University of BRISTOL

Performance Portability for Next-Generation Heterogeneous Systems

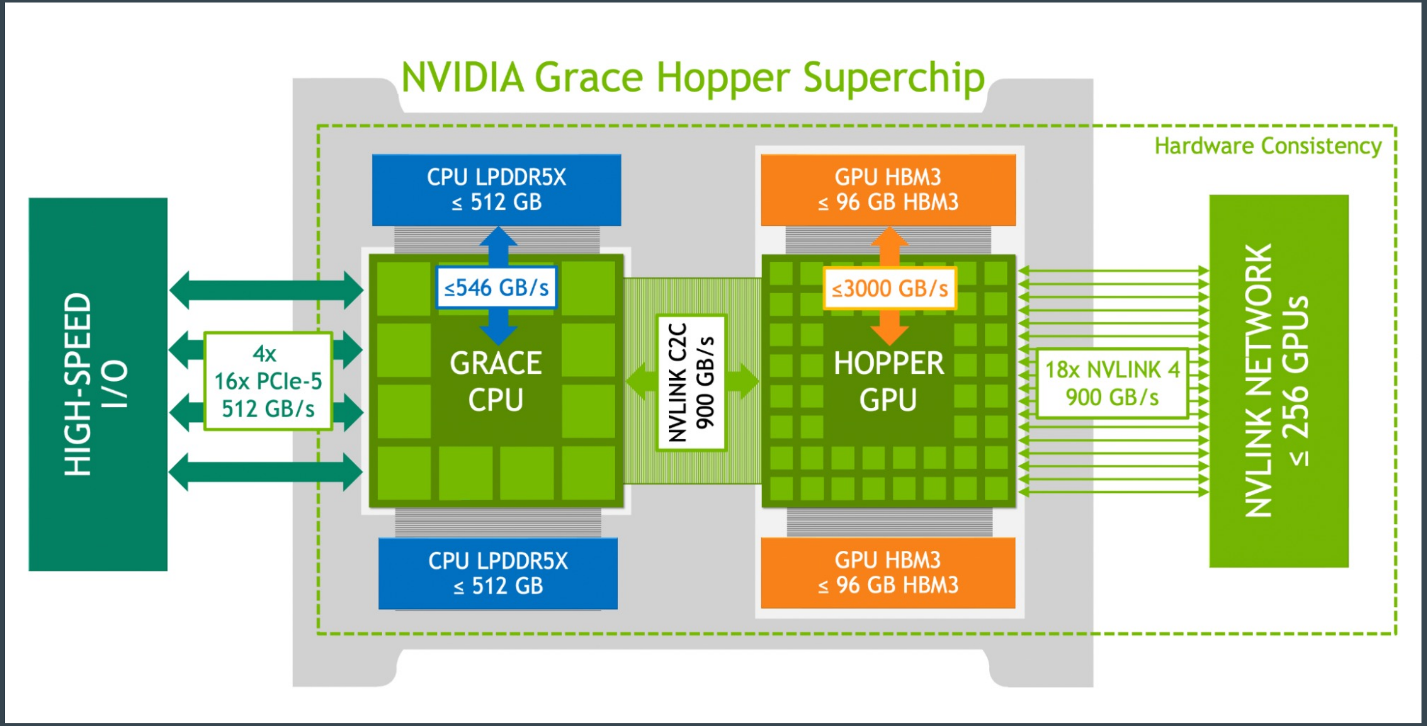
Dr Tom Deakin

Rank	System	Accelerator
1	Frontier	✅
2	Supercomputer Fugaku	❌
3	LUMI	✅
4	Leonardo	✅
5	Summit	✅
6	Sierra	✅
7	Sunway TaihuLight	❌
8	Perlmutter	✅
9	Selene	✅
10	Tianhe-2A	✅

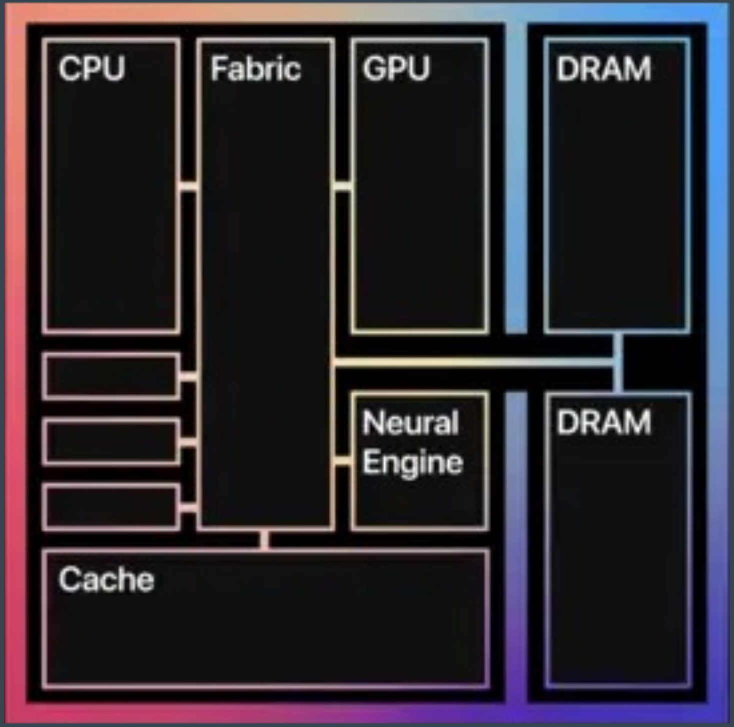
Source: TOP500 November 2022

Latency	Throughput
“Complex” cores	More “simple” cores
Instruction Level Parallelism	Very wide SIMD
Deep cache hierarchy	Fast context switching
NUMA	Programable memory hierarchy
Wide SIMD	Latest memory technology

