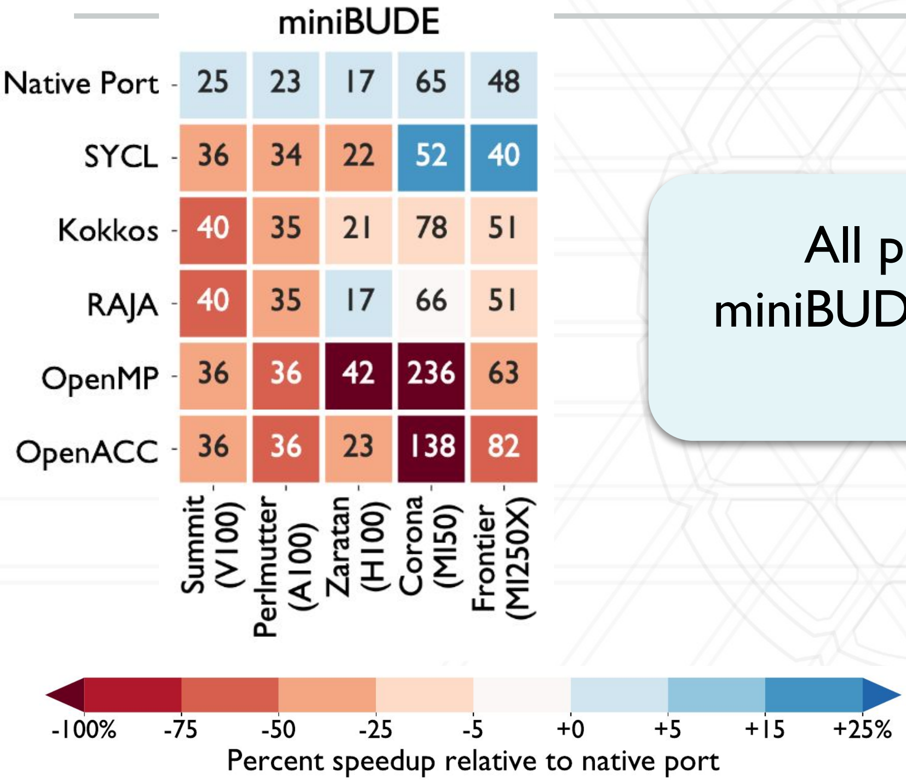
XSBench Results



All programming models can achieve reasonable portability for XSBench

Number in cell is runtime (seconds)
Blue means faster than native, red means slower

miniBUDE Results



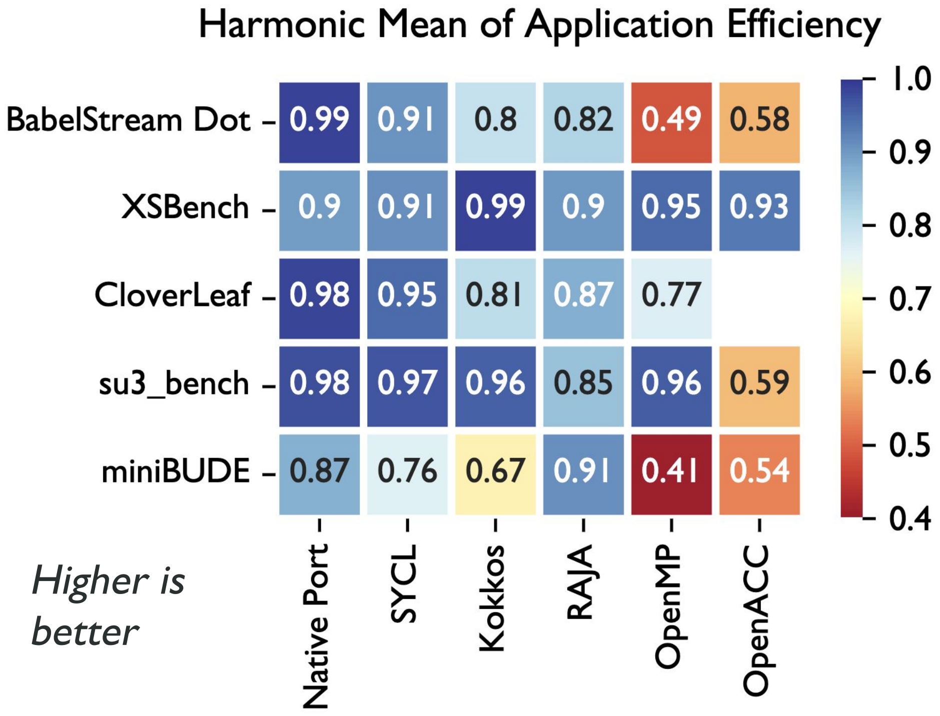
All programming models struggle with miniBUDE, a compute-bound kernel relying on shared memory

