Prof Simon McIntosh-Smith

HPC research group

University of Bristol

Performance Portability Across Diverse Computer Architectures

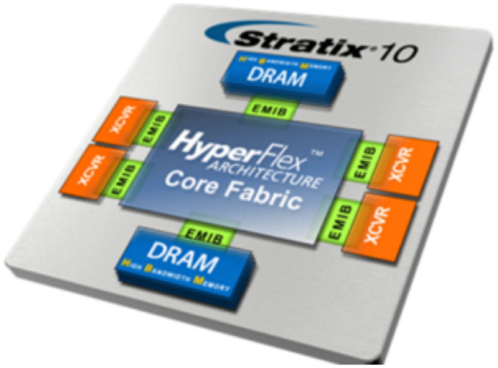
University of Bristol

http://uob-hpc.github.io

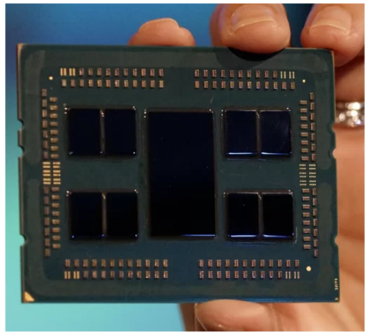
GW4

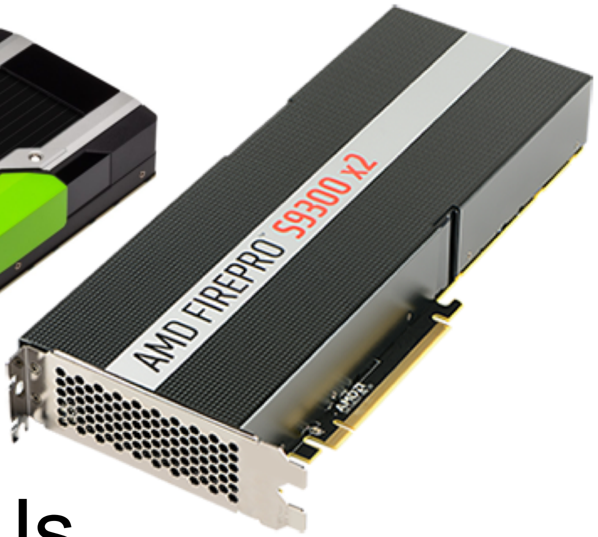
Recent processor trends in HPC

FPGAs



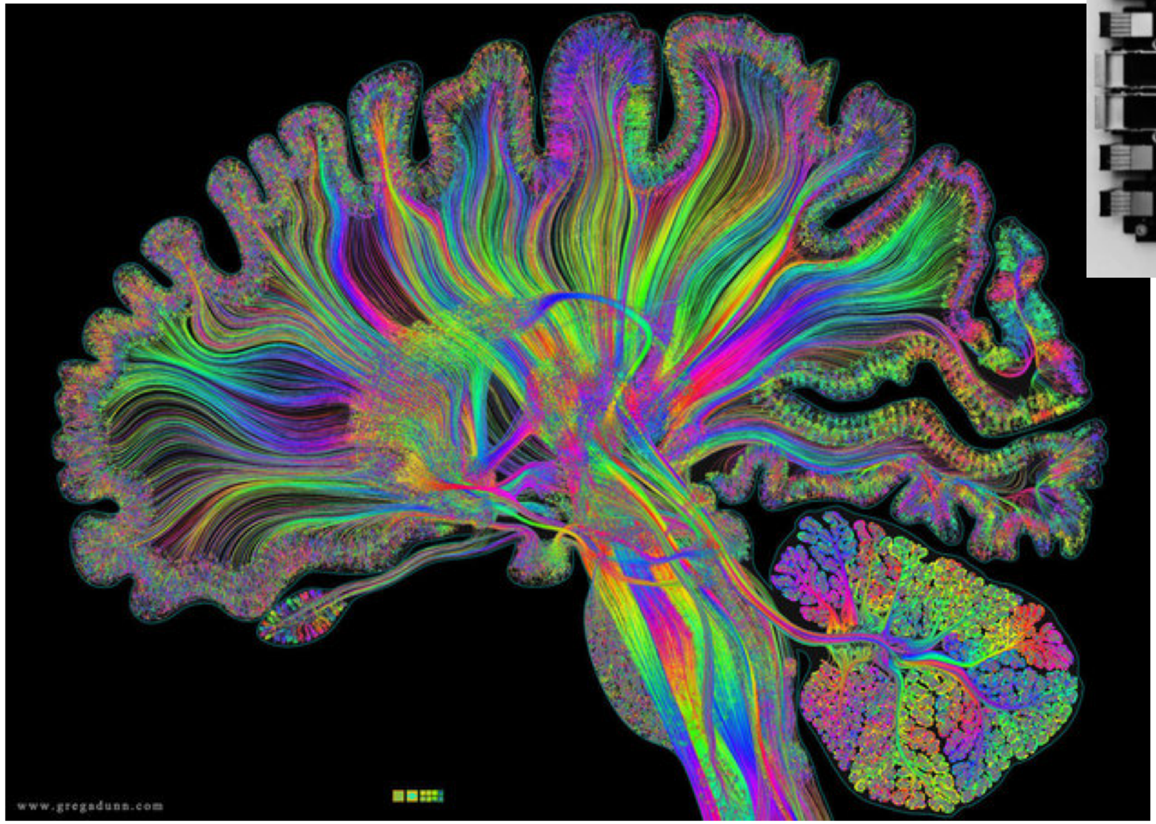
Many-core CPUs

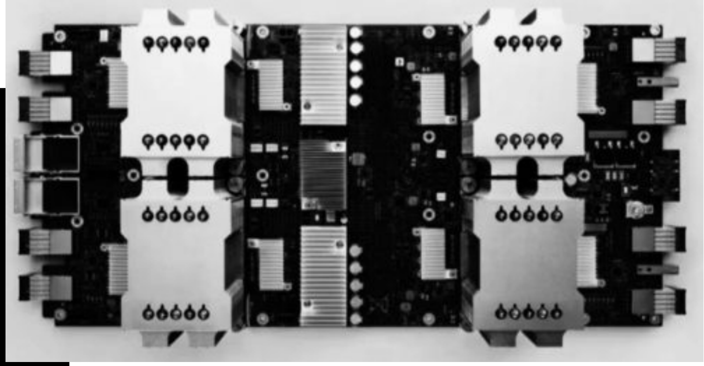


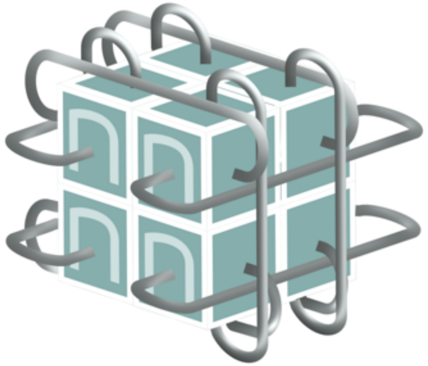


GPUs

Emerging architectures

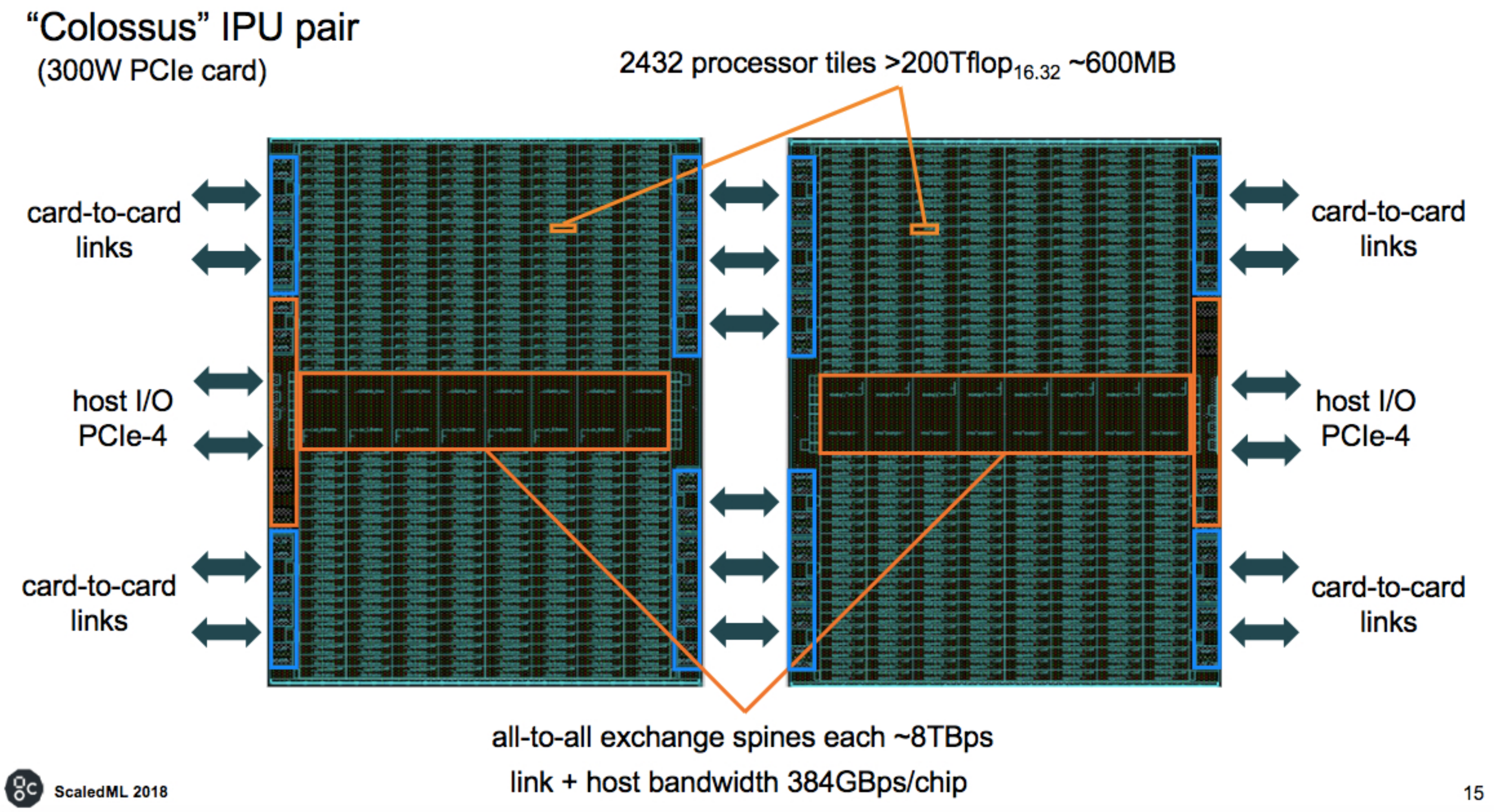






Google’s Tensorflow Processing Unit (TPU), GraphCore, Intel’s Nervana

GRAPHCORE IPU pair – 600MB @ 90TB/s



