ISC High Performance

SHAPING TOMORROW

JUNE 24 - JULY 2, 2021 | ISC-HPC.COM

Andrei Poenaru

Wei-Chen Lin

Simon McIntosh-Smith

A Performance Analysis of Modern Parallel Programming Models Using a Compute-Bound Application

University of BRISTOL

UKRI Engineering and Physical Sciences Research Council

GW4

Introduction to the GW4 Isambard 2 supercomputer

- **Isambard 2** is a £4.1M EPSRC project, run by a consortium of the GW4 Alliance, the Met Office, HPE/Cray, Fujitsu and Arm, to deliver a Tier-2 HPC service to researchers across the UK and around the world
- Funded in late 2019, Isambard 2 builds on Isambard 1's achievements as the world's first Arm64-based production supercomputer
- **Isambard 1** has been a huge success, proving for the first time that Arm works for supercomputing in production environments

Isambard 2 production system

- Cray XC50 ‘Scout’ with Aries interconnect
  - 21,504 ARMv8 cores (336n x 2s x 32c)
  - Marvell ThunderX2 32 core @ 2.5 GHz
- HPE Apollo 80 A64FX system
  - 3,456 Fujitsu A64FX cores (72n x 48c)
- Multi-Architecture Comparison System
  - CXL, Rome, V100, …
  - Used for previous performance portability work [1]
- Cray HPC optimised software stack
- Hosted for the Consortium by the Met Office in Exeter

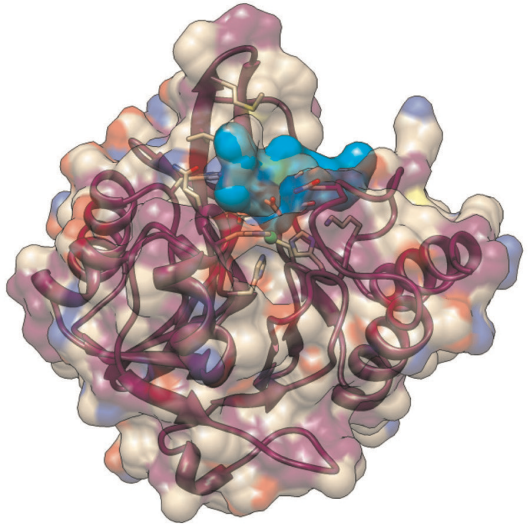
Performance Portability

- The upcoming exascale systems together utilise combinations of CPUs from two vendors and GPUs from three vendors
  - It is not desirable to use platform- or vendor-specific programming models
  - A portable approach is needed
- The C++ language has particular appeal
  - It can achieve performance similar to C and Fortran
  - It offers modern features to write more expressive and safer code
- Parallel C++ hopes to outweigh any lost performance with programmer productivity

Modern Programming Models

- Two modern, single-source frameworks with a focus on performance, portability, and productivity have emerged: **Kokkos** and **SYCL**
  - **Kokkos** is a new framework developed natively for C++
    - Distributed as source, builds with the host compiler
  - **SYCL** builds on previous OpenCL toolchains and integrates them with modern C++
    - Compilers: ComputeCpp, DPC++, hipSYCL
- Both frameworks can generate machine code for CPUs and GPUs without any change to the high-level source code

The Bristol University Docking Engine (BUDE)



- BUDE is an application for *in silico* molecular docking
  - Predicts the structures formed between molecules and estimates the strength of their interactions [2]
- Docking is computationally challenging because of the many different ways in which two molecules may be arranged together
  - 3 translational and 3 rotational degrees of freedom
  - Order of 10^7 trials
- The most computationally intensive mode is *virtual screening*
  - Molecules of drug candidates (*ligands*) are generated using a genetic algorithm and are bonded to a *target* protein molecule

miniBUDE: A Compute-Bound Mini-App

- A mini-app for BUDE virtual-screening runs
- Kernels written in widely used parallel programming models
  - Baseline implementation written in **OpenCL** – virtually identical to the core kernel of the full-scale BUDE application
  - **CUDA** port with minimal changes
  - **CPU OpenMP**
  - **OpenMP target** and **OpenACC** based on CPU version
  - Re-implementations in **Kokkos** and **SYCL**
- Achieves >56% of peak compute performance on CXL and Rome