Number in cell is runtime (seconds)
Blue means faster than native, red means slower

Portability Optimizations

- Kokkos CloverLeaf: switching reduction operation from 2D to 1D parallelism
  - Improves performance on all systems due to fewer barrier stalls
- OpenMP/OpenACC su3_bench: align complex number struct to 32-byte boundary
  - Improves performance on NVIDIA by reducing load instructions
- RAJA miniBUDE: leverage new dynamic shared memory features
  - Makes the RAJA port more consistent with other models, improves NVIDIA performance

Performance Portability Metric



Higher is better

- Pennycook (2019) metric for performance portability
- Applied to performance after our optimizations
- SYCL, Kokkos, and RAJA generally outscore OpenMP and OpenACC

Conclusion

- We examined performance portability for seven programming models across NVIDIA and AMD GPUs
  - SYCL, Kokkos, and RAJA slightly outperform OpenMP and OpenACC
  - HIP performance is sometimes surprisingly poor without additional tuning
- Building these apps is now significantly more automated thanks to our efforts in creating Spack environments
- Clearer compilation processes, improved profiling tools, and exposure of additional tuning capabilities in the programming models are all on the critical path

jhdavis@umd.edu

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant No. 2047120, the NSF Graduate Research Fellowship Program under Grant No. DGE 2236417, and the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, DOE Computational Science Graduate Fellowship under Award No. DE-SC0021.

This work was performed in part under the auspices of the U.S. DOE by Lawrence Livermore National Laboratory under Contract DE-AC5207NA27344 (LLNL-CONF-855581).

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC05-00OR22725, and of the National Energy Research Scientific Computing Center (NERSC), a U.S. DOE Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC awards DDR-ERCAP0025593 and DDR-ERCAP0029890.

jhdavis@umd.edu

Spack Environment Efforts

- All miniapps now have working Spack packages to build on NVIDIA and AMD systems, for all programming models
- Created a Spack Environment for the suite to allow users to install all applications and dependencies with one command
- Collecting experimental results also automated using Spack Python scripting tools

Spack

Insights from Porting Experiences

- OpenACC to OpenMP
  - Straightforward, just rote directive substitution
- Porting to Kokkos:
  - Boilerplate code needed if you want to avoid changing original data structures
  - Convert C++ dynamic arrays to unmanaged Views to deep_copy into device View
- Porting to RAJA:
  - Relying on separate libraries (RAJA and Umpire) for compute and data portability makes compilation harder to get right
  - But, permits incremental development, and Spack can help (develop environments)

Remaining Portability Outliers

- Lagging OpenACC performance on AMD GPUs
- Poor performance with CloverLeaf and miniBUDE for Kokkos, OpenMP, and OpenACC
- Poor reduction performance for OpenACC and OpenMP
- Poor reduction performance on AMD GPUs for RAJA

Compilers Used

Prog. Model	Summit	Perlmutter	Corona	Frontier
CUDA	GCC 12.2.0	GCC 12.2.0	N/A	N/A
HIP	N/A	N/A	ROCmCC 5.7.0	ROCmCC 5.7.0
SYCL*	DPC++ 2024.01.20	DPC++ 2024.01.20	DPC++ 2024.01.20	DPC++ 2024.01.20
Kokkos	GCC 12.2.0	GCC 12.2.0	ROCmCC 5.7.0	ROCmCC 5.7.0
RAJA	GCC 12.2.0	GCC 12.2.0	ROCmCC 5.7.0	ROCmCC 5.7.0
OpenMP†	NVHPC 24.1	NVHPC 24.1	LLVM 17.0.6	LLVM 17.0.6
OpenACC	NVHPC 24.1	NVHPC 24.1	Clacc 2023-08-15	Clacc 2023-08-15

* We use AdaptiveCpp 23.10.0 for SYCL CloverLeaf due to performance improvement.