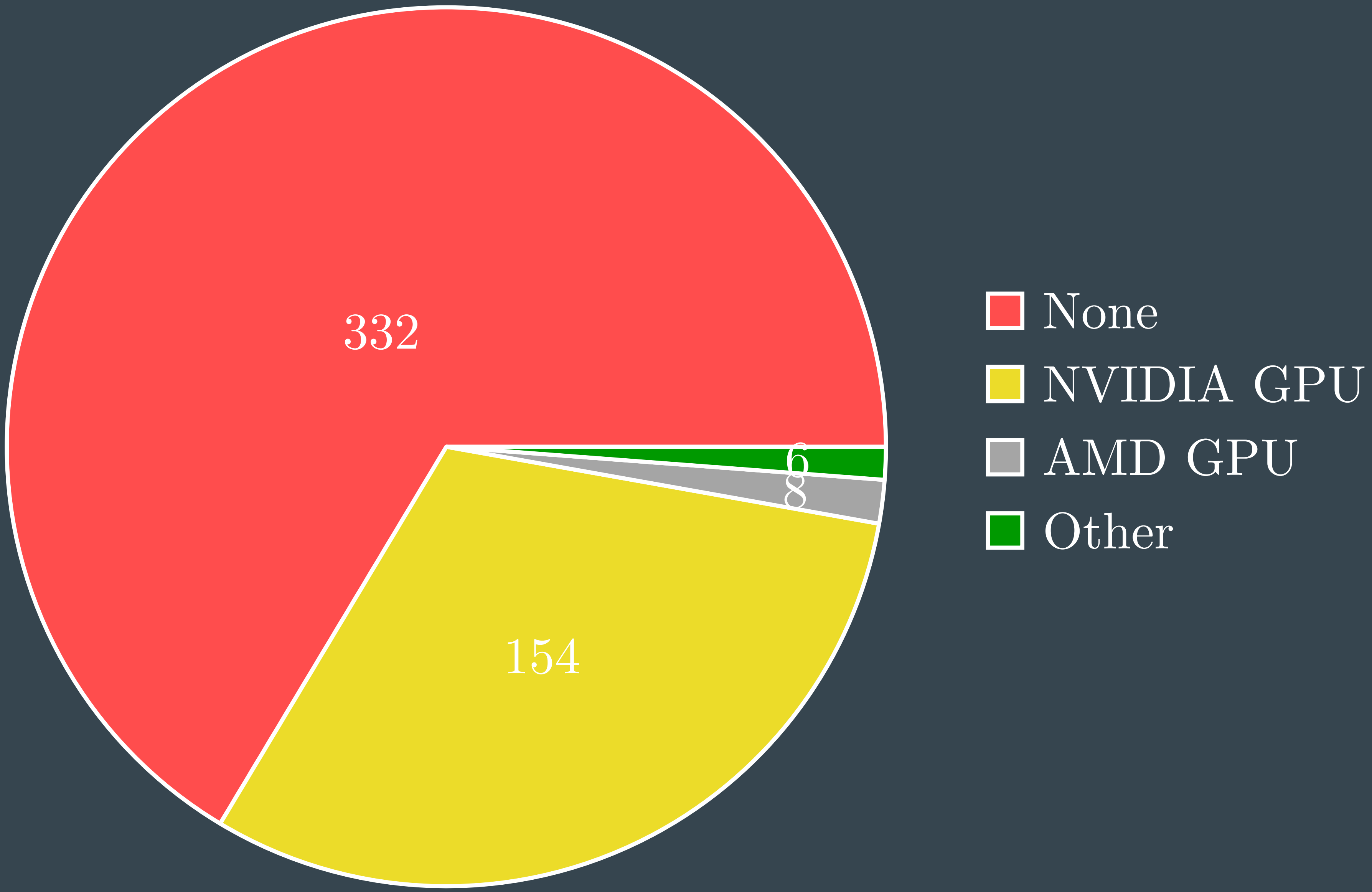
NVIDIA Grace-Hopper



Apple M1



Images belong to their respective owners



Data: TOP500 June 2022
Graph: doi.org/10.1109/P3HPC56579.2022.00006

Tension between migrating to next system (which may be GPUs), and keeping running on current system

Performance, Portability, and Productivity

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

Problem

Application

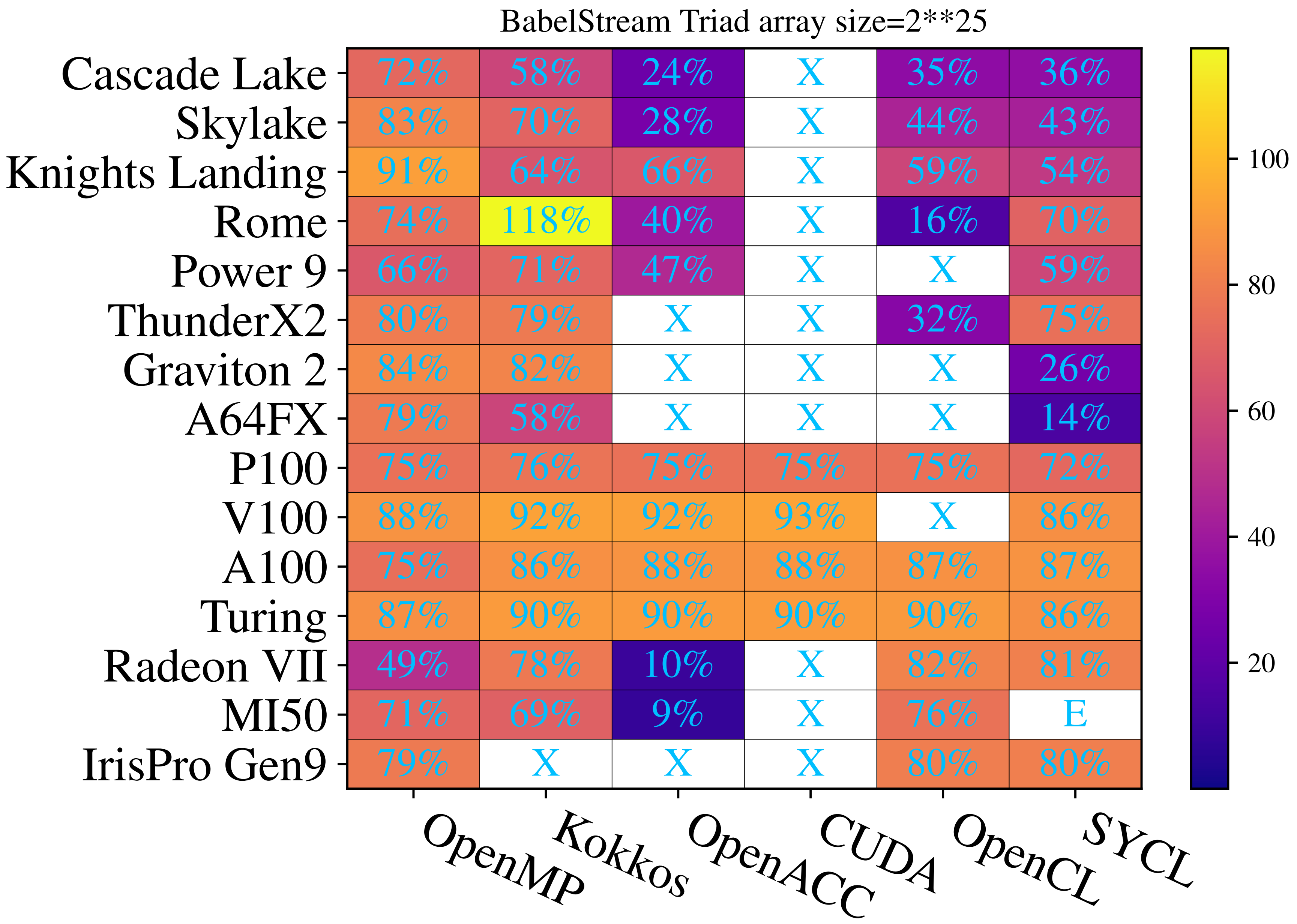
Platform

Efficiency

More details in doi.org/10.1109/P3HPC51967.2020.00007

$$\Phi(a,p,H) = \begin{cases} \dfrac{|H|}{\displaystyle\sum_{i \in H} \frac{1}{e_i(a,p)}} & \text{if, } \forall i \in H \\ & e_i(a,p) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