Evaluation Methodology

- All mainstream HPC compilers applicable to each model
  - All optimisations enabled and target set explicitly, similar to `-march=native -Ofast`
  - Thread binding, e.g. with `OMP_PROC_BIND` or `aprun`
- Two input decks:
  - Small – 26 ligands
  - Large – 2672 ligands
- Warm-up + 8 iterations, 2^{16} poses per iteration
  - Iterations correspond to genetic algorithm generations

Hardware

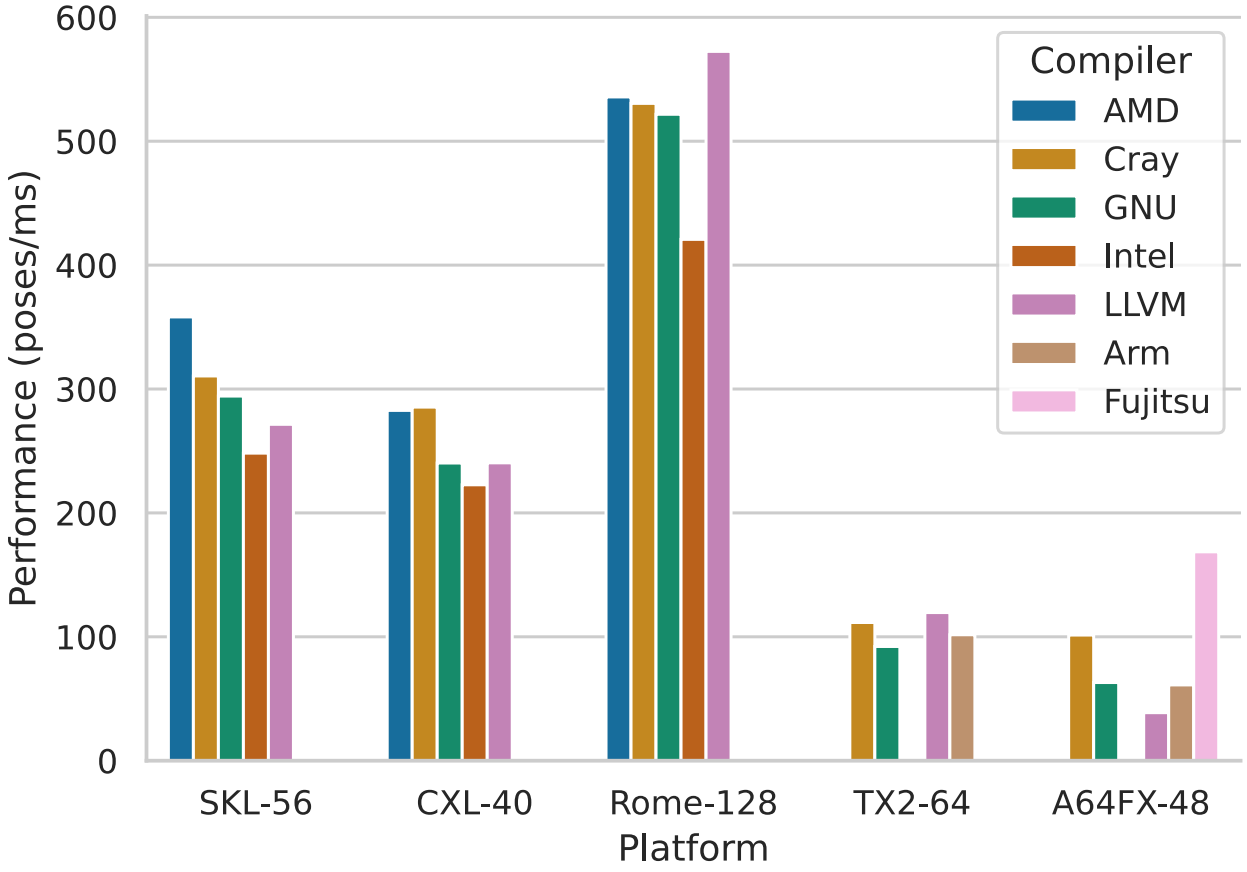
CPUs

Platform	Cores	Clock Speed	Peak FLOP/s (32-bit)
AMD EPYC Rome 7742	2 x 64	2.25 GHz	13.8 TFLOP/s
Fujitsu A64FX	48	1.8 GHz	5.5 TFLOP/s
Intel Skylake 8176	2 x 28	2.1 GHz	5.7 TFLOP/s
Intel Cascade Lake 6230	2 x 20	2.1 GHz	4.0 TFLOP/s
Marvell ThunderX2	2 x 32	2.5 GHz	2.6 TFLOP/s

GPUs

Platform	Cores	Clock Speed	Peak FLOP/s (32-bit)
AMD Radeon VII	60	1.4 GHz	13.8 TFLOP/s
Intel Iris Pro 580	72	0.95 GHz	1.1 TFLOP/s
NVIDIA V100	80	1.13 GHz	15.7 TFLOP/s