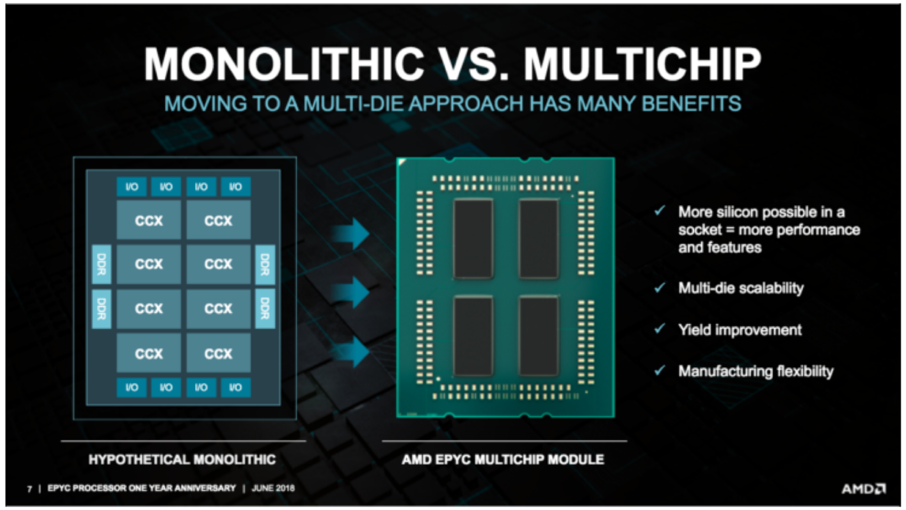
Recent CPU trends

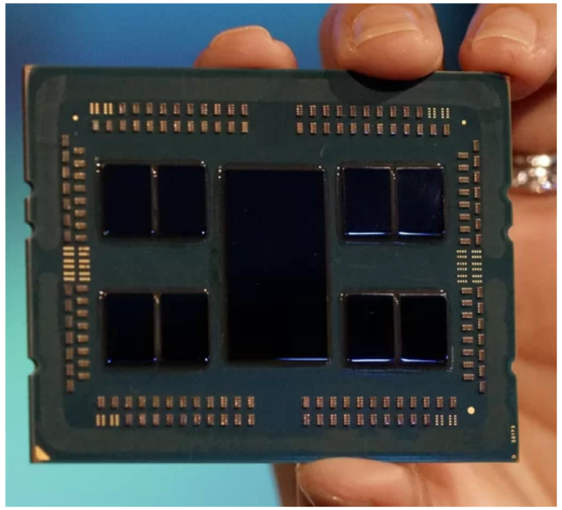
- CPUs have evolved to include **lots of cores** and **wide vector units**
- 32 core CPUs now common (AMD Naples, Marvell ThunderX2)
- 48, 64 core CPUs arrive within the next 12 months (A64fx, Rome)
- This **renewed competition in CPUs** is crucial to the health of the HPC ecosystem, and for performance per dollar
  - What about competition in GPUs? Intel and AMD…?

AMD's Rome showing where mainstream CPUs are heading

From late 2019:

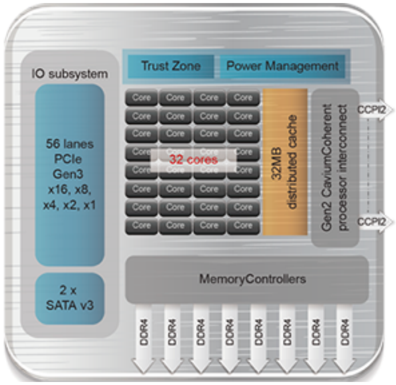
- Up to 64 heavyweight x86 cores per CPU
- Uses 8 chiplets of 8 cores each, plus an I/O chiplet