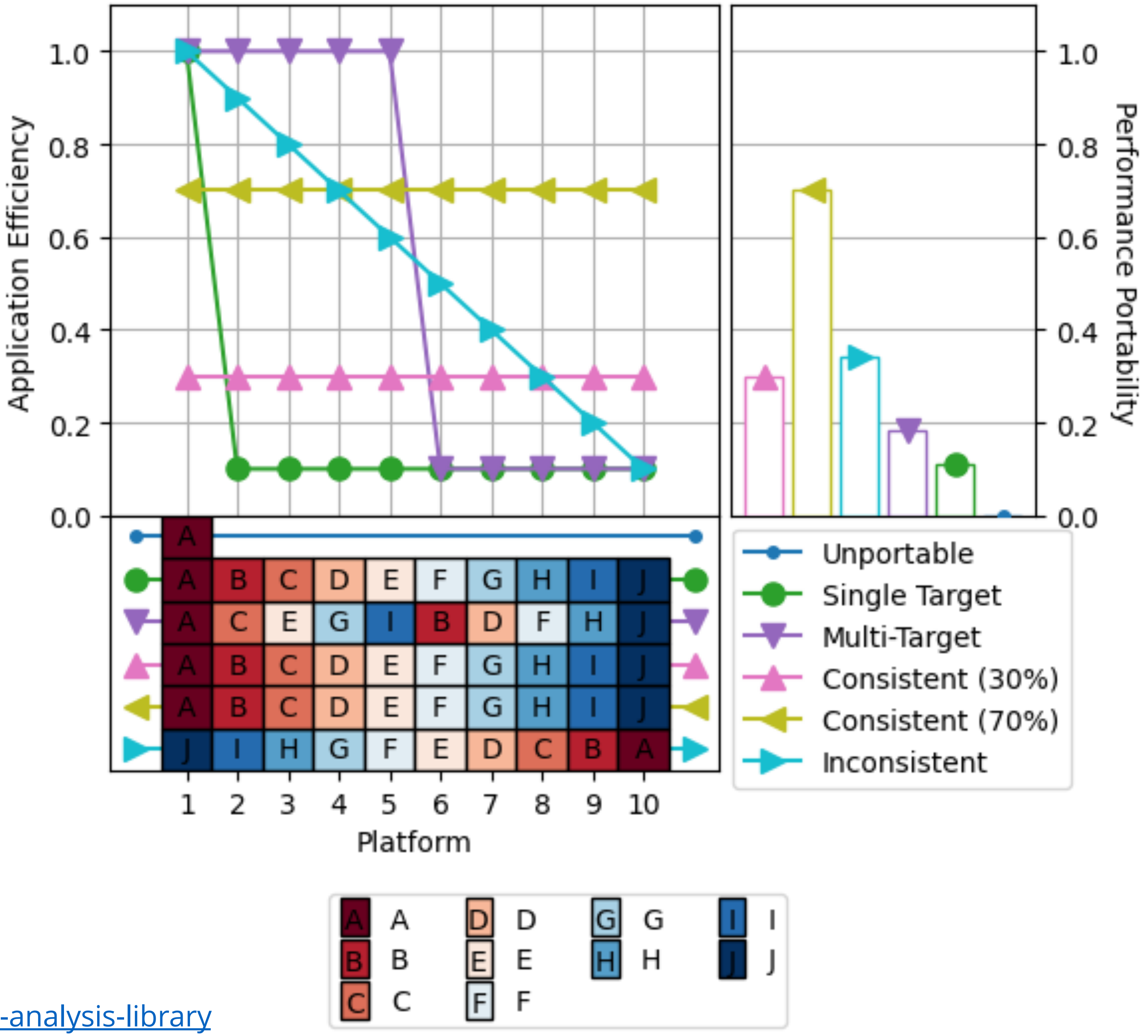
From Pennycook, Sewall and Lee: doi.org/10.1016/j.future.2017.08.007