Results: CPU + OpenMP

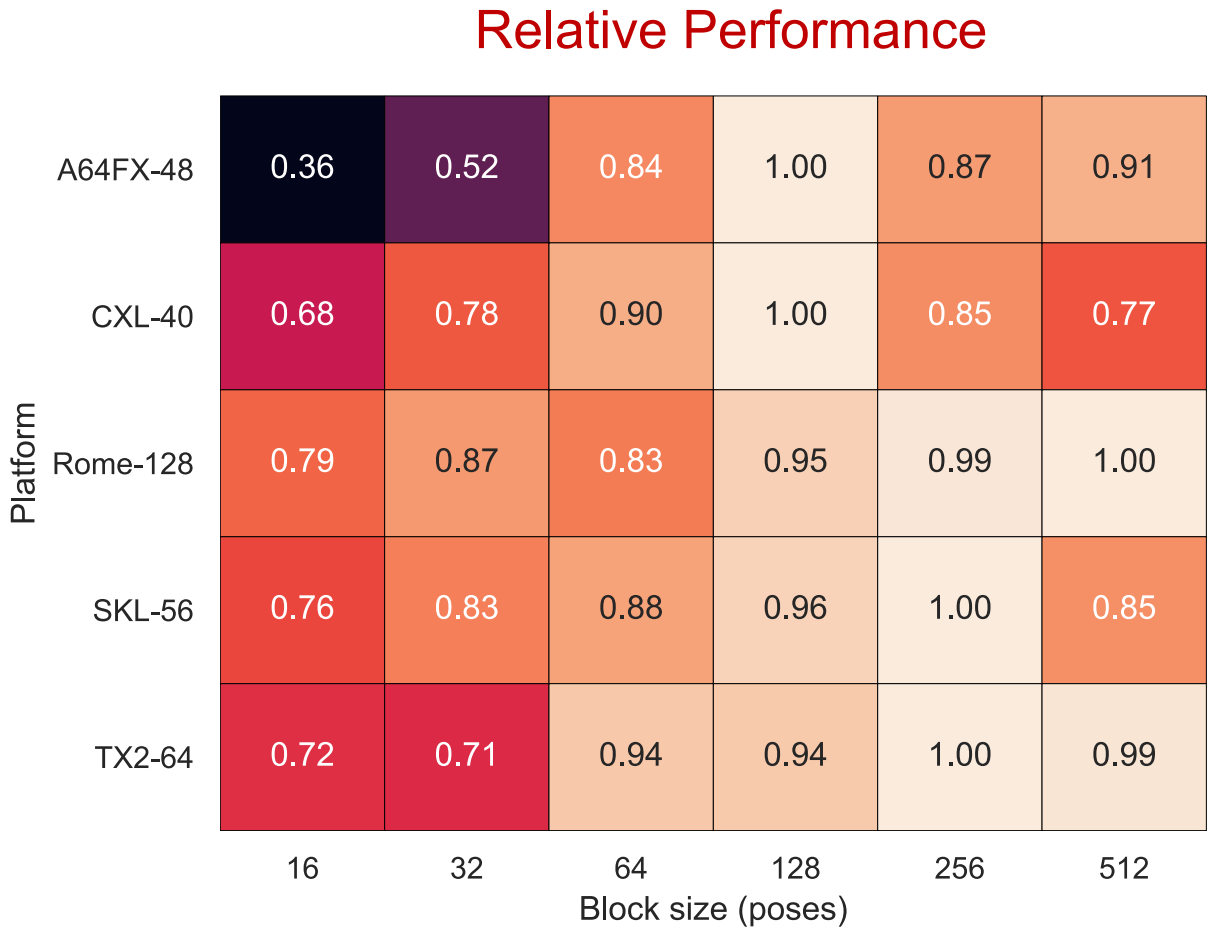


Higher numbers correspond to higher performance.

- Compiler versions are latest available
  - Full details in the paper
- Vectorisation played a big role in achieving good performance
- The A64FX relies on the heavy optimisations from the Fujitsu compiler
  - Including software pipelining

CPU + OpenMP Workgroup Size