† We use ROCmCC 5.7.0 for OpenMP su3_bench on AMD GPUs due to performance improvement.

Portable Programming Models

- GPU programming models can provide a solution through portable abstractions of hardware capabilities
- …but developers now have many options to choose from:

Directive-based

OpenMP

OpenACC
More Science, Less Programming

Library-based

kokkos

RAJA

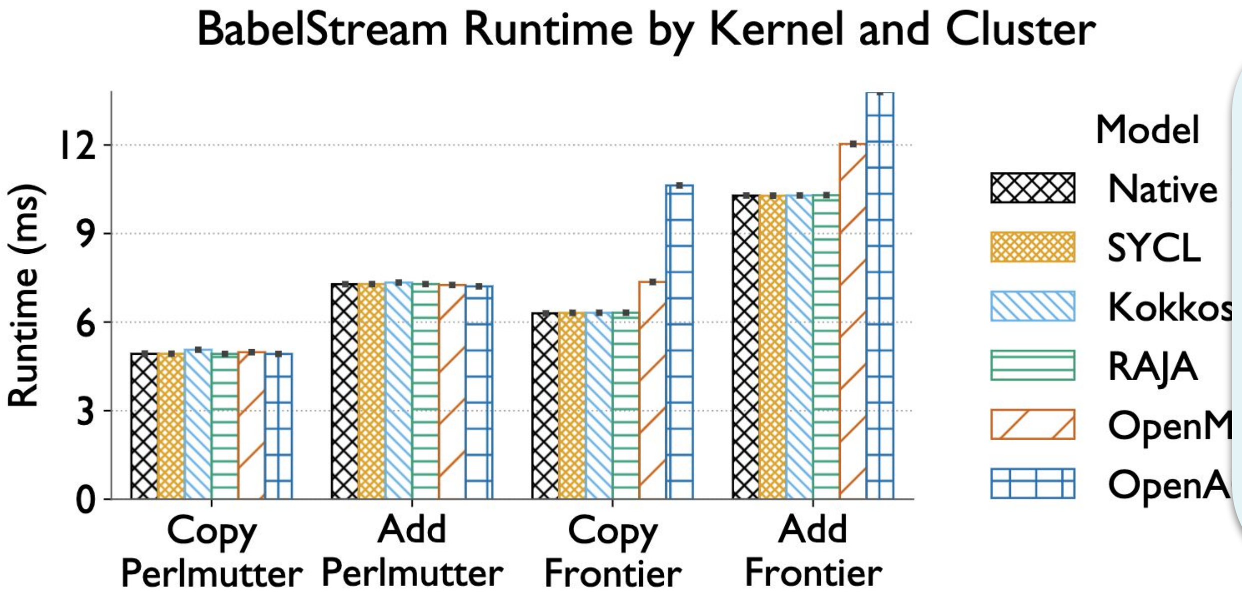
Language Extension

AMD ROCm

SYCL

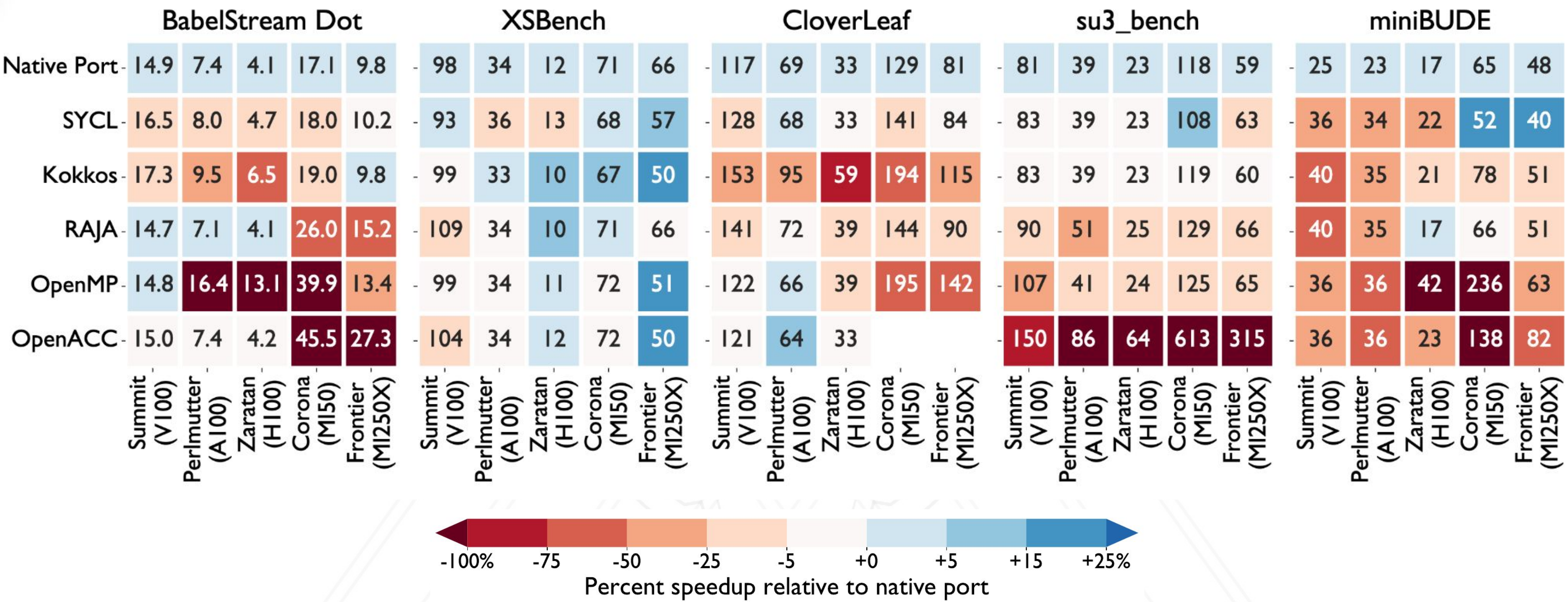
NVIDIA CUDA

Simple Kernels in BabelStream



For BabelStream kernels besides dot, all models can provide excellent performance portability besides directive-based models on AMD GPUs

