

# Overview of open platform programming methods for exascale computing

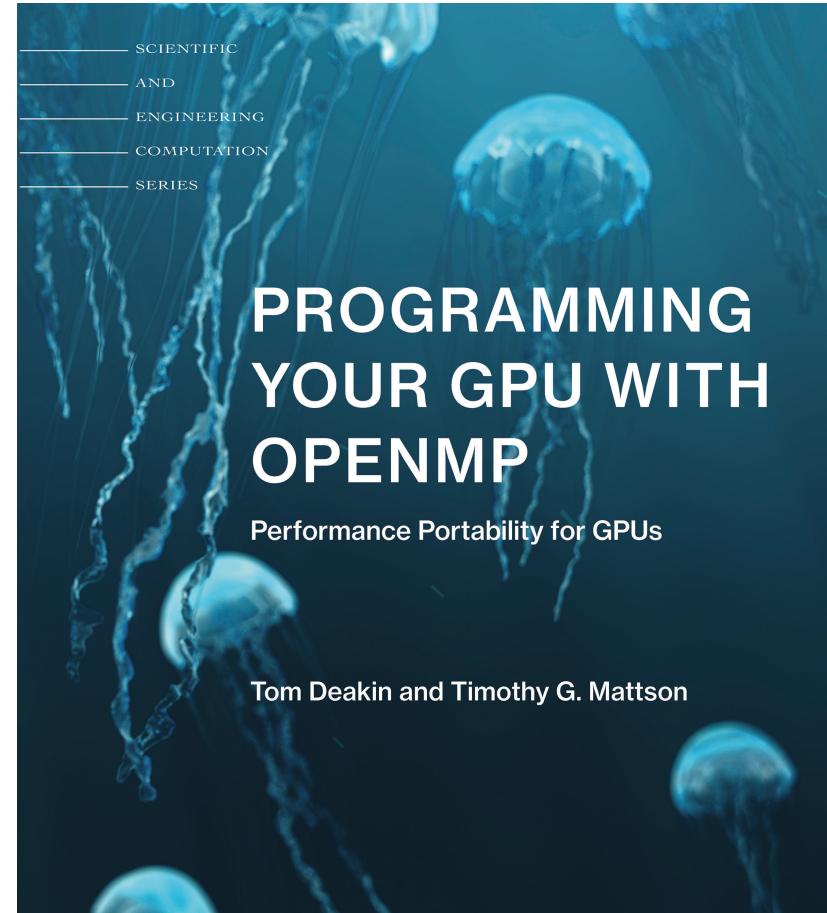
Dr Tom Deakin  
University of Bristol

Monday, June 24<sup>th</sup> 2024

# Some bias

- I wrote a book on Programming GPUs with OpenMP
- I am Chair of the SYCL Working Group for The Khronos Group

I am speaking today as an academic based on my research





# Tracking Performance Portability on the Yellow Brick Road to Exascale

Tom Deakin\*, Andrei Poenaru\*, Tom Lin\* and Simon McIntosh-Smith\*

\*Department of Computer Science, University of Bristol, UK

Email: {tom.deakin, andrei.poenaru, s.mcintosh-smith}@bristol.ac.uk

**Abstract**—With Exascale machines on our immediate horizon, there is a pressing need for applications to be made ready to best exploit these systems. However, there will be multiple paths to Exascale, with each system relying on processor and accelerator technologies from different vendors. As such, applications will be required to be portable between these different architectures, but it is also critical that they are efficient too. These double requirements for portability and efficiency begets the need for performance portability. In this study we survey the performance portability of different programming models, including the open standards OpenMP and SYCL, across the diverse landscape of Exascale and pre-Exascale processors from Intel, AMD, NVIDIA, Fujitsu, Marvell, and Amazon, together encompassing GPUs and CPUs based on both x86 and Arm architectures. We also take a historical view and analyse how performance portability has changed over the last year.

**Index Terms**—performance portability, programming models

## I. INTRODUCTION

Exascale-class supercomputers are on the immediate horizon,

To further enable the development of performance-portable programs, in this study we update and greatly expand our earlier, wide-reaching study on performance portability [1]. We include the latest and greatest architectures, including for the first time the Arm-based Fujitsu A64FX processor, the NVIDIA Ampere GPU, and Intel GPUs. Thus, this study spans the processor architecture design space of the first Exascale machines.