- Heatmap shows fractions of performance, normalised to best result on each platform
- The workgroup size influenced both vectorisation and loop unrolling
  - Forcing unrolling and interleaving in the compiler helps on A64FX
- In general, the more OoO resources available, the higher the optimal workgroup size



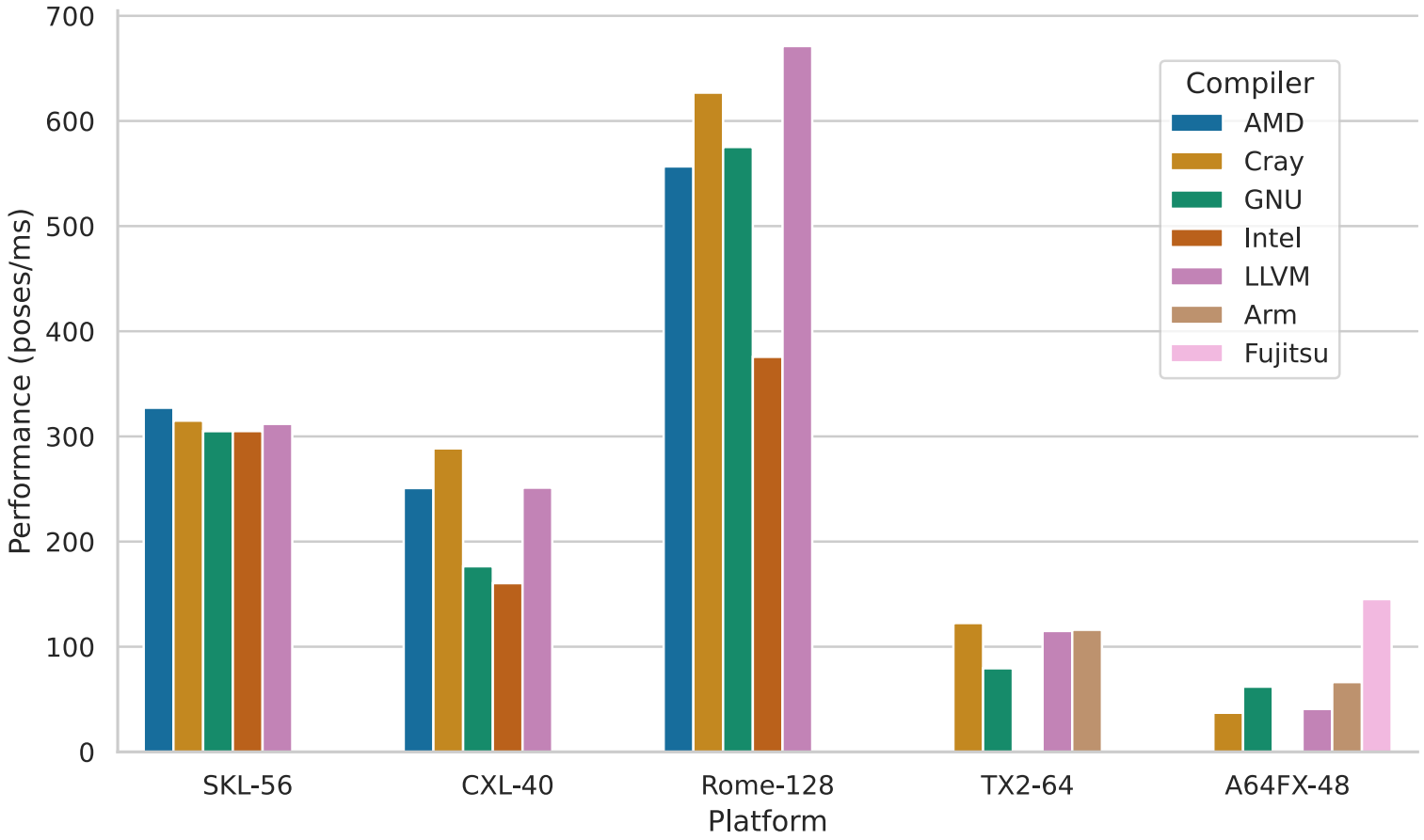
Lighter colours correspond to higher performance.