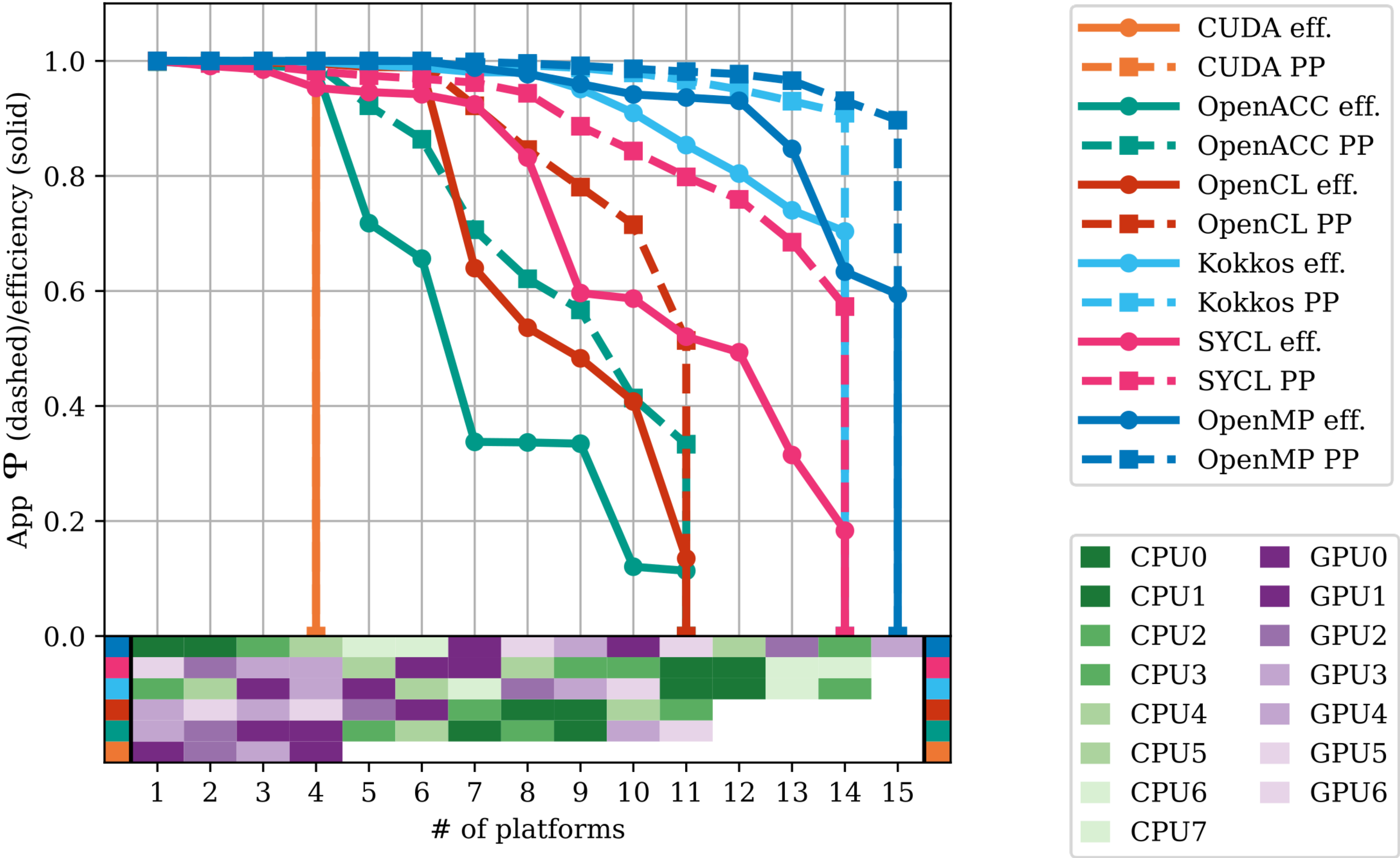
BabelStream Triad array size=2**25

	OpenMP	Kokkos	OpenACC	CUDA	OpenCL	SYCL
Cascade Lake	72%	58%	24%	X	35%	36%
Skylake	83%	70%	28%	X	44%	43%
Knights Landing	91%	64%	66%	X	59%	54%
Rome	74%	118%	40%	X	16%	70%
Power 9	66%	71%	47%	X	X	59%
ThunderX2	80%	79%	X	X	32%	75%
Graviton 2	84%	82%	X	X	X	26%
A64FX	79%	58%	X	X	X	14%
P100	75%	76%	75%	75%	75%	72%
V100	88%	92%	92%	93%	X	86%
A100	75%	86%	88%	88%	87%	87%
Turing	87%	90%	90%	90%	90%	86%
Radeon VII	49%	78%	10%	X	82%	81%
MI50	71%	69%	9%	X	76%	E
IrisPro Gen9	79%	X	X	X	80%	80%

From doi.org/10.1109/P3HPC51967.2020.00006

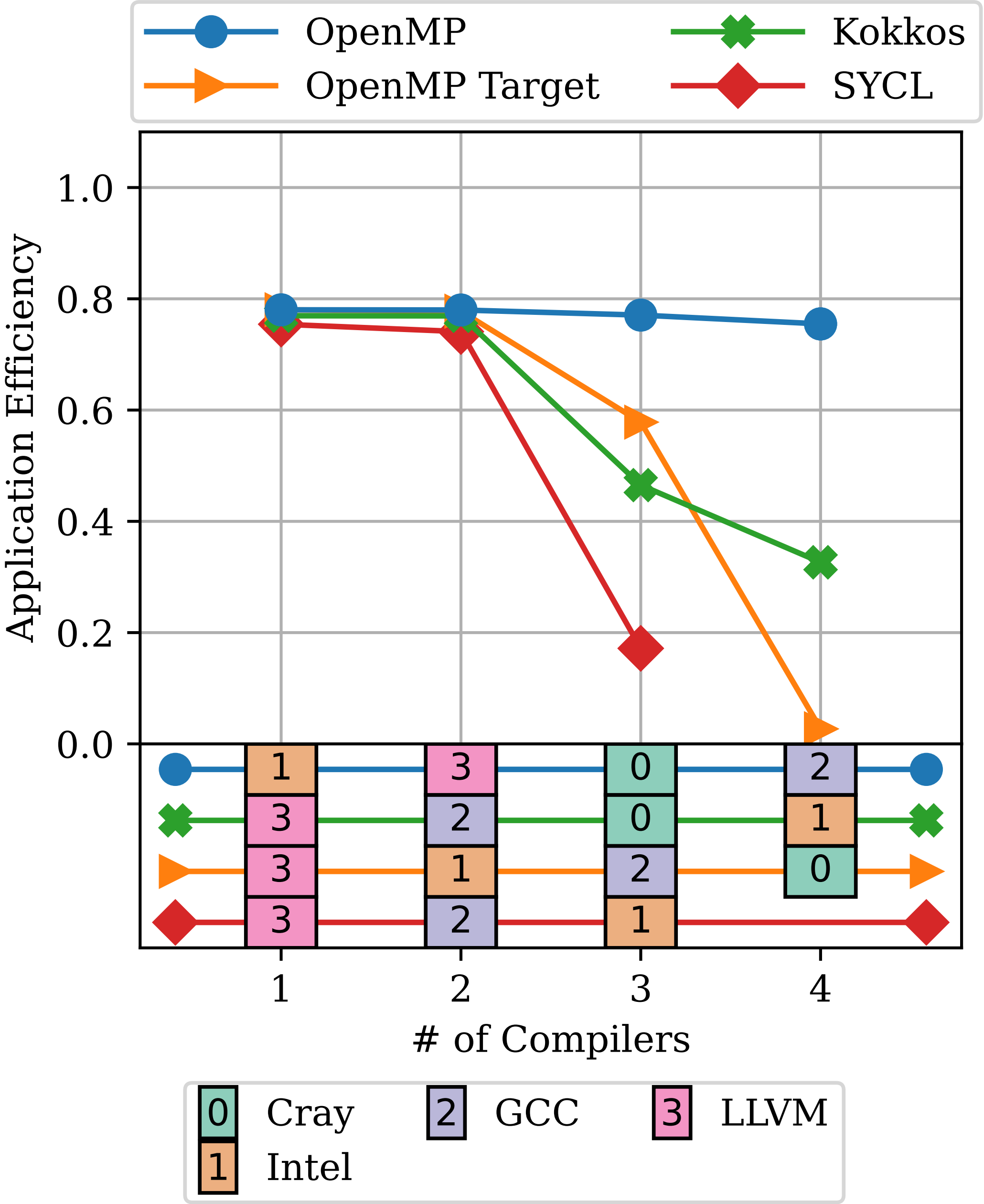


From https://intel.github.io/p3-analysis-library
Based on doi.org/10.1109/MCSE.2021.3097276