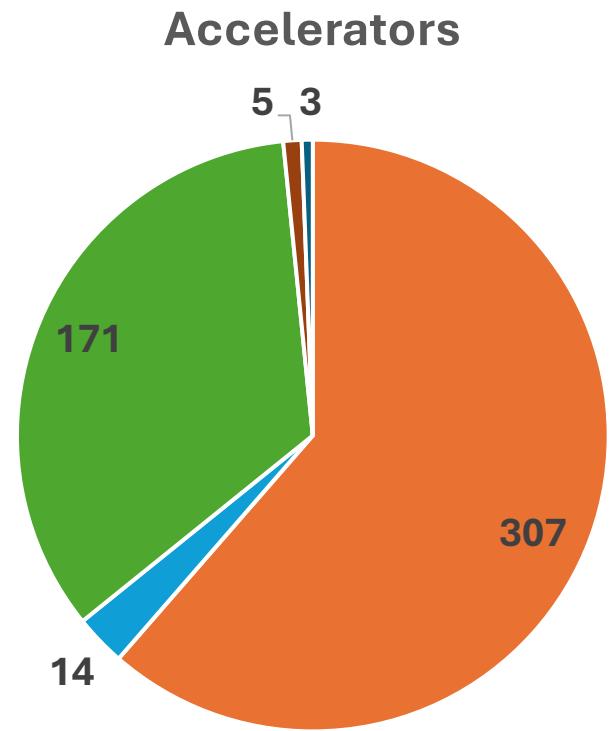
As this work is an expansion and update of the 2019 study, we are able to begin to explore the historical perspective for how performance portability changes over time. The ecosystems surrounding each of the processors have had time to expand and mature, and therefore by refreshing many of the results from the original study in 2019 we can track the progress of support, performance, and performance portability.

In this update, for the first time we include results from applications written in SYCL. The applications we include are all open source and were ported for the purposes of this study thus representing a contribution to the community in

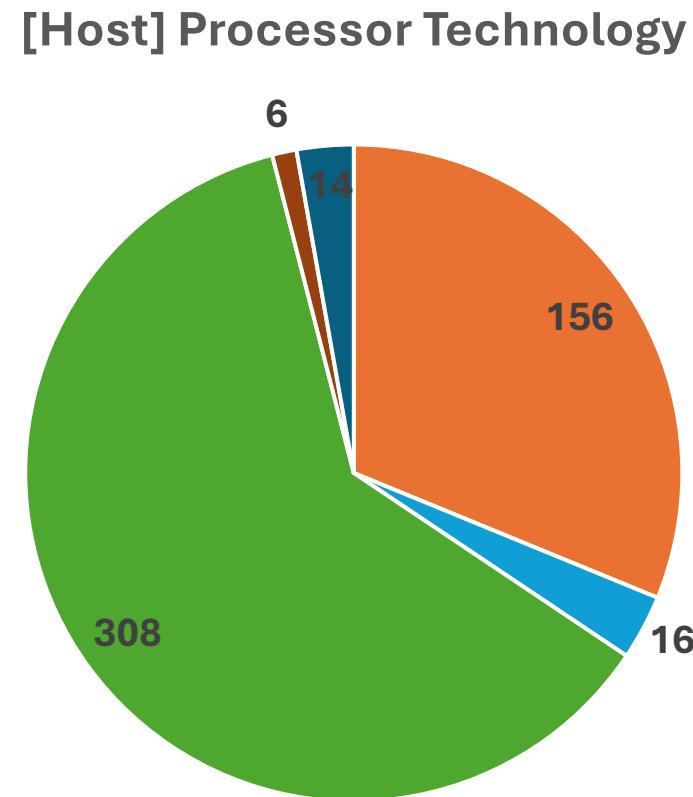


WE'RE NOT IN  
KANSAS ANYMORE.

# Homogeneous-heterogenous world of Top 500



■ None ■ AMD ■ NVIDIA ■ Intel ■ Other



■ AMD Zen 2-4 ■ Arm ■ Intel (8 generations) ■ Power ■ Other

Data: TOP500 June 2024

Updated version of chart from: Deakin, Cownie, Lin, McIntosh-Smith,  
*Heterogeneous Programming for the Homogeneous Majority*  
<https://doi.org/10.1109/P3HPC56579.2022.00006>