University of BRISTOL

UKRI Engineering and Physical Sciences Research Council

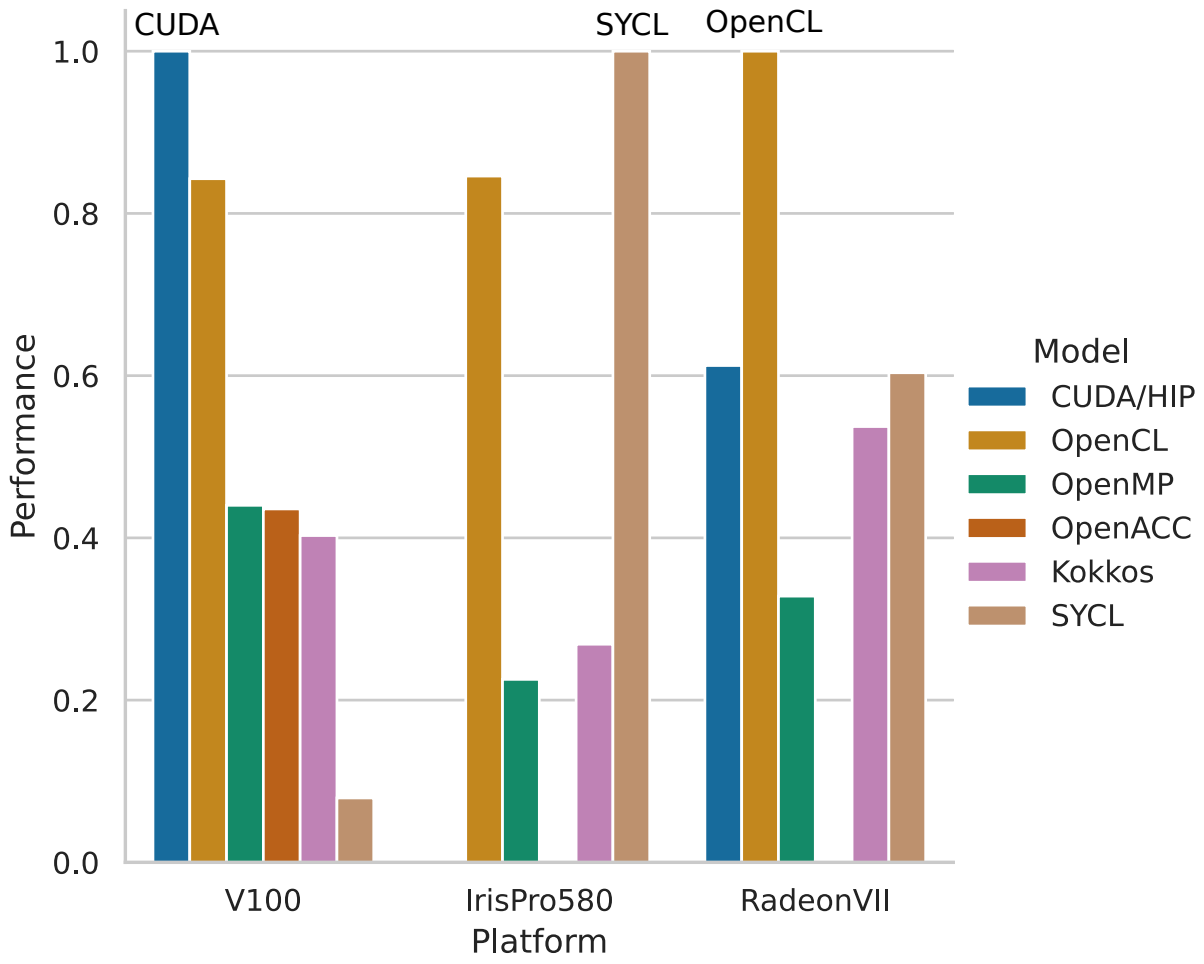
GW4

Results: CPU + Kokkos



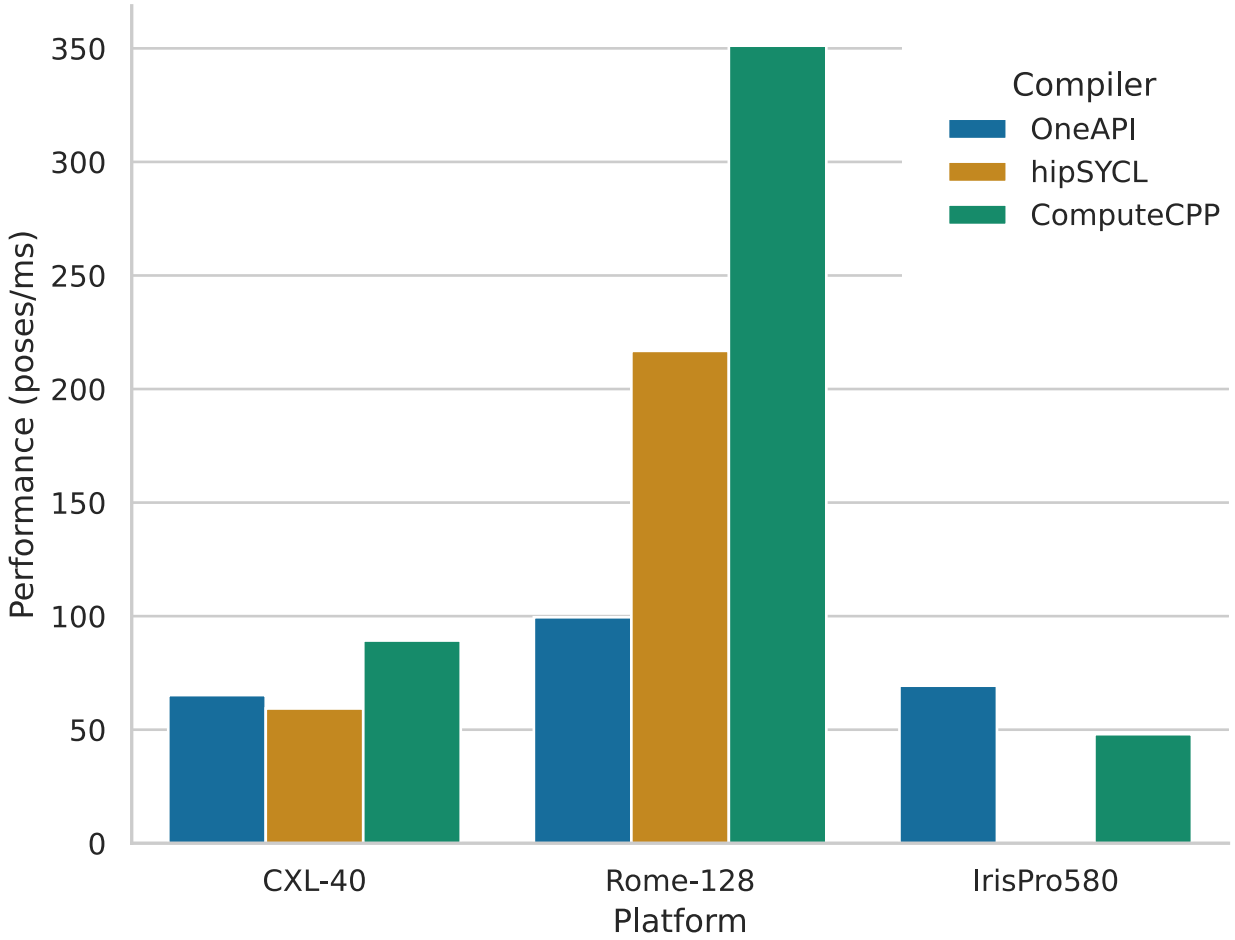
- Strong correlation to baseline OpenMP results
  - Indicated Kokkos uses OpenMP backend efficiently on all architectures
- Fujitsu Compiler critical for A64FX [3]
- Cray compiler usually fastest otherwise