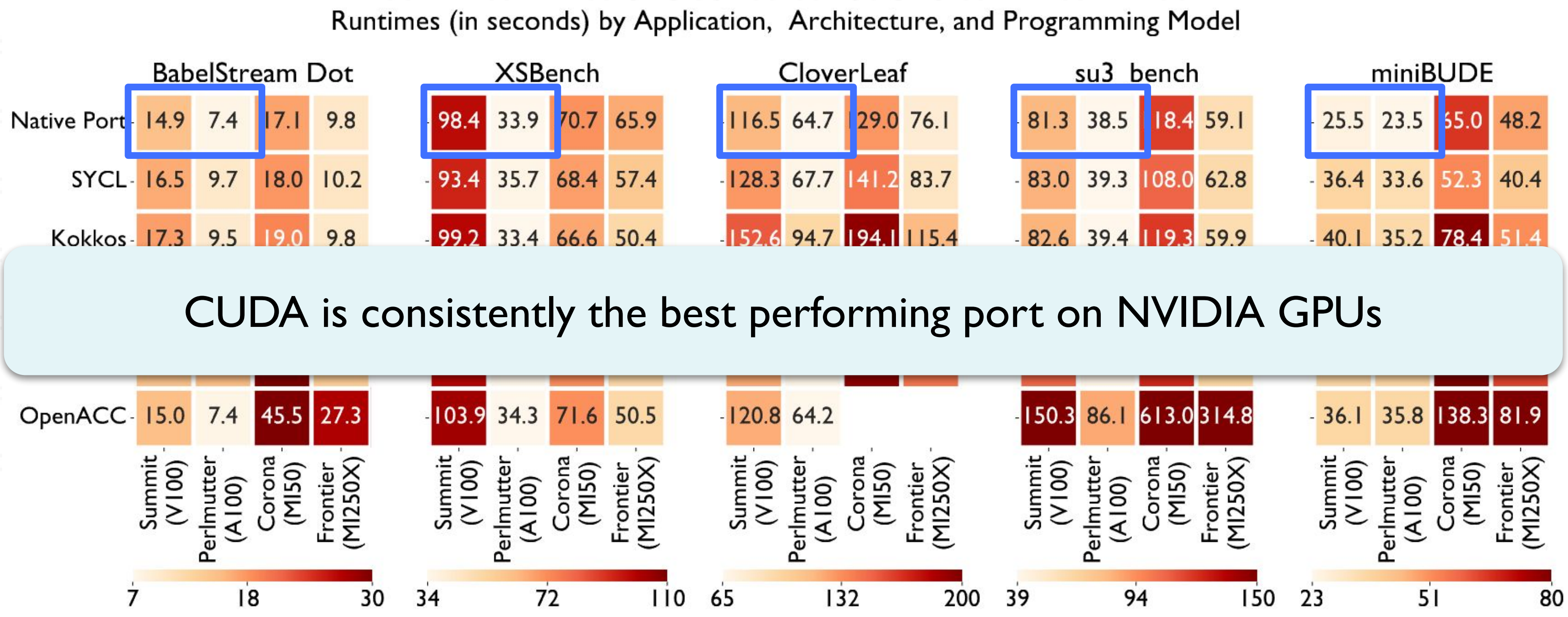
All Results



PSSG

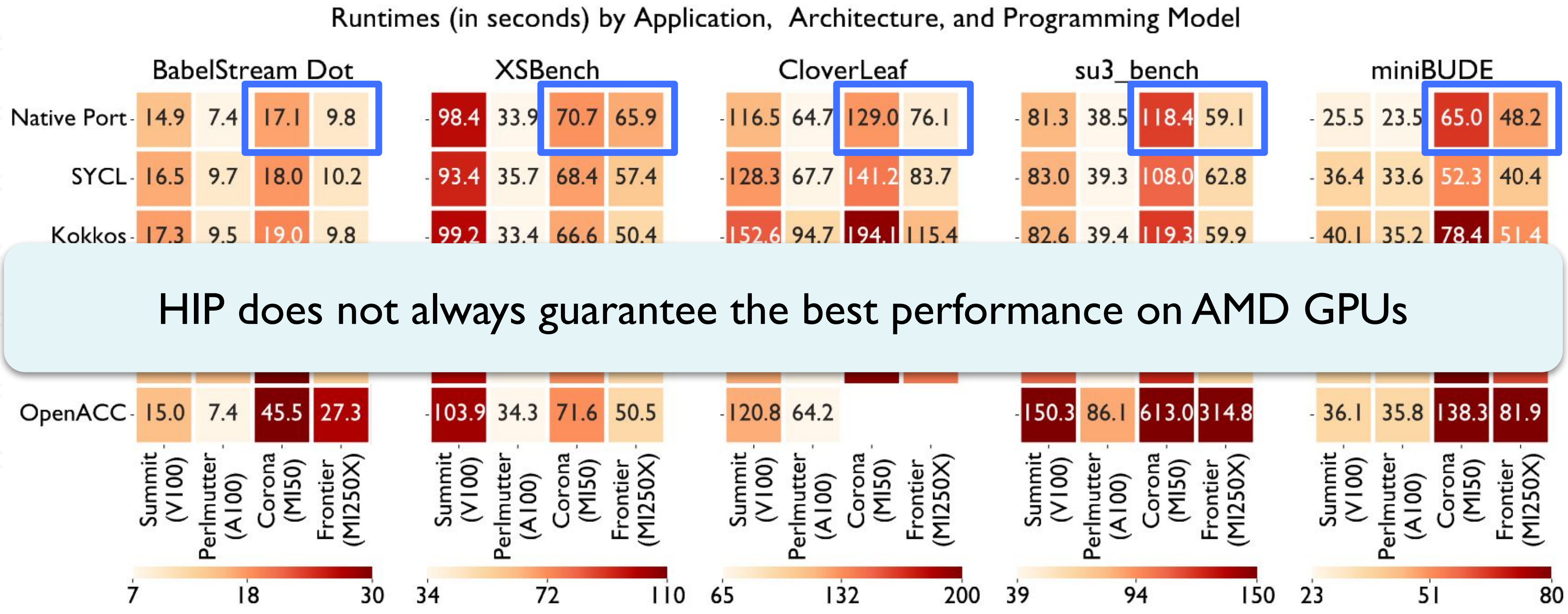
UNIVERSITY OF MARYLAND

Observation 1: CUDA on NVIDIA



CUDA is consistently the best performing port on NVIDIA GPUs