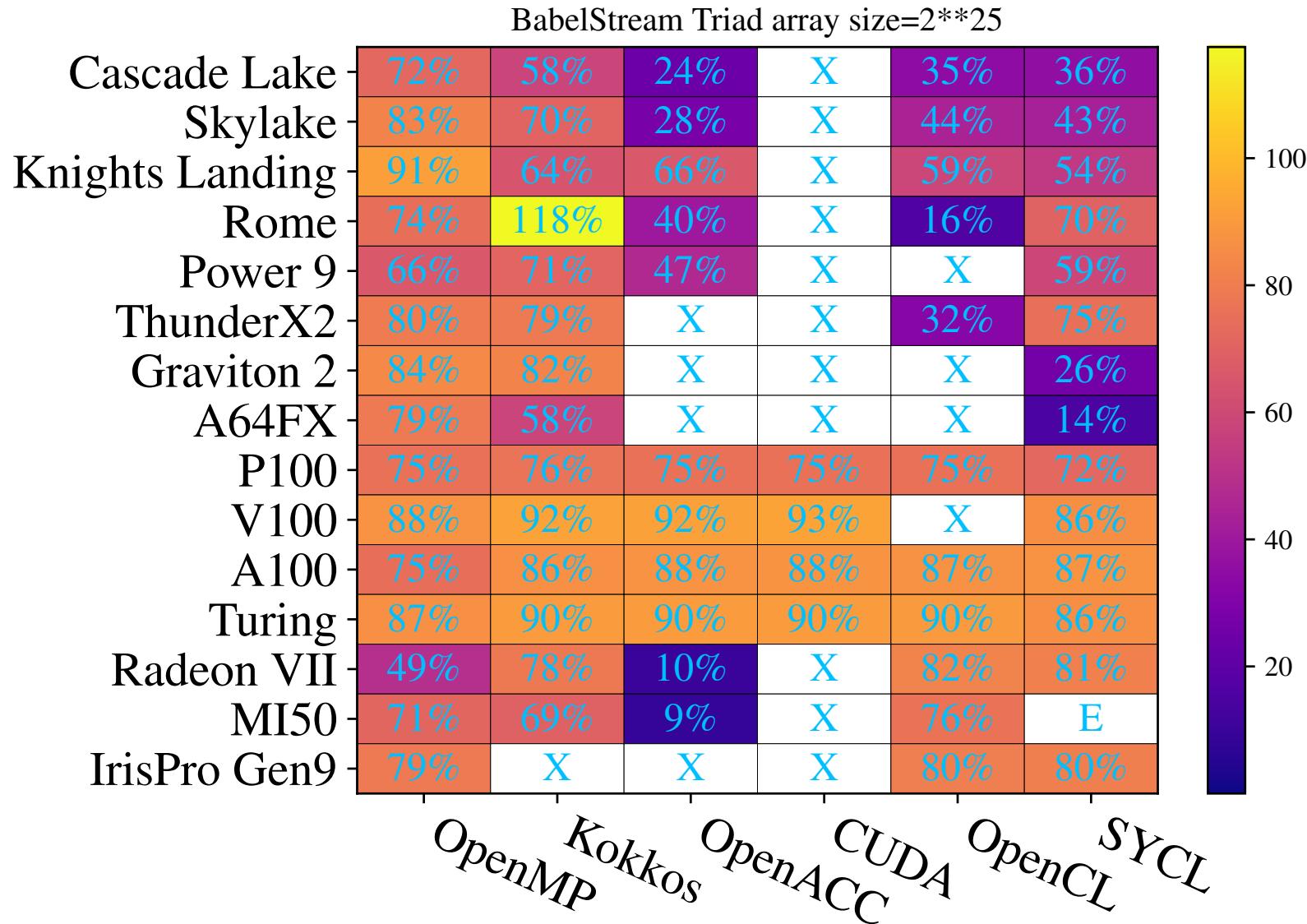
# Bristol definition of performance portability

“A code is performance portable if it can achieve a similar fraction of peak hardware performance on a range of different target architectures”.

- Needs to be a good fraction of best achievable (i.e., hand optimised).
- Range of architectures depends on your goal, but important to allow for future developments.
- Most interested in consistency of distribution of performance across systems
- Aligns with PP metric from Pennycook, et al.

From Pennycook, Sewall, Jacobsen, Deakin, McIntosh-Smith  
*Navigating Performance, Portability, and Productivity*  
<https://doi.org/10.1109/MCSE.2021.3097276>

# Back to the beginning of the yellow brick road



# Heterogeneous programming model abstractions

Device discovery  
and control

Data location and  
movement in  
discrete memory  
spaces

Expressing  
concurrent and  
parallel work

# OpenMP = OpenMP 1 + OpenMP 4/5 ( +tasks) ?

```
#pragma omp parallel for  
for (int i = 0; i < N; ++i) {  
    C[i] = A[i] + B[i];  
}  
  
#pragma omp target enter data \  
map(alloc: C[:N]) \  
map(to: A[:N], B[:N])  
  
#pragma omp target  
#pragma omp loop  
for (int i = 0; i < N; ++i) {  
    C[i] = A[i] + B[i];  
}  
  
#pragma omp target exit data \  
map(from: C[:N])
```