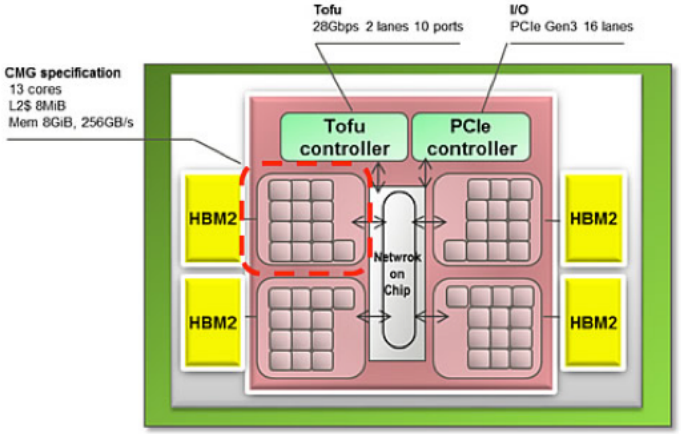


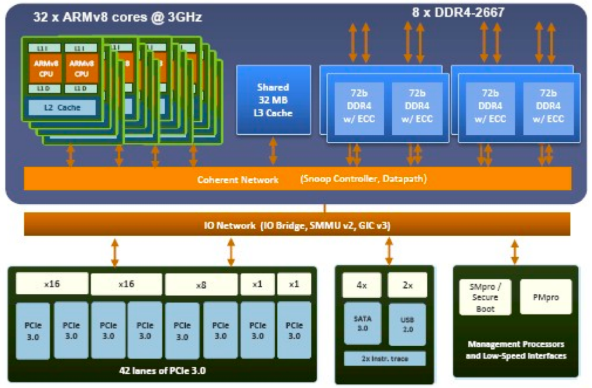


Chiplets likely to be an important future trend…

Emerging competition from Arm CPU vendors







University of BRISTOL

GW4

http://uob-hpc.github.io

A bit of history on Performance Portability in Bristol

What do I mean by “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.”

Questions:

- **Does it have to be a “good” fraction?** YES! Within 20% of “best achievable”, i.e. of hand-optimized OpenMP, CUDA, …
- **How wide is the range of target architectures?** Depends on your goal, but important to allow for future architectural developments

Original Article

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

High performance *in silico* virtual drug screening on many-core processors

The International Journal of High Performance Computing Applications
2015, Vol. 29(2) 119–134

DOI: 10.1177/1094342014528252
hpc.sagepub.com
SAGE

Simon McIntosh-Smith[1], James Price[1], Richard B Sessions[2] and Amaurys A Ibarra[2]

Abstract

Drug screening is an important part of the drug development pipeline for the pharmaceutical industry. Traditional, lab-based methods are increasingly being augmented with computational methods, ranging from simple molecular similarity searches through more complex pharmacophore matching to more computationally intensive approaches, such as molecular docking. The latter simulates the binding of drug molecules to their targets, typically protein molecules. In this work, we describe BUDE, the Bristol University Docking Engine, which has been ported to the OpenCL industry standard parallel programming language in order to exploit the performance of modern many-core processors. Our highly optimized OpenCL implementation of BUDE sustains 1.43 TFLOP/s on a single Nvidia GTX 680 GPU, or 46% of peak performance. BUDE also exploits OpenCL to deliver effective performance portability across a broad spectrum of different computer architectures from different vendors, including GPUs from Nvidia and AMD, Intel's Xeon Phi and multi-core CPUs with SIMD instruction sets.

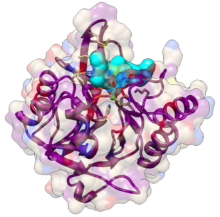
Unive
BRI

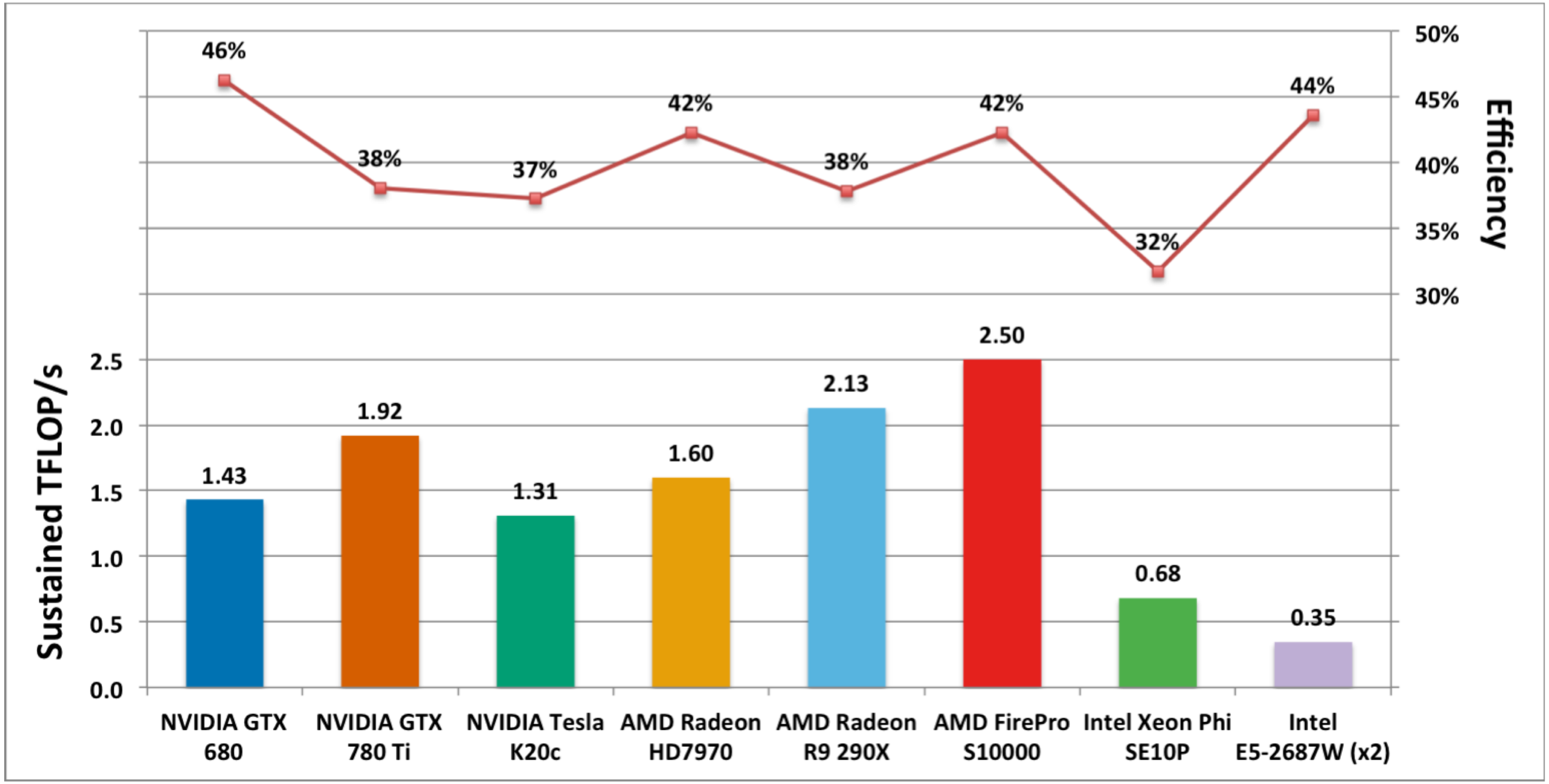
Keywords

Molecular docking, *in silico* virtual drug screening, many-core, GPU, OpenCL, performance portability

GW4

Bristol's first performance portable project: The BUDE molecular docking code





"High Performance *in silico* Virtual Drug Screening on Many-Core Processors", S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, IJHPCA 2014

University of BRISTOL

GW4

What about bandwidth bound codes?

- We developed "**BabelStream**" to measure the achievable fraction of peak memory bandwidth (formerly known as "GPU-STREAM")
- Cross platform
  - CPUs, GPUs, …
- Cross language
  - C/C++, OpenMP inc. target, CUDA, OpenACC, Kokkos, SYCL, …
- http://uob-hpc.github.io/BabelStream/

Deakin, T., Price, J., Martineau, M., & McIntosh-Smith, S. *Evaluating attainable memory bandwidth of parallel programming models via BabelStream*. International Journal of Computational Science and Engineering, April 2017.