Observation 2: HIP on AMD

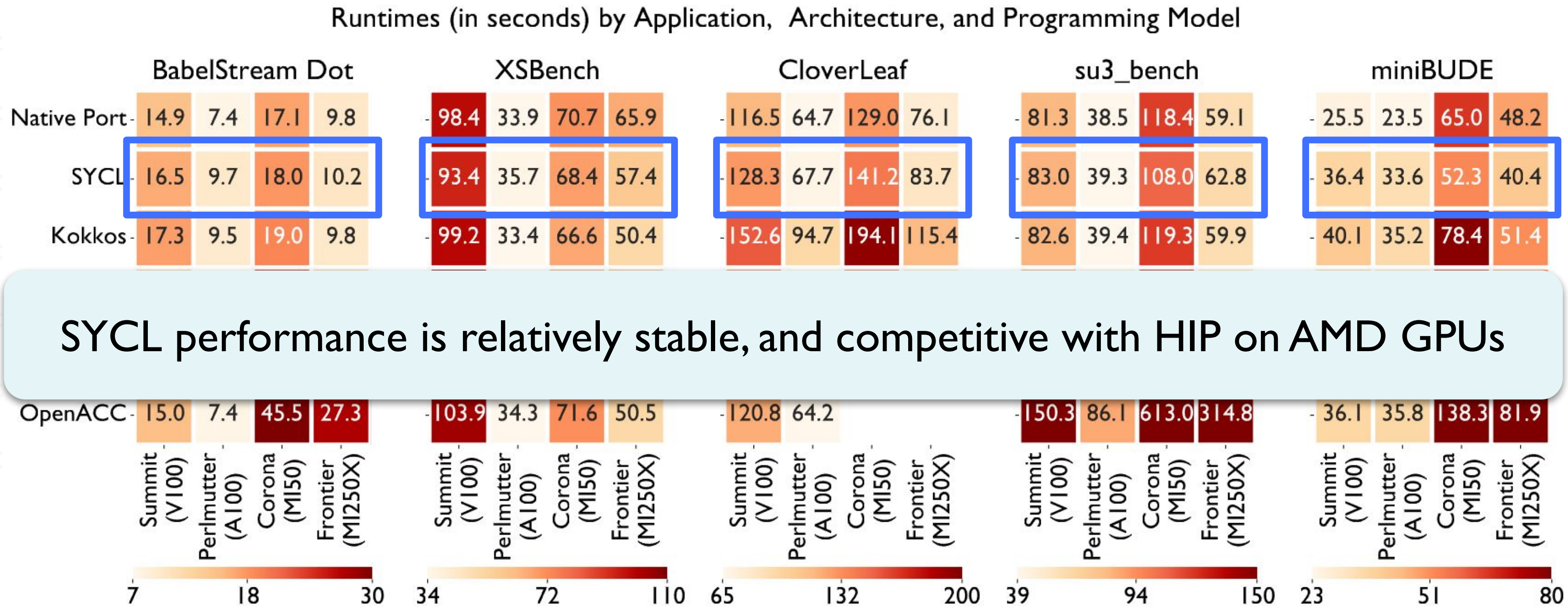


HIP does not always guarantee the best performance on AMD GPUs

PSSG

UNIVERSITY OF MARYLAND

Observation 3: SYCL

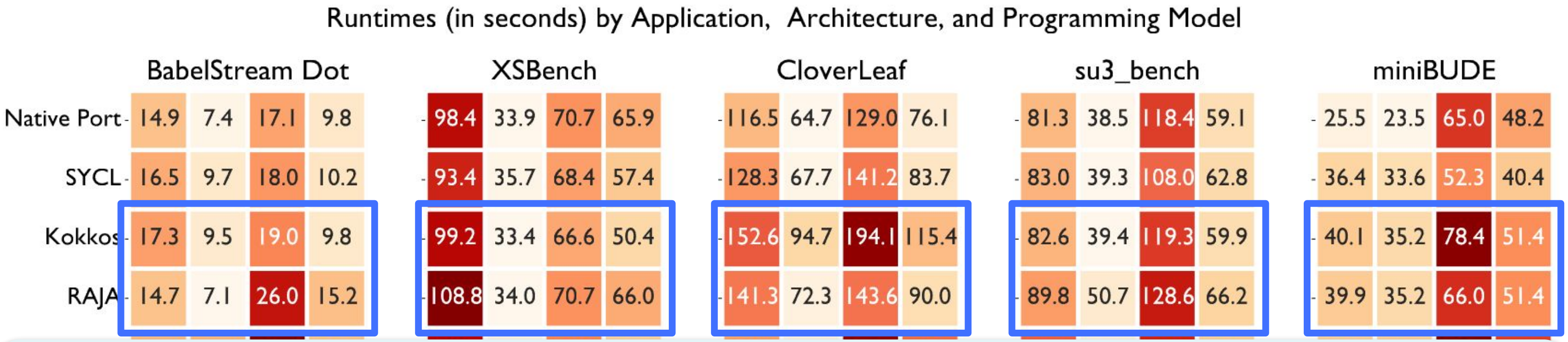