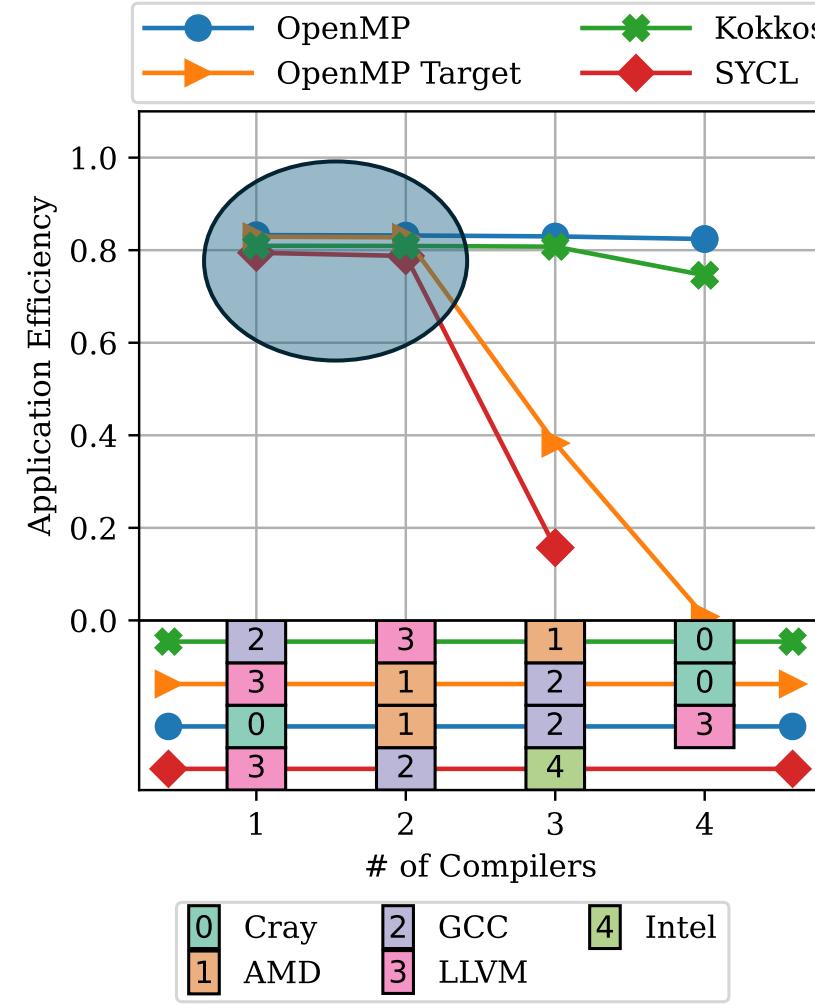
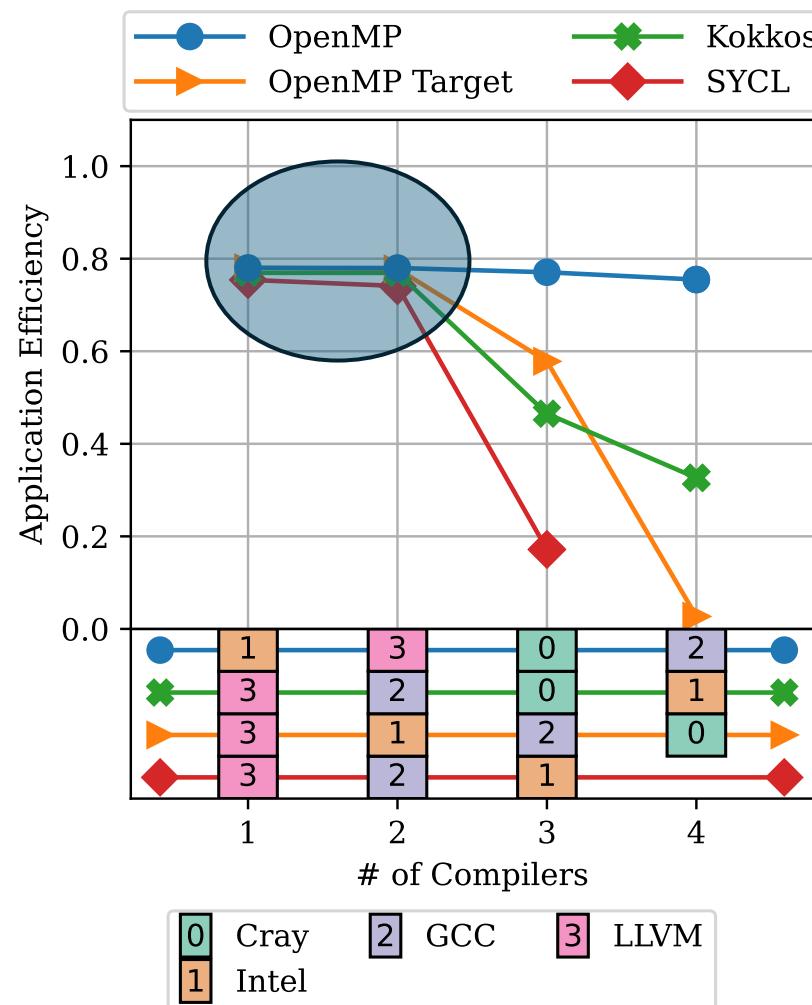
Can you just write the target version and get good performance?