Evaluating attainable memory bandwidth of parallel programming models via BabelStream

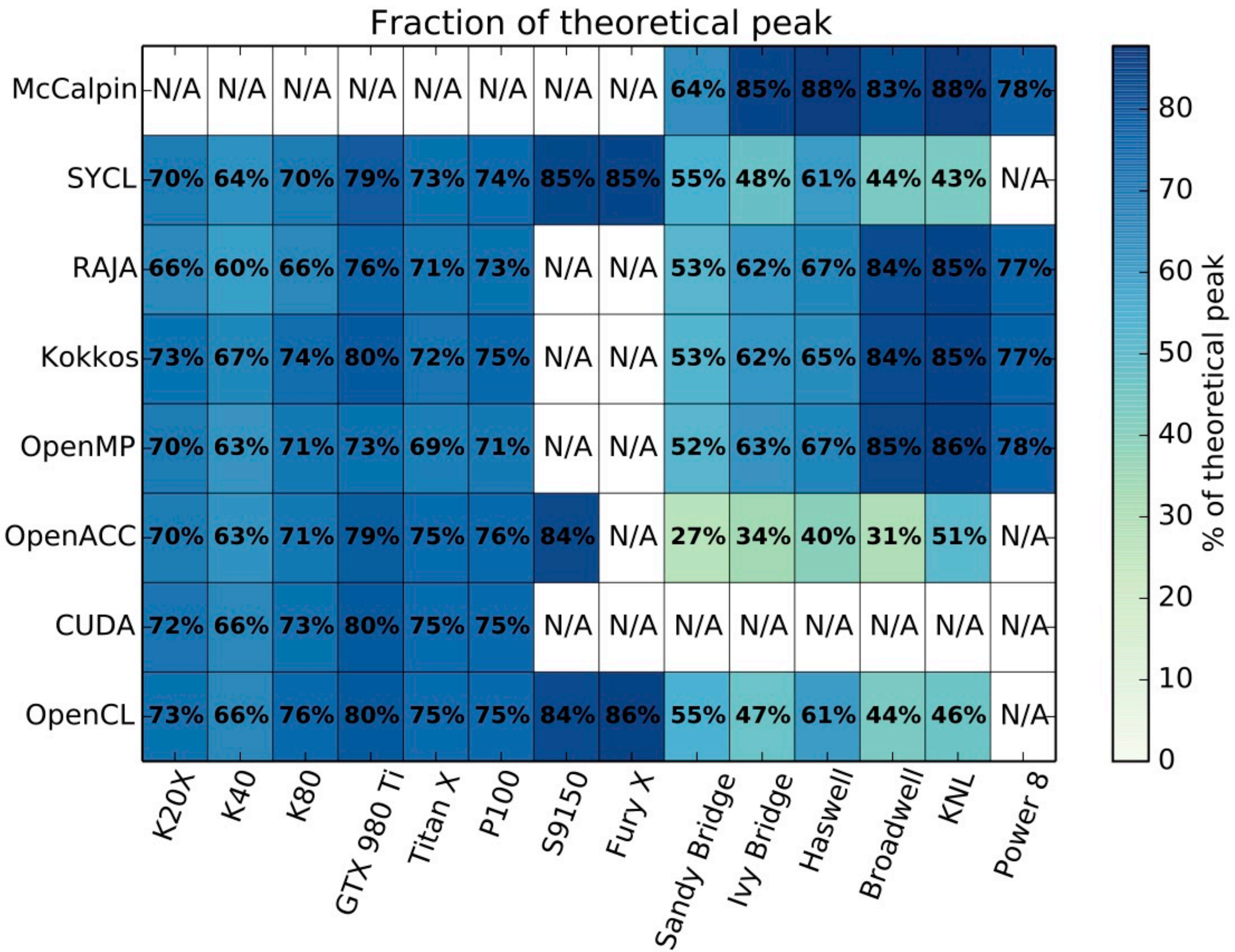
Tom Deakin*, James Price, Matt Martineau and Simon McIntosh-Smith

Department of Computer Science,
University of Bristol,
Bristol, UK
Email: tom.deakin@bristol.ac.uk
Email: J.Price@bristol.ac.uk
Email: m.martineau@bristol.ac.uk
Email: cssnmis@bristol.ac.uk
*Corresponding author

Abstract: Many scientific codes consist of memory bandwidth bound kernels. One major advantage of many-core devices such as general purpose graphics processing units (GPGPUs) and the Intel Xeon Phi is their focus on providing increased memory bandwidth over traditional CPU architectures. Peak memory bandwidth is usually unachievable in practice and so benchmarks are required to measure a practical upper bound on expected performance. We augment the standard STREAM kernels with a dot product kernel to investigate the performance of simple reduction operations on large arrays. The choice of programming model should ideally not limit the achievable performance on a device. BabelStream (formally GPU-STREAM) has been updated to incorporate a wide variety of the latest parallel programming models, all implementing the same parallel scheme. As such this tool can be used as a kind of Rosetta Stone which provides both a cross-platform and cross-programming model array of results of achievable memory bandwidth.

Keywords: performance portability; many-core; parallel programming models; memory bandwidth benchmark.

Unive
BRI



From: http://uob-hpc.github.io/BabelStream/results/

ISC 2014

On the Performance Portability of Structured Grid Codes on Many-Core Computer Architectures

Simon McIntosh-Smith, Michael Boulton, Dan Curran, and James Price

Department of Computer Science, University of Bristol,
Woodland Road, Clifton, Bristol, BS8 1UB, UK
http://www.cs.bris.ac.uk/home/simonm/

Abstract. With the advent of many-core computer architectures such as GPGPUs from NVIDIA and AMD, and more recently Intel's Xeon Phi, ensuring performance portability of HPC codes is potentially becoming more complex. In this work we have focused on one important application area — structured grid codes — and investigated techniques for ensuring performance portability across a diverse range of different, high-end many-core architectures. We chose three codes to investigate: a 3D lattice Boltzmann code (D3Q19 BGK), the CloverLeaf hydrodynamics mini application from Sandia's Mantevo benchmark suite, and ROTORSIM, a production-quality structured grid, multiblock, compressible finite-volume CFD code. We have developed OpenCL versions of these codes in order to provide cross-platform functional portability, and compared the performance of the OpenCL versions of these structured grid codes to optimized versions on each platform, including hybrid OpenMP/MPI/AVX versions on CPUs and Xeon Phi, and CUDA versions on NVIDIA GPUs. Our results show that, contrary to conventional wisdom, using OpenCL it is possible to achieve a high degree of performance portability, at least for structured grid applications, using a set of straightforward techniques. The performance portable code in OpenCL is also highly competitive with the best performance using the native parallel programming models on each platform.