SYCL performance is relatively stable, and competitive with HIP on AMD GPUs

PSSG

UNIVERSITY OF MARYLAND

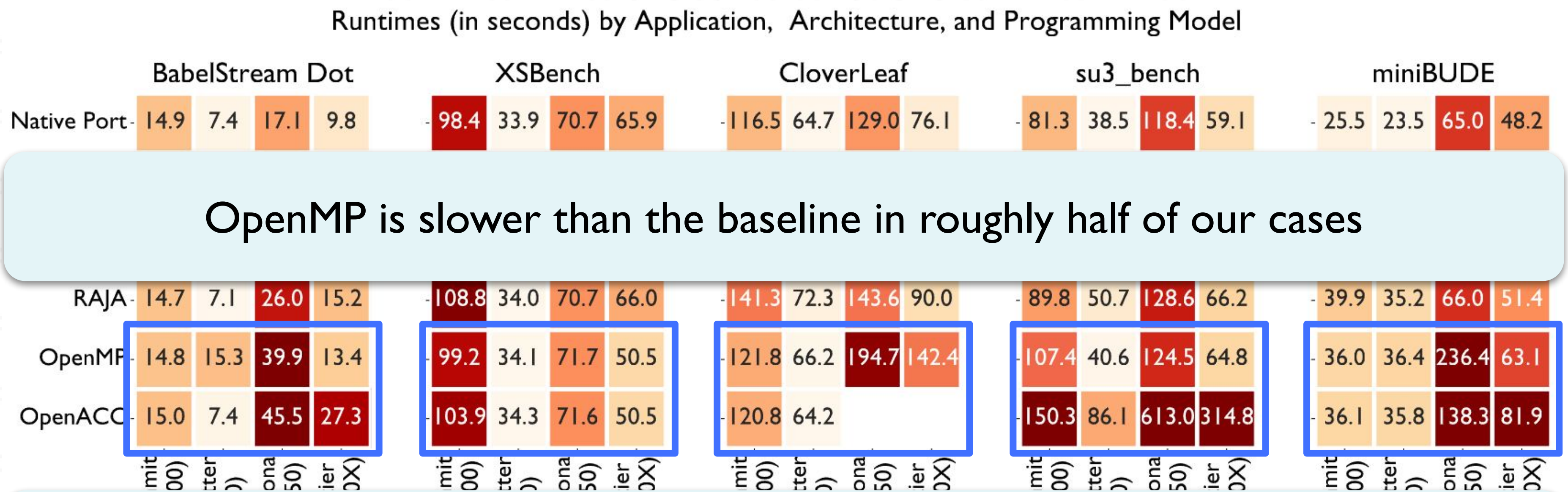
Observation 4: Kokkos and RAJA



Kokkos and RAJA can achieve near-baseline performance

Kokkos slightly better for AMD, RAJA slightly better for NVIDIA

Observation 5: OpenMP and OpenACC



OpenMP is slower than the baseline in roughly half of our cases

OpenACC performance suffers on AMD GPUs