<https://doi.org/10.1109/P3HPC56579.2022.00006>

# Icelake

# BabelStream

# Milan



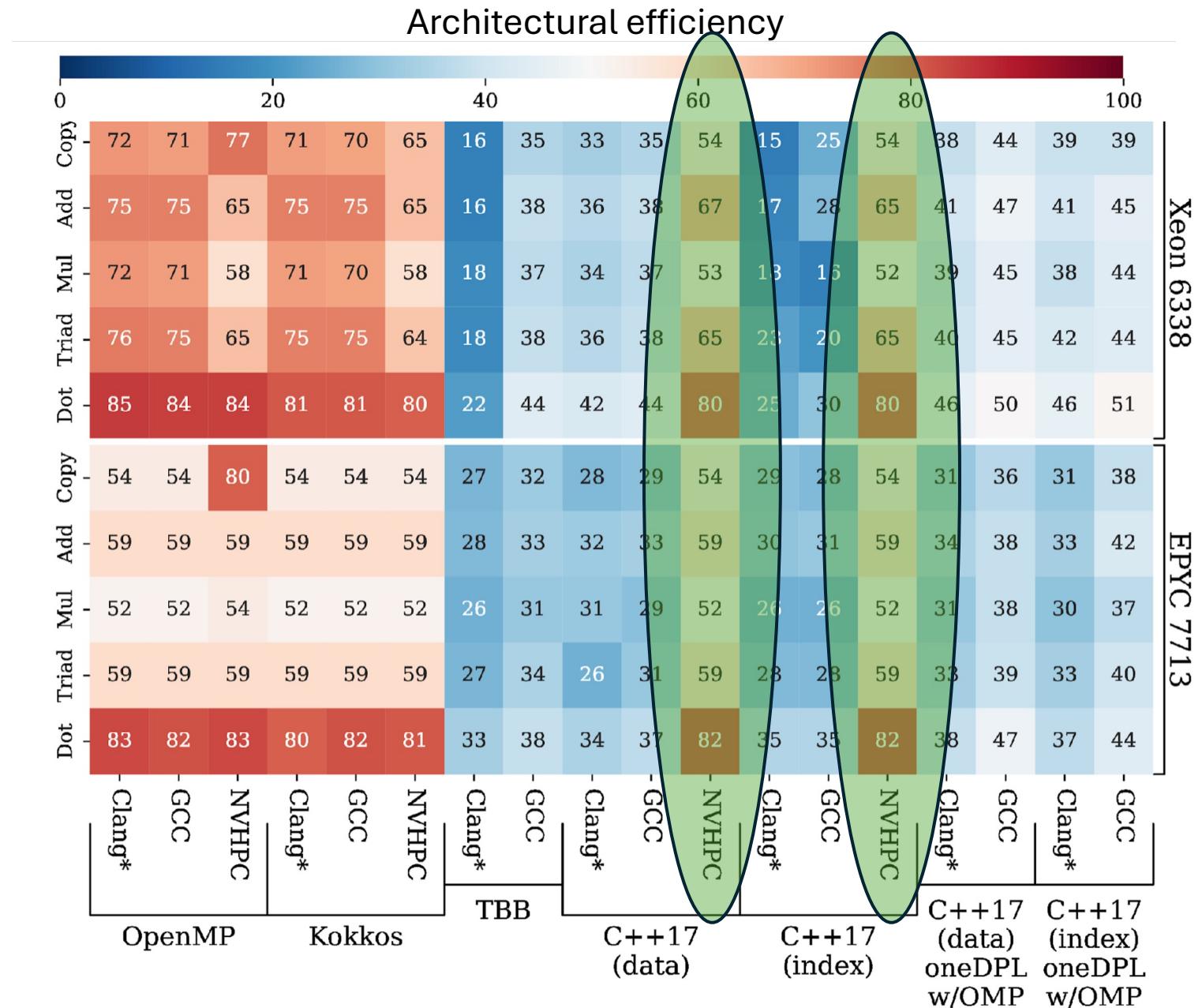
“Then, if you don't mind,  
I'll go with you,” said the  
Lion, “for my life is simply  
unbearable without a bit  
of courage.”