From doi.org/10.1109/MCSE.2021.3097276

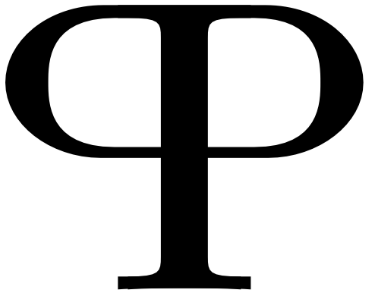
BabelStream

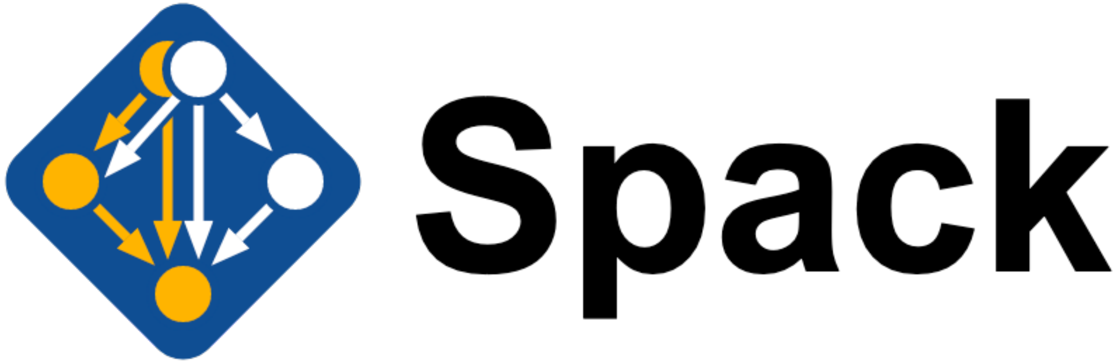


From doi.org/10.1109/P3HPC56579.2022.00006

The University of Bristol is an Intel oneAPI Center of Excellence helped support this work.
This work was supported by the Engineering and Physical Sciences Research Council ASiMoV project [EP/S005072/1].

https://github.com/ukri-excalibur/excalibur-tests





Logos belong to their respective owners

This work was supported by the Engineering and Physical Sciences Research Council as part of ExCALIBUR Hardware & Enabling Software [EP/X031829/1]