After BabelStream, more realistic bandwidth bound codes

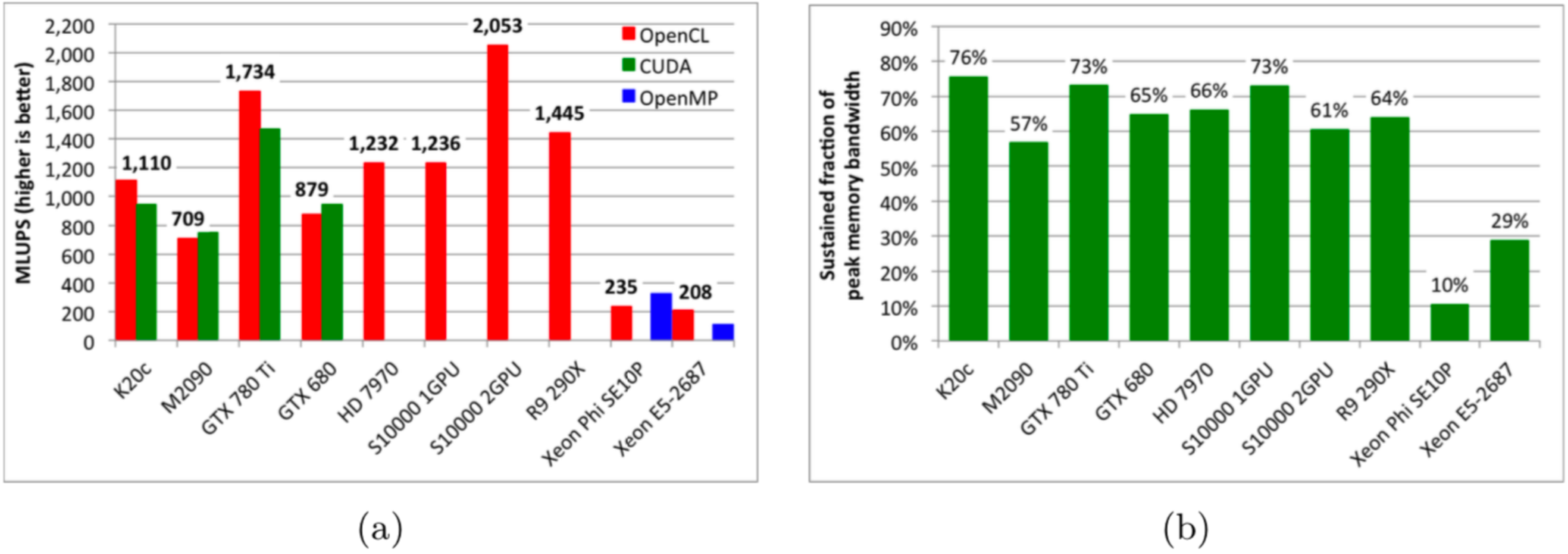


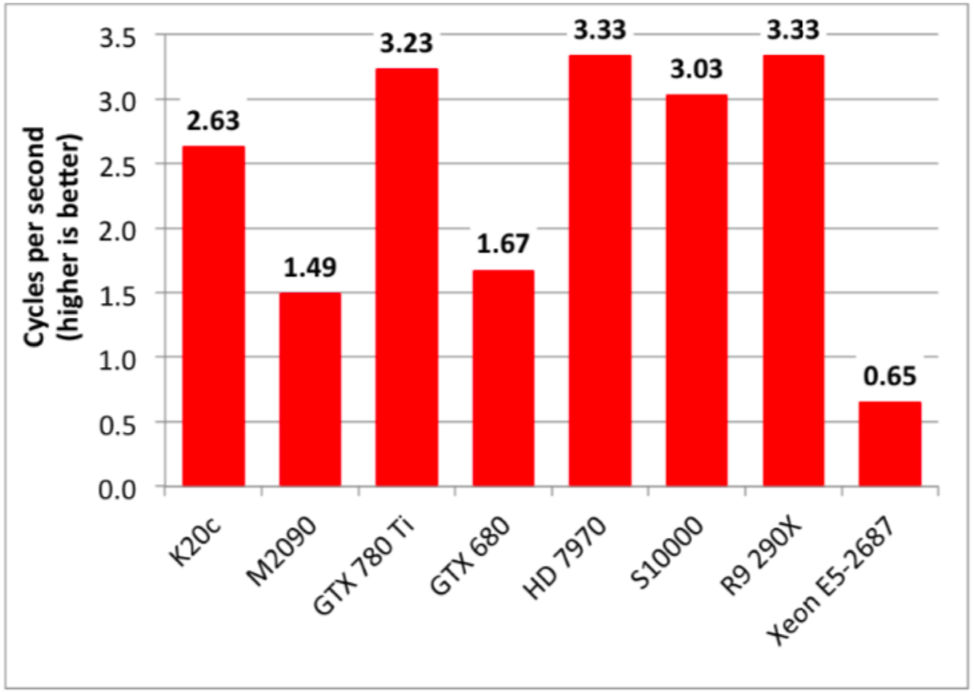
Fig. 1: D3Q19-BGK performance. Figure 1a shows MLUPS on the vertical axis, while Figure 1b shows the fraction of peak memory bandwidth sustained during the benchmark runs (higher is better in both graphs).

University of BRISTOL

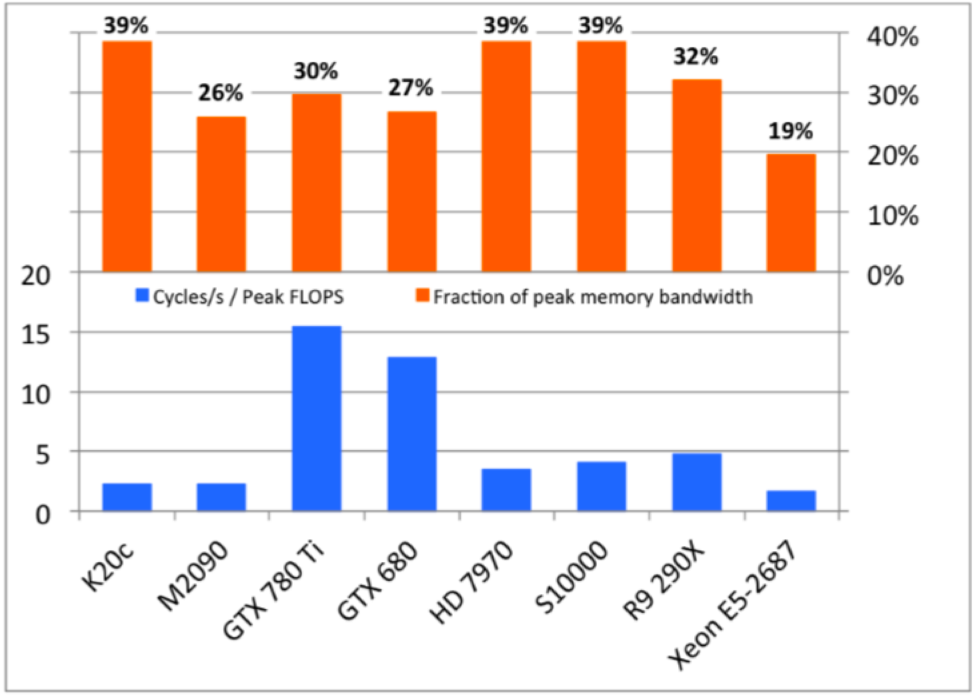
S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price, "On the performance portability of structured grid codes on many-core computer architectures", ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

GW4

After BabelStream, more realistic bandwidth bound codes



(a)



(b)