from The Wonderful Wizard of Oz  
by L. Frank Baum



	C++17 StdPar 	SYCL 	OpenMP target 	HIP/CUDA 	OpenCL 	Kokkos 	Julia 
Code portability	Compiler/ Compiler flags	Compiler/ Compiler flags	Compiler/ Compiler flags	Not portable	Runtime	Compiler/ Compiler flags	Partial* * library dependent
Device portability	CPU,GPU	CPU,GPU,FPGA	CPU,GPU	GPU (CPU via third-party impl.)	CPU,GPU, FPGA	CPU,GPU	CPU,GPU
Supported platform	Intel/AMD/NVIDIA	Intel/AMD/NVIDIA	Intel/AMD/NVIDIA	Vendor-only	Intel/AMD/NVIDIA	Intel/AMD/NVIDIA	Intel/AMD/NVIDIA
Format	Single-source	Single-source	Single-source	Single-source	Multi-source	Single-source	Single-source
Data movement	Implicit: USM	Explicit: accessors Implicit: USM	Explicit: pragmas Implicit: USM	Explicit: vendor API Implicit: USM	Explicit: buffers Implicit: SVM	Explicit: views	Explicit: library API
Traversal	std::for_each std::for_each_n std::transform	queue.submit([&](auto &h) { h.parallel_for(...); });	# OpenMP >= 5.0 omp loop omp target teams distributed \ parallel for	_global_ void kernel(...) ... // ... kernel<<<N>>>(...)	(> 10 lines, in two files)	Kokkos::parallel_for	Yes* *library dependent
Reduction	std::transform_reduce std::reduce std::accumulate	queue.submit([&](sycl::handler &h) { h.parallel_for( sycl::reduction(...),... );	omp reduction(inscan,...) { omp scan inclusive(...) omp scan exclusive(...) }	(> 10 lines, “roll your own”)	(> 10 lines, “roll your own”)	Kokkos::parallel_reduce	Yes* *library dependent
Task asynchronous/ scheduling	No control (Ongoing proposals)	Command queues	#pragma omp nowait/depends(...) (Blocking by default)	Streams	Command queues	Futures (C++ like)	Yes* *library dependent
Affinity	No control (Ongoing proposals)	Device API	#pragma omp device(...) (Host is also a device)	Vendor API	Device API	Device API	Yes* *library dependent

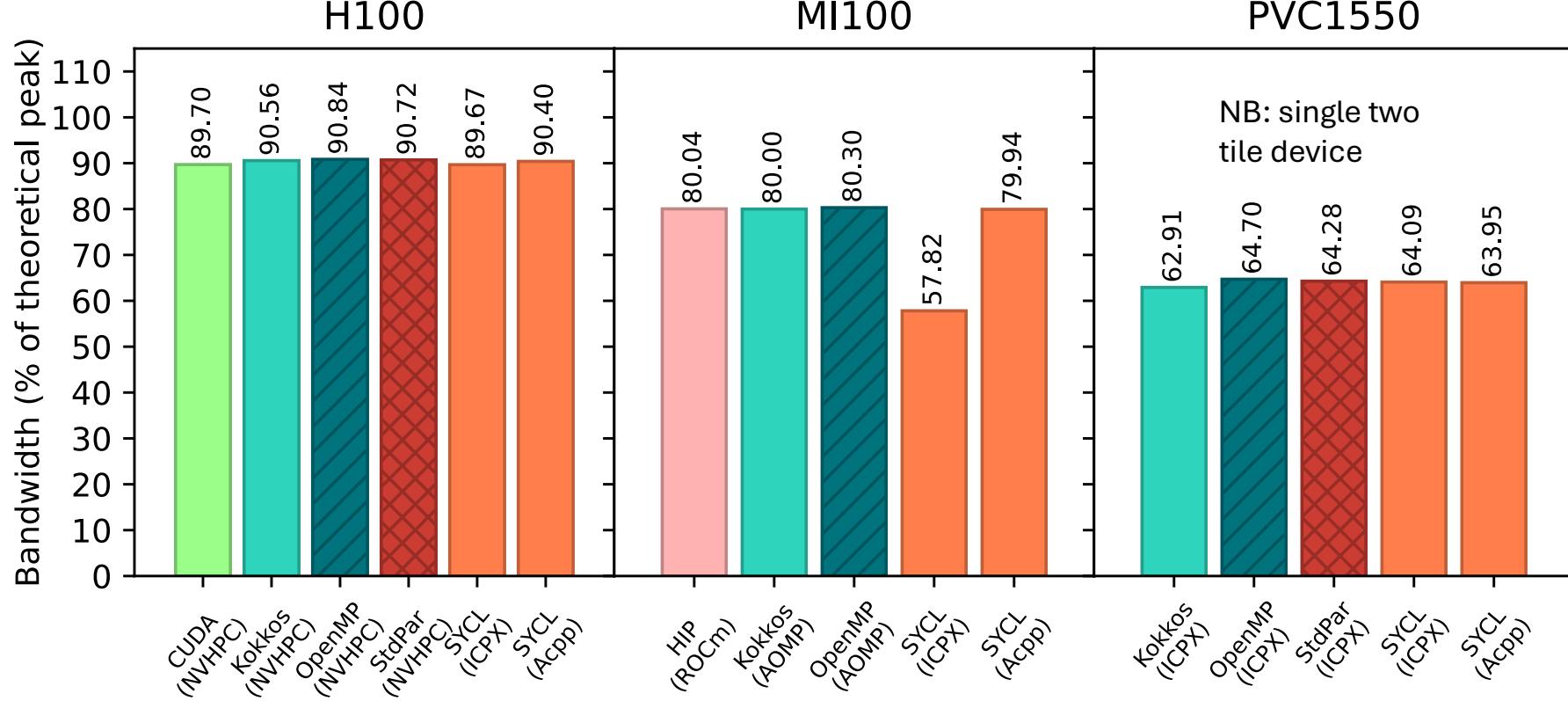
# ISO C++ parallel algorithms on x86 CPUs