Results: GPU



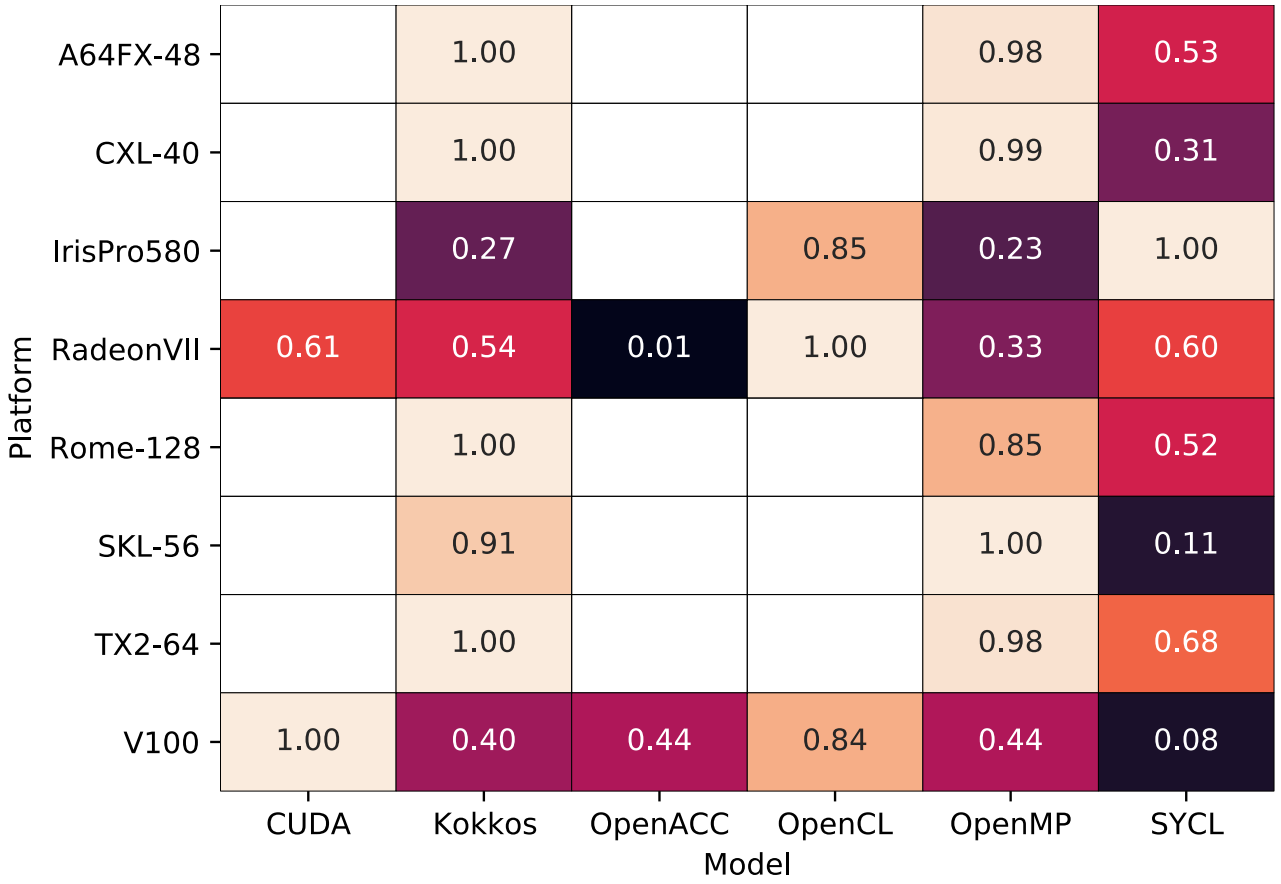
- Performance normalised to best result on each platform
- Bars missing where models not usable
  - OpenACC usable with GNU on Radeon VII, but only used a single thread
- No model performed best on *all* platforms
  - But OpenCL achieved >80% everywhere!
- Note that these are very different classes of devices…