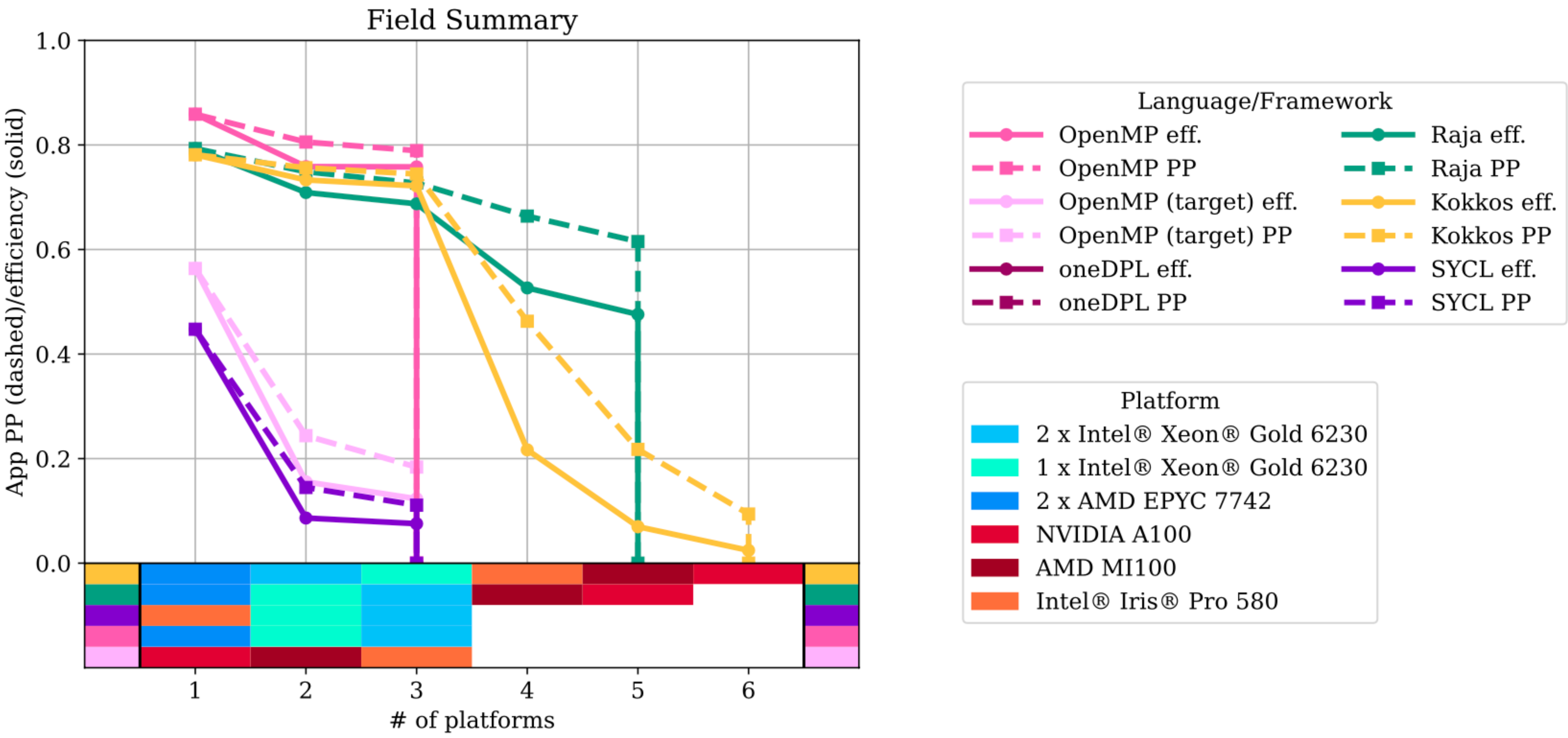
https://github.com/uob-hpc/babelstream

Device discovery and control

Data location and movement in discrete memory spaces

Expressing concurrent and parallel work

Logos belong to their respective owners

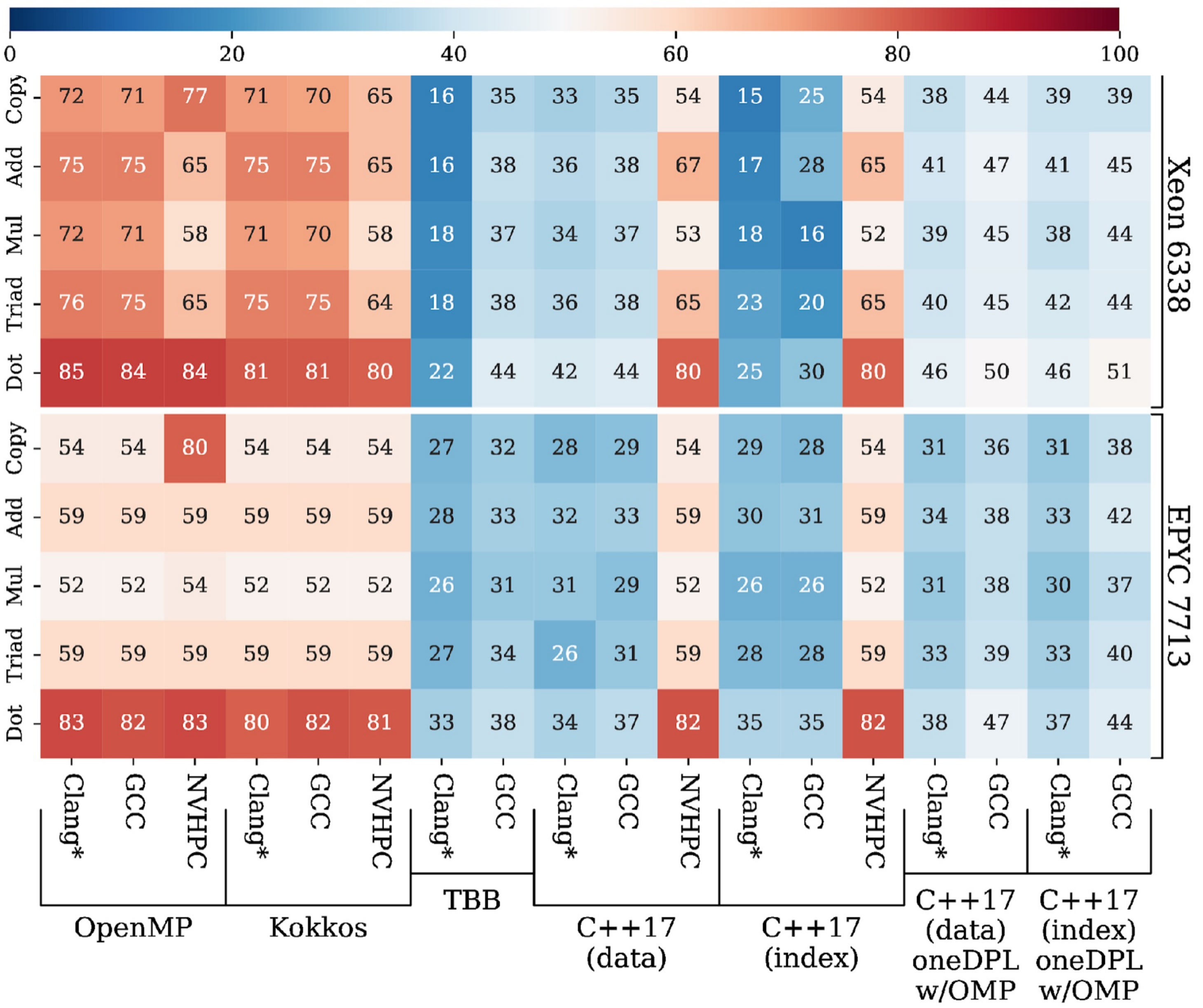


From doi.org/10.1109/P3HPC54578.2021.00007

This work was supported by the Engineering and Physical Sciences Research Council ASiMoV project [EP/S005072/1].

x86 CPU

Architectural efficiency

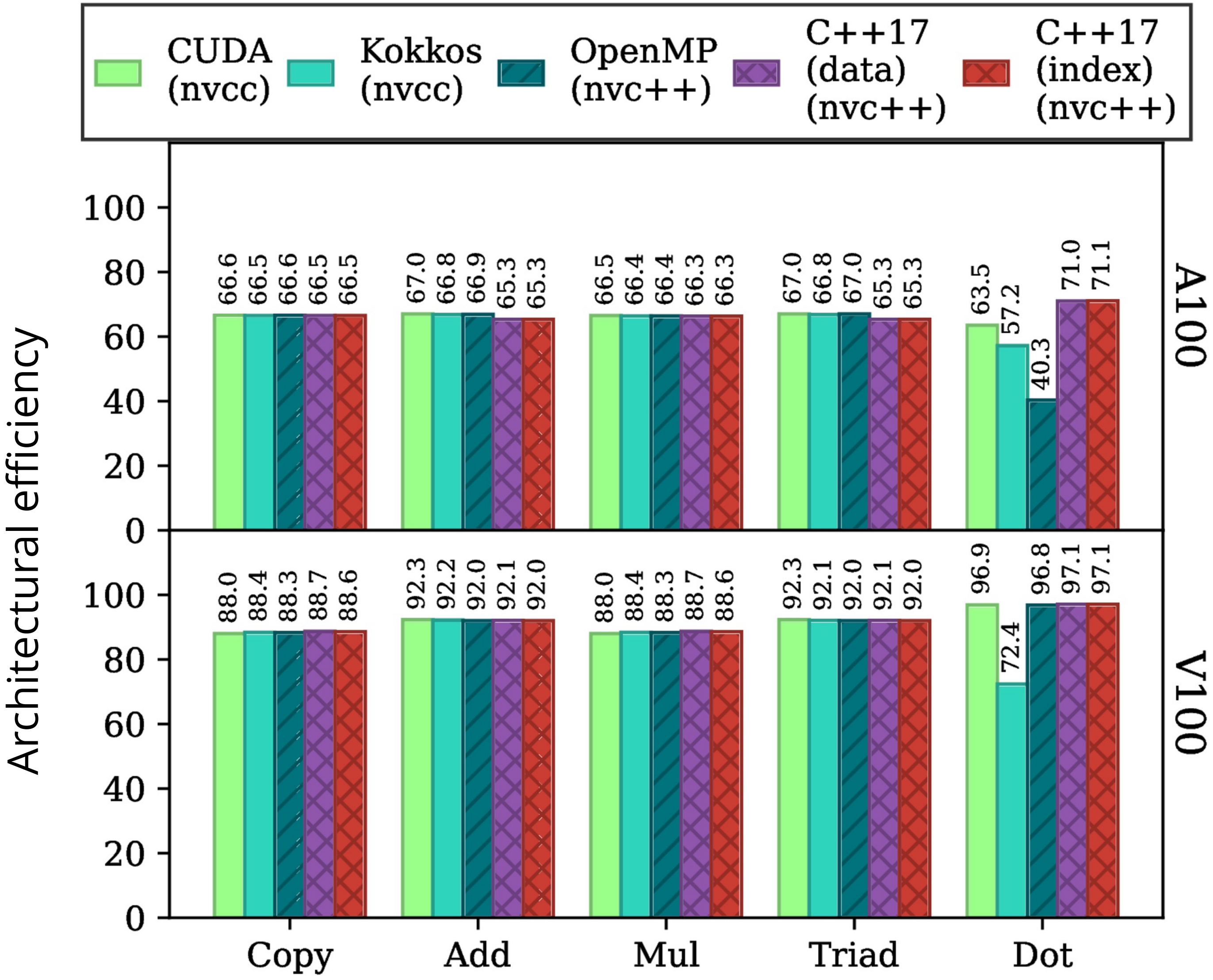


		OpenMP			Kokkos			TBB		C++17 (data)			C++17 (index)			C++17 (data) oneDPL w/OMP		C++17 (index) oneDPL w/OMP	
		Clang*	GCC	NVHPC	Clang*	GCC	NVHPC	Clang*	GCC	Clang*	GCC	NVHPC	Clang*	GCC	NVHPC	Clang*	GCC	Clang*	GCC
Xeon 6338	Copy	72	71	77	71	70	65	16	35	33	35	54	15	25	54	38	44	39	39
	Add	75	75	65	75	75	65	16	38	36	38	67	17	28	65	41	47	41	45
	Mul	72	71	58	71	70	58	18	37	34	37	53	18	16	52	39	45	38	44
	Triad	76	75	65	75	75	64	18	38	36	38	65	23	20	65	40	45	42	44
	Dot	85	84	84	81	81	80	22	44	42	44	80	25	30	80	46	50	46	51
EPYC 7713	Copy	54	54	80	54	54	54	27	32	28	29	54	29	28	54	31	36	31	38
	Add	59	59	59	59	59	59	28	33	32	33	59	30	31	59	34	38	33	42
	Mul	52	52	54	52	52	52	26	31	31	29	52	26	26	52	31	38	30	37
	Triad	59	59	59	59	59	59	27	34	26	31	59	28	28	59	33	39	33	40
	Dot	83	82	83	80	82	81	33	38	34	37	82	35	35	82	38	47	37	44

From doi.org/10.1109/PMBS56514.2022.00009

The University of Bristol is an Intel oneAPI Center of Excellence helped support this work.