Lin, Deakin, McIntosh-Smith

Evaluating ISO C++ Parallel Algorithms on Heterogeneous HPC Systems

<https://doi.org/10.1109/PMBS56514.2022.00009>



On latest GPUs from NVIDIA, AMD, and Intel, performance portability for BabelStream possible in most mainstream performance portable programming models:

- ISO C++ stdpar, OpenMP, SYCL, Kokkos
- Same performance as “native” CUDA/HIP

# FORTRAN

- We also looked at parallelism in Fortran (DO CONCURRENT)
- Lots of recent progress in this space, and more to explore.
- See Hammond, Deakin, Cownie, McIntosh-Smith, *Benchmarking Fortran DO CONCURRENT on CPUs and GPUs Using BabelStream*, <https://doi.org/10.1109/PMBS56514.2022.00013>

Same conclusion as C++ paper.



"I shall take the heart," returned the Tin Woodman; "for brains do not make one happy, and happiness is the best thing in the world."

from The Wonderful Wizard of Oz  
by L. Frank Baum



“I am everywhere,”  
answered the Voice,  
“but to the eyes of  
common mortals I am  
invisible.”

from *The Wonderful Wizard of Oz*  
by L. Frank Baum

See Doerfert, et al. *Breaking the Vendor Lock: Performance Portable Programming through OpenMP as Target Independent Runtime Layer*,  
<https://doi.org/10.1145/3559009.3569687>