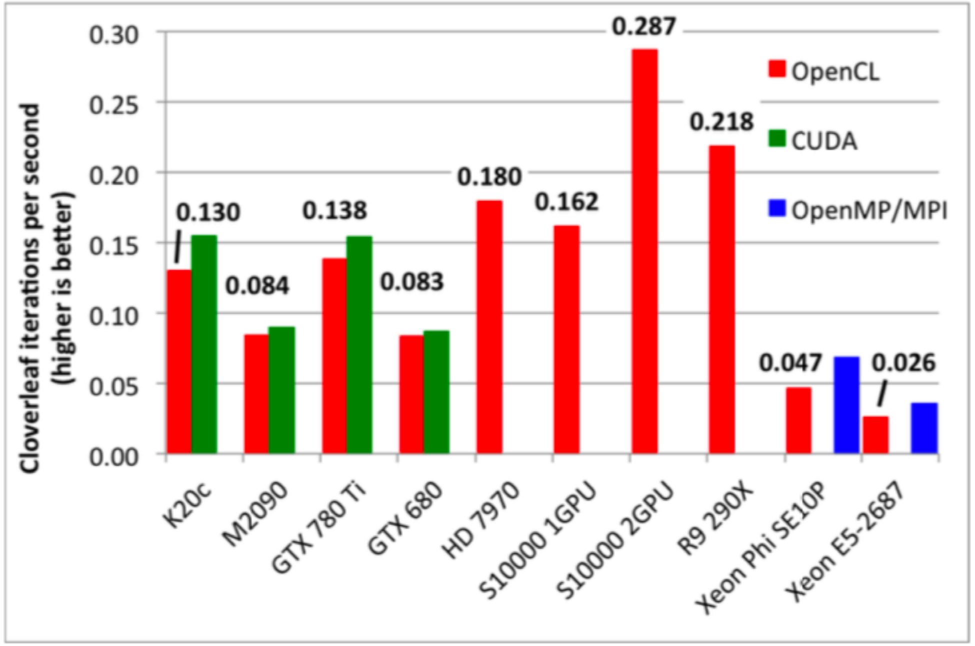
Fig. 2: ROTORSIM performance. Figure 2a shows performance in cycles per second. Figure 2b shows the sustained fraction of memory bandwidth on each device (top), and performance relative to each device's peak double precision floating point capability (bottom).

University of BRISTOL

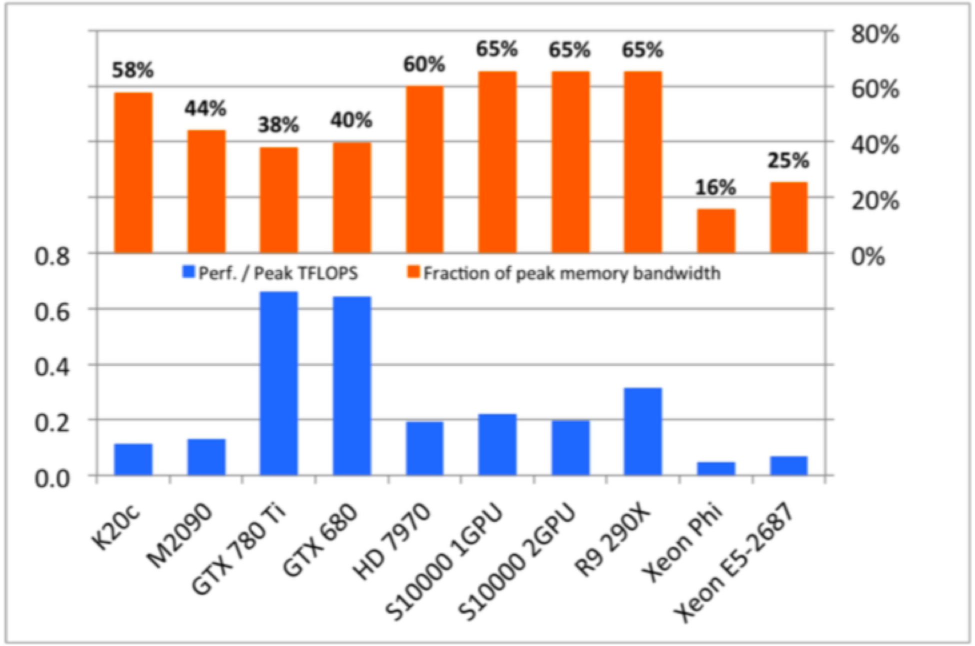
S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price, "On the performance portability of structured grid codes on many-core computer architectures", ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

GW4

After BabelStream, more realistic bandwidth bound codes



(a)



(b)

Fig. 3: CloverLeaf performance. Figure 3a shows performance in iterations per second. Figure 3b shows the sustained fraction of peak memory bandwidth (top), and performance relative to peak double precision floating point (bottom).

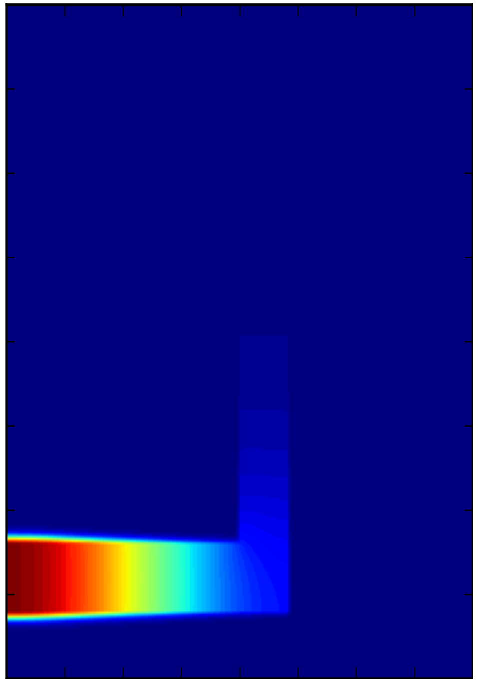
University of BRISTOL

S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price, "On the performance portability of structured grid codes on many-core computer architectures", ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4

GW4

More complex bandwidth bound codes

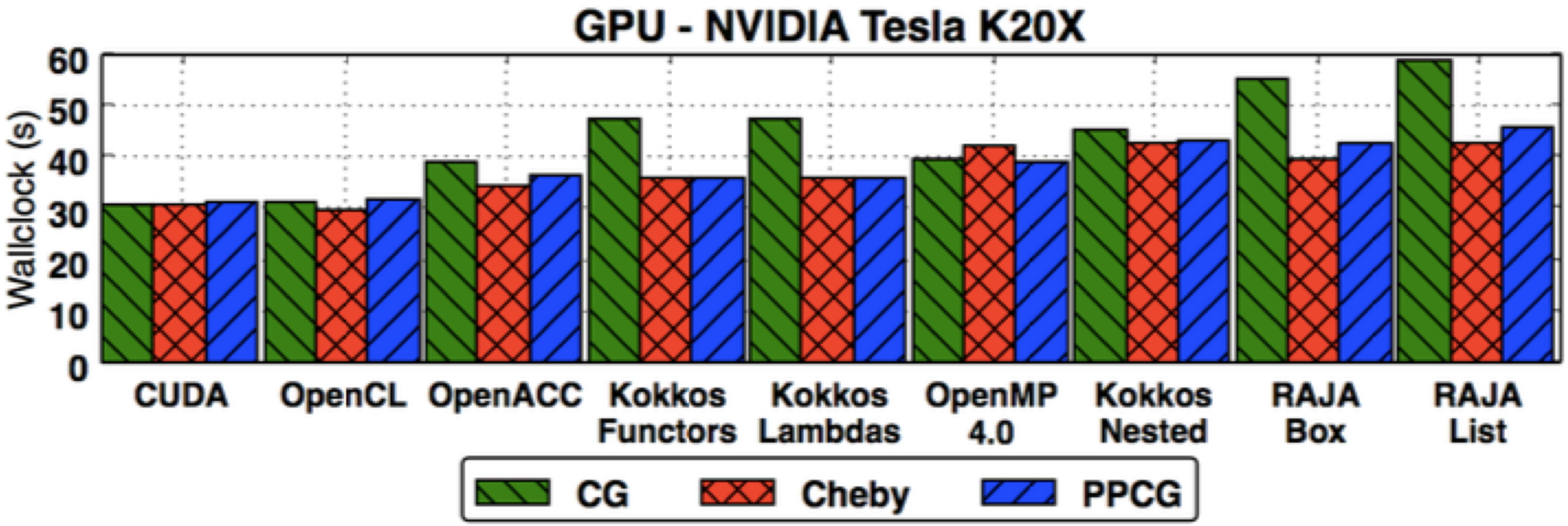
TeaLeaf heat conduction mini-app from the Mantevo suite of benchmarks



- Implicit, sparse, matrix-free solvers, structured grid
  - Conjugate Gradient (CG)
  - Chebyshev
  - Preconditioned Polynomial CG (PPCG)
- Memory bandwidth bound
- Good strong and weak scaling on Titan & Piz Daint

McIntosh-Smith, S., Martineau, M., et al. *TeaLeaf: a mini-application to enable design-space explorations for iterative sparse linear solvers*. WRAp workshop, IEEE Cluster 2017, Honolulu, USA.

TeaLeaf Performance Portability on GPUs



For TeaLeaf, all of the programming models got to within 25% of the performance of hand-optimised OpenCL / CUDA

Martineau, M., McIntosh-Smith, S. Gaudin, W., *Assessing the Performance Portability of Modern Parallel Programming Models using TeaLeaf,* 2016, CC-PE

University of BRISTOL

GW4

Performance Portability: the next phase

S. J. Pennycook, J. D. Sewall and V. W. Lee
Intel Corporation
Santa Clara, California
{john.pennycook,jason.sewall,victor.w.lee}@intel.com

***Abstract*—The term "performance portability" has been informally used in computing to refer to a variety of notions which generally include: 1) the ability to run one application across multiple hardware platforms; and 2) achieving some notional level of performance on these platforms. However, there has been a noticeable lack of consensus on the precise meaning of the term, and authors' conclusions regarding their success (or failure) to achieve performance portability have thus been subjective. Comparing one approach to performance portability with another has generally been marked with vague claims and verbose, qualitative explanation of the comparison. This paper presents a concise definition for performance portability, along with a simple metric that accurately captures the performance and portability of an application across different platforms. The utility of this metric is then demonstrated with a retroactive application to previous work.**

and demonstrate its accuracy and utility for quantifying an application's performance *and* portability; and

3) We retroactively apply our metric to a number of published application studies, thereby highlighting the utility of a shared metric when comparing and contrasting different approaches to performance portability.

II. Related Work

There have been a number of efforts to develop new programming models, languages and tools that provide users with a productive means of achieving performance portability. Some have proposed the use of domain-specific languages (DSLs), providing a limited set of high-level abstractions for a spe-

A more rigorous metric for Performance Portability

For a given set of platforms H, the performance portability P of an application a solving problem p is:

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

Where $e_i(a,p)$ is the performance efficiency of application a solving problem p on platform i.

Two ways to measure Performance Portability

Definitions from the Pennycook, Sewall and Lee paper:

1. **Architectural efficiency**:
   Achieved performance as a ***fraction of peak theoretical hardware performance***. This represents the ability of an application to utilize hardware efficiently;

2. **Application efficiency**:
   Achieved performance as a ***fraction of best observed performance***. This represents the ability of an application to use the most appropriate implementation and algorithm for each platform

A systematic evaluation of Performance Portability

- Studying Performance Portability is ***hard***!
  - Have to be **rigorous** about doing as well as possible across a wide range issues: architectures, programming languages, algorithms, compilers, …
- It takes a lot of effort to do this well
- Motivated by our results so far, in Bristol we have initiated a wide-ranging evaluation of Performance Portability:
  - Across many codes
  - Across many programming languages
  - Across many architectures
- Our goal is to share these codes and results to further the fundamental understanding of performance portability

Codes in the Bristol Performance Portability study

BabelStream:	simple measure of achievable memory bandwidth
CloverLeaf:	structured grid hydrodynamics
TeaLeaf:	structured grid heat diffusion
Neutral:	Monte Carlo neutral particle transport
MiniFMM:	fast multipole method
SNAP*:	structured grid deterministic neutral particle transport
unSNAP*:	unstructured grid deterministic neutral particle transport
Mini-HYDRA:	unstructured grid CFD (name TBC)
Mini-PRECISE:	combustion code

* = work in progress

University of BRISTOL

GW4

Parallel programming languages in the Bristol PP study

- OpenMP
- OpenMP target
- Kokkos CPU
- Kokkos GPU
- OpenACC
- CUDA
- OpenCL
- RAJA*
- SYCL*
- Flat MPI*

* = to come

Target hardware platforms

CPUs:

- Intel Skylake
- Intel KNL
- AMD Naples, Rome*
- IBM POWER9
- Marvell ThunderX2
- Marvell ThunderX3/4/5*
- Ampere eMAG
- Fujitsu A64fx*

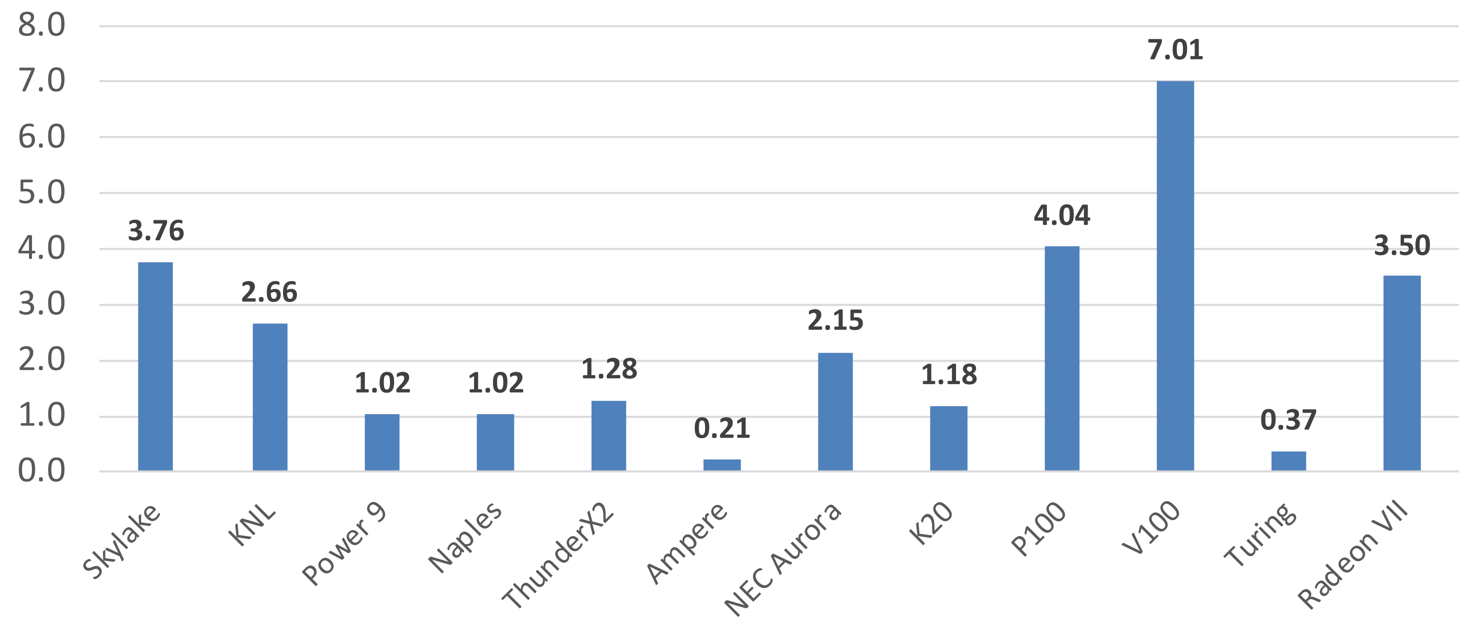
Accelerators:

- NEC Aurora
- NVIDIA Turing
- NVIDIA Volta
- NVIDIA Pascal
- AMD Radeon VII
- FPGAs*

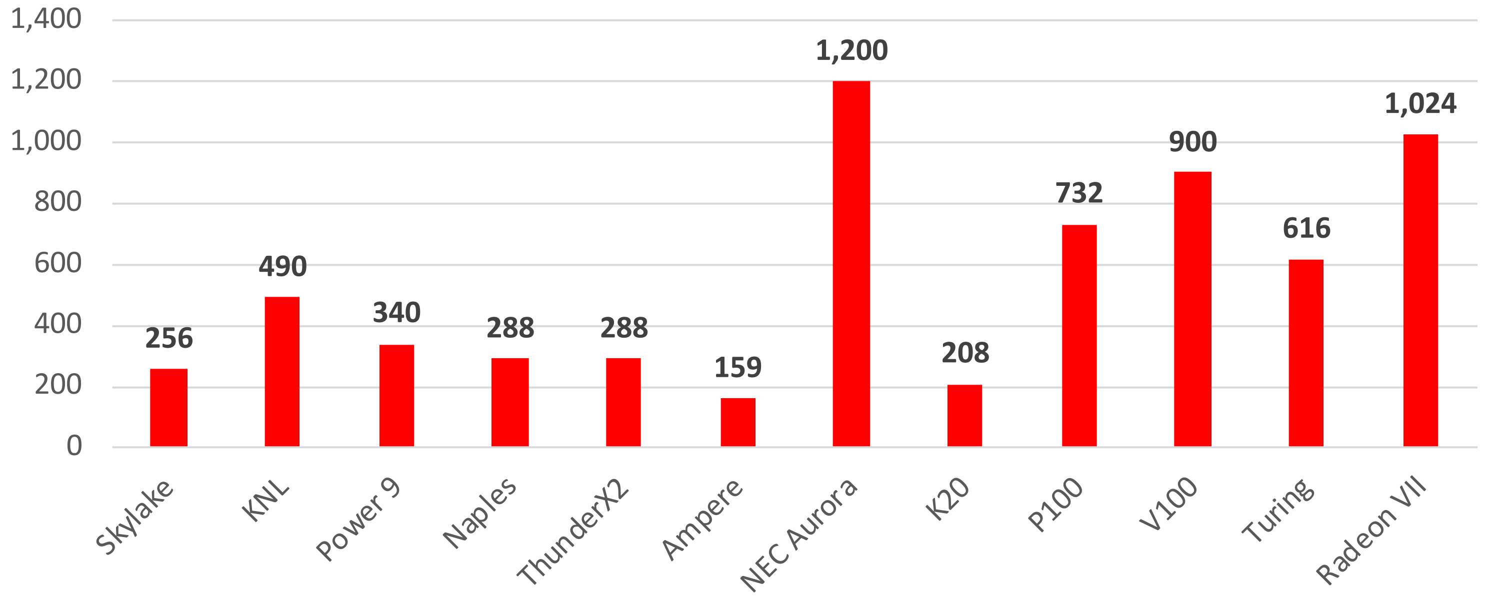
* = to come

Architecture	Sockets	Cores	Clocks Speed (GHz)	Peak DP FLOP/s	Peak SP FLOP/s	Peak BW (GB/s)
Skylake	2	28	2.1	3.76	7.53	256
KNL	1	64	1.3	2.66	5.32	490
Power 9	2	20	3.2	1.02	2.05	340
Naples	2	32	2.0	1.02	2.05	288
ThunderX2	2	32	2.5	1.28	2.56	288
Ampere	1	32	3.3	0.21	0.42	159
NEC Aurora	1	8	1.4	2.15	4.30	1,200
K20			0.71	1.18	3.52	208
P100			1.13	4.04	8.07	732
V100			1.37	7.01	14.03	900
Turing			1.35	0.37	11.75	616
Radeon VII			1.40	3.50	13.80	1,000

Peak D.P. FLOP/s



Peak BW GB/s

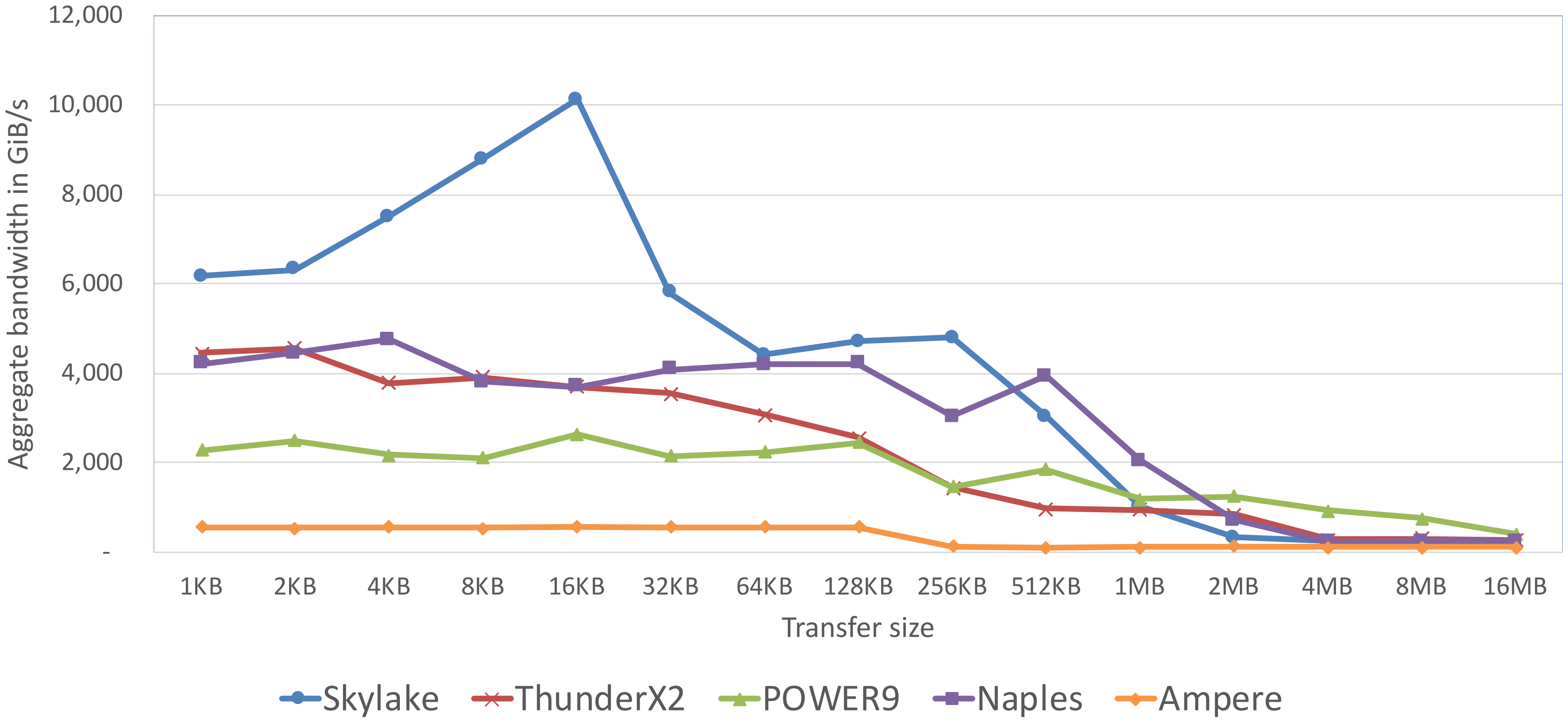


University of BRISTOL

http://uob-hpc.github.io

GW4

Quantifying performance: CPU memory bandwidth