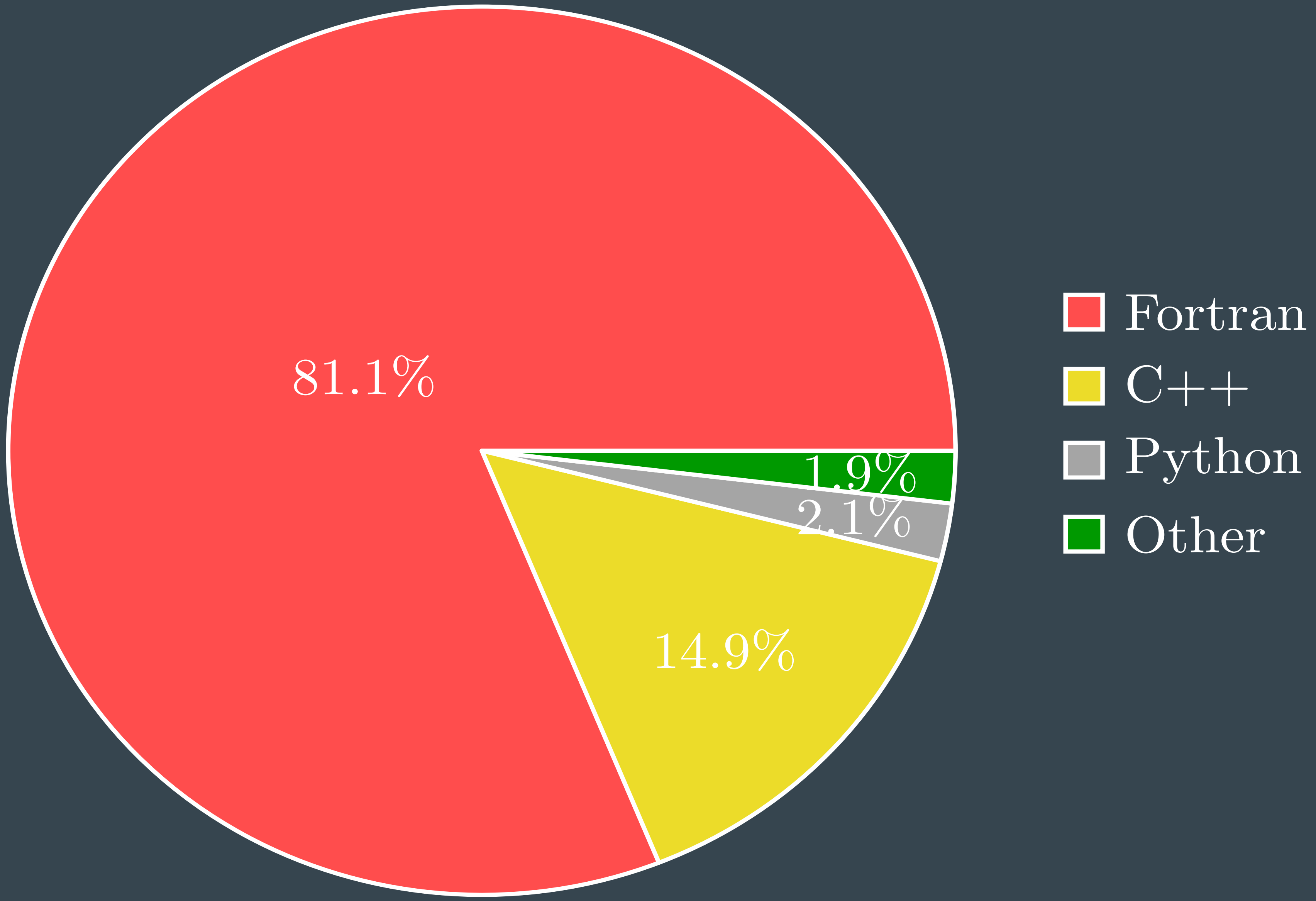
NVIDIA GPUs



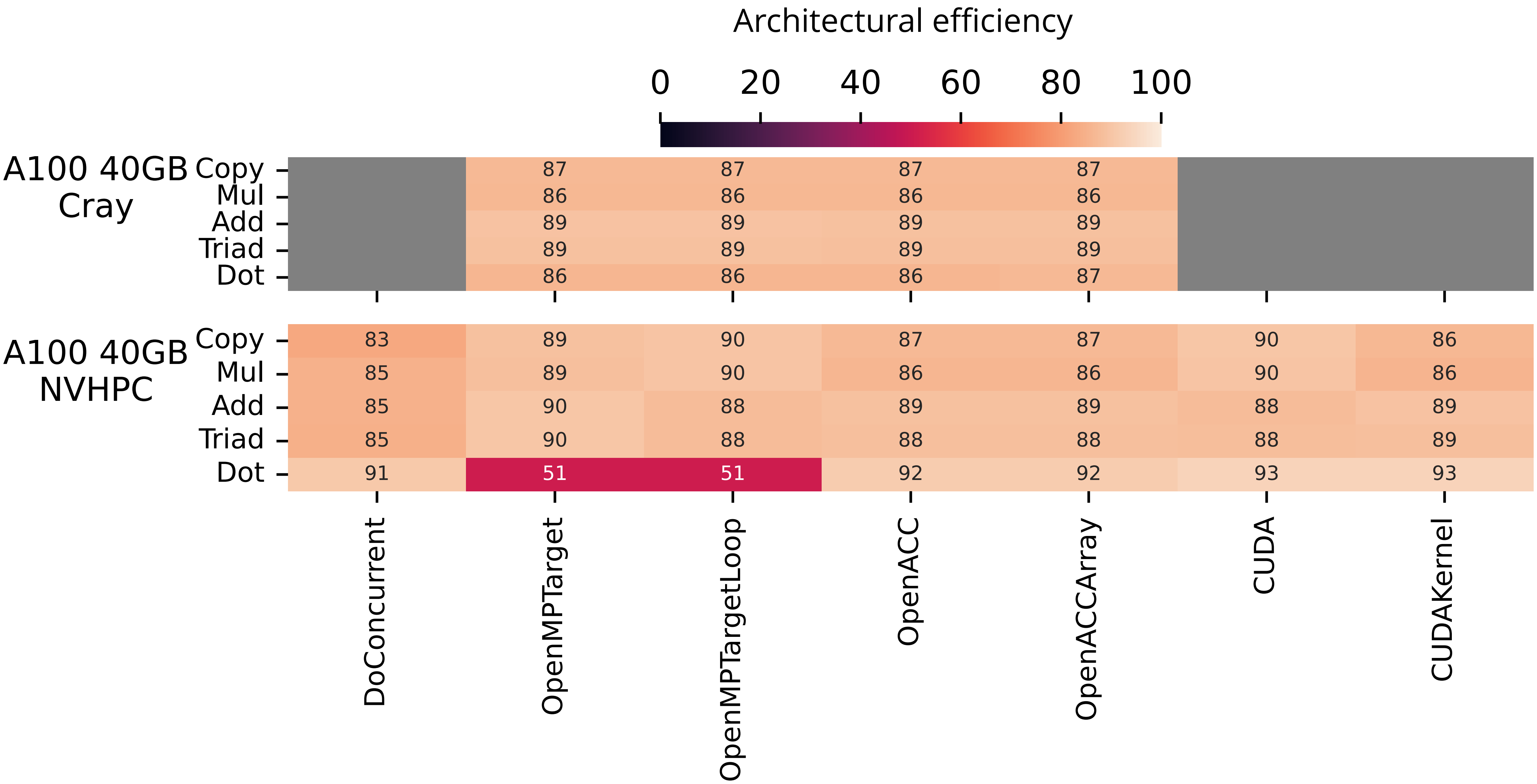
From doi.org/10.1109/PMBS56514.2022.00009