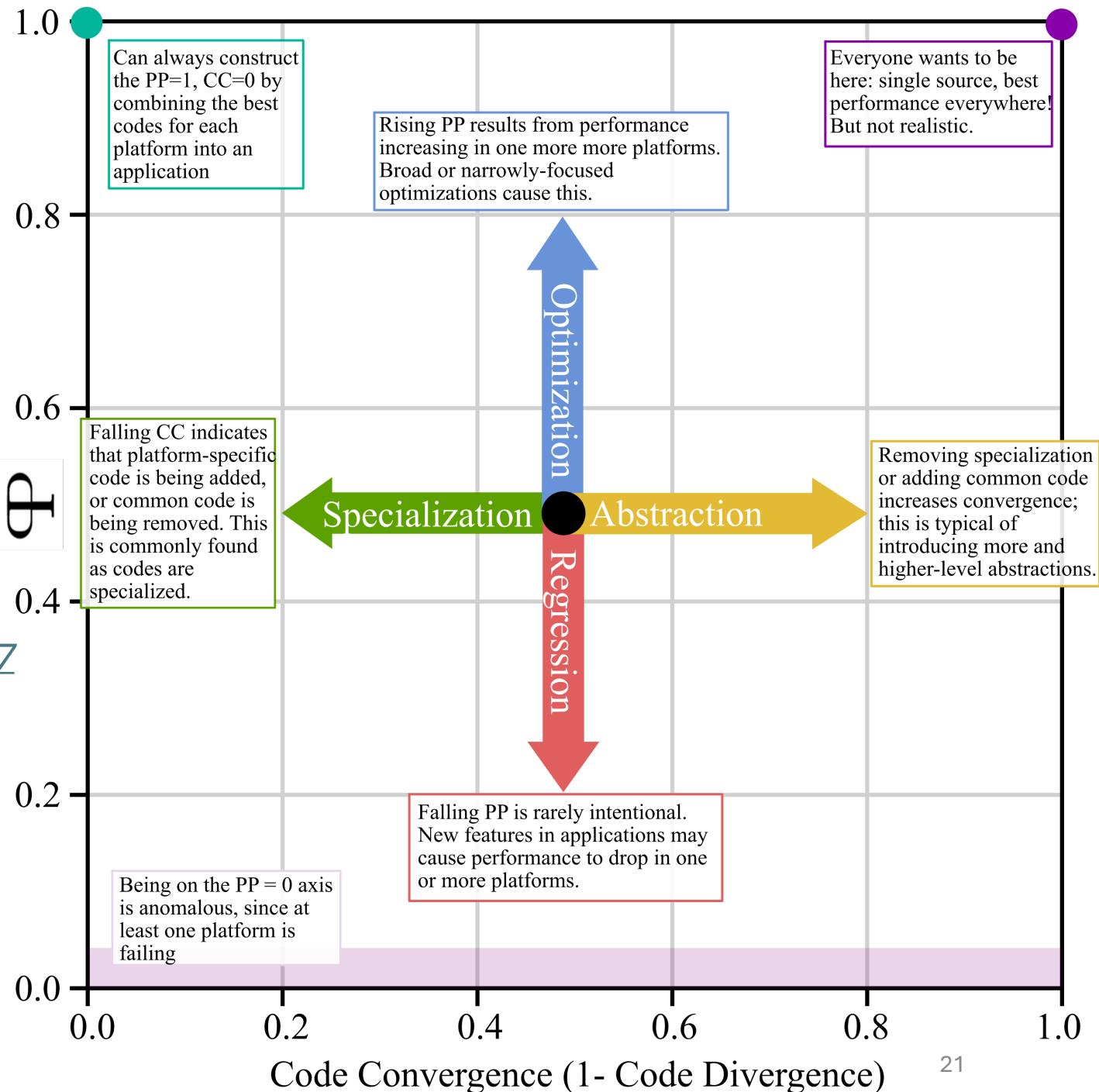
“But I do not want people to call me a fool, and if my head stays stuffed with straw instead of with brains, as yours is, how am I ever to know anything?”

from *The Wonderful Wizard of Oz*  
by L. Frank Baum

# The necessity of specialisation

From Pennycook, Sewall, Jacobsen, Deakin, McIntosh-Smith  
*Navigating Performance, Portability, and Productivity*

<https://doi.org/10.1109/MCSE.2021.3097276>



# The Yellow Brick Road to Productive Performance Portability is paved with OpenMP and SYCL

**OPENMP®**

Shared-memory Parallelizing Road

**SYCL**

Single-source Heterogeneous programming

# Related performance portability papers

Tuomas, Christidi, Giordano, Dubrovska, Quinn, Maynard, Case, Olgu, and Deakin. “Principles for Automated and Reproducible Benchmarking.” In First International Workshop on HPC Testing and Evaluation of Systems, Tools, and Software. IEEE, 2023.

<https://doi.org/10.1145/3624062.3624133>

Deakin, T, James C, Lin, W.C., and McIntosh-Smith, S. “Heterogeneous Programming for the Homogeneous Majority.” In International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2022.

<https://doi.org/10.1109/P3HPC56579.2022.00006>

Hammond, J.R., Deakin, T, Cownie, J. and McIntosh-Smith, S. “Benchmarking Fortran DO CONCURRENT on CPUs and GPUs Using BabelStream.” In International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2022.

<https://doi.org/10.1109/PMBS56514.2022.00013>

Lin, W.C, Deakin T, and McIntosh-Smith S. “Evaluating ISO C++ Parallel Algorithms on Heterogeneous HPC Systems.” In International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2022.

<https://doi.org/10.1109/PMBS56514.2022.00009>

Deakin, T and McIntosh-Smith S. “Evaluating the Performance of HPC-Style SYCL Applications.” In International Workshop on OpenCL and SYCLCon (IWOC/SYCLCon). ACM, 2020.

<https://doi.org/10.1145/3388333.3388643>

Pennycook, S. J., Sewall, J. D., Jacobsen, D. W., Deakin,T and McIntosh-Smith, S. “Navigating Performance, Portability and Productivity.” Computing in Science and Engineering, 2021.

<https://doi.org/10.1109/MCSE.2021.3097276>

Deakin, T., Poenaru, A., Lin, T., and McIntosh-Smith, A., “Tracking Performance Portability on the Yellow Brick Road to Exascale.: In International Workshop on Performance Portability and Productivity in HPC (P3HPC), 2020.

<https://doi.org/10.1109/P3HPC51967.2020.00006>