Results: SYCL



- SYCL 1.2.1, using `sycl::nd_range`
  - Retains OpenCL's copy to local memory via `async_work_group_copy`
- Alternative implementation using `parallel_for` and `sycl::range`
  - Similar performance
- Code ran on GPUs unchanged from CPUs
  - On AMD and NVIDIA GPUs, hipSYCL is the only usable implementation

Performance Portability



- Heatmap shows relative performance on all programming models, normalised to the best result on each platform
  - Lighter colours show higher performance
  - Blank cells are currently impossible results
- OpenMP, Kokkos, and SYCL possible on all platforms
  - OpenMP and Kokkos within 15% of best performance on CPUs
  - Low-level programming faster on GPUs

Summary

- Modern programming models can perform on-par with traditional ones
  - Their platform support continues to grow
- True performance portability is still out of reach
  - No single version of the code achieved the best performance—or at least a high fraction of it
- Even for a small kernel, platform-specific optimisations and empirical tuning accounted for more than 30% of the performance
  - Enough to differentiate the best-performing implementation
- Immature drivers and ecosystem around SYCL was an obstacle
- Kokkos was reliable and lightweight

It's easy to apply for time on Isambard

- Please contact the Isambard PI, Prof. Simon McIntosh-Smith simonm at cs.bris.ac.uk, who will help you determine if Isambard will work for you. If it will, applying for an account is quick and easy.

- Small amounts of pump-priming time are available for free, to try porting, optimizing for Arm etc.

- Larger amounts of time for real science runs can be applied for via the regular EPSRC "Access to HPC" calls, or via some CCPs.

Thank you

- Reproducibility:
  - miniBUDE code: https://github.com/UoB-HPC/miniBUDE
  - Build and run scripts: https://github.com/UoB-HPC/performance-portability

- Bristol HPC Homepage: https://uob-hpc.github.io
  - Publications: https://uob-hpc.github.io/publications
  - Benchmarks: https://github.com/UoB-HPC/benchmarks

- Isambard: https://gw4-isambard.github.io/docs/index.html

University of BRISTOL

UKRI Engineering and Physical Sciences Research Council

GW4

References

[1] T. Deakin, A. Poenaru, T. Lin, and S. McIntosh-Smith, ‘Tracking Performance Portability on the Yellow Brick Road to Exascale’. Proceedings of the 2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2020. In press.

[2] S. McIntosh-Smith, J. Price, R. B. Sessions and A. A. Ibarra. ‘High performance in silico virtual drug screening on many-core processors’. In: The International Journal of High Performance Computing Applications 29.2 (2015), pp. 119–134. DOI: 10.1177/ 1094342014528252.

[3] Andrei Poenaru et al. ‘An Evaluation of the Fujitsu A64FX for HPC Applications ’. Cray User Group 2021. In press.