The University of Bristol is an Intel oneAPI Center of Excellence helped support this work.

ARCHER2: March-August 2022

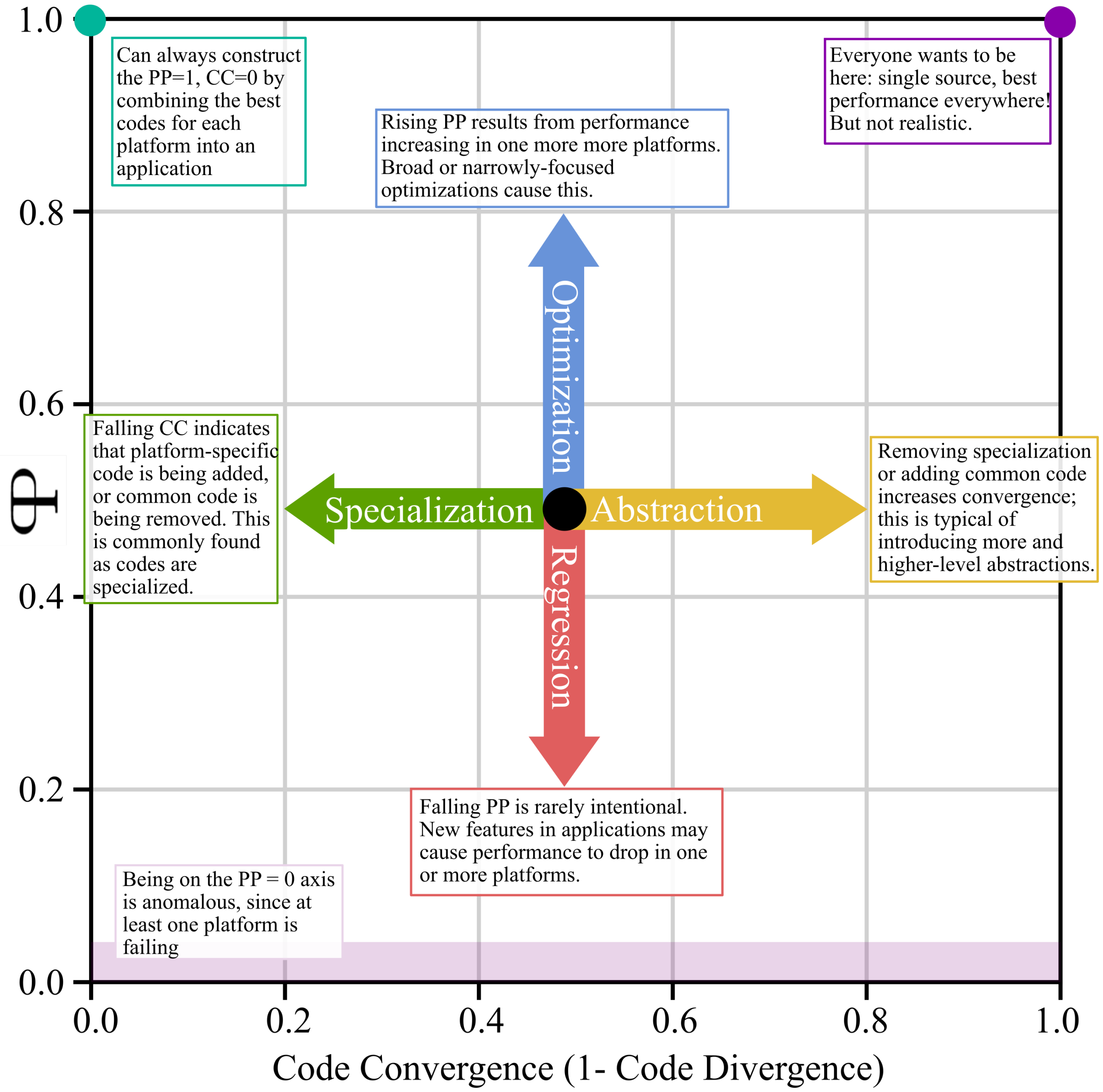


From doi.org/10.1109/PMBS56514.2022.00013



From doi.org/10.1109/PMBS56514.2022.00013

Specialisation?



Can always construct the PP=1, CC=0 by combining the best codes for each platform into an application

Rising PP results from performance increasing in one more more platforms. Broad or narrowly-focused optimizations cause this.

Everyone wants to be here: single source, best performance everywhere! But not realistic.

Falling CC indicates that platform-specific code is being added, or common code is being removed. This is commonly found as codes are specialized.

Removing specialization or adding common code increases convergence; this is typical of introducing more and higher-level abstractions.

Falling PP is rarely intentional. New features in applications may cause performance to drop in one or more platforms.

Being on the PP = 0 axis is anomalous, since at least one platform is failing

From doi.org/10.1109/P3HPC56579.2022.00006

Which performance portable programming model should I use?

http://uob-hpc.github.io/2020/05/05/choosing-models.html

Use open standard parallel programming models

Express all concurrent work asynchronously

Build in tuning parameters

Test all compilers & runtimes, on all systems

Tell your vendor

Programming Your GPU with OpenMP

Performance Portability for GPUs

By Tom Deakin and Timothy G. Mattson

November 7, 2023
Preorder via MIT Press website

https://hpc.tomdeakin.com

@tjdeakin

tom.deakin@bristol.ac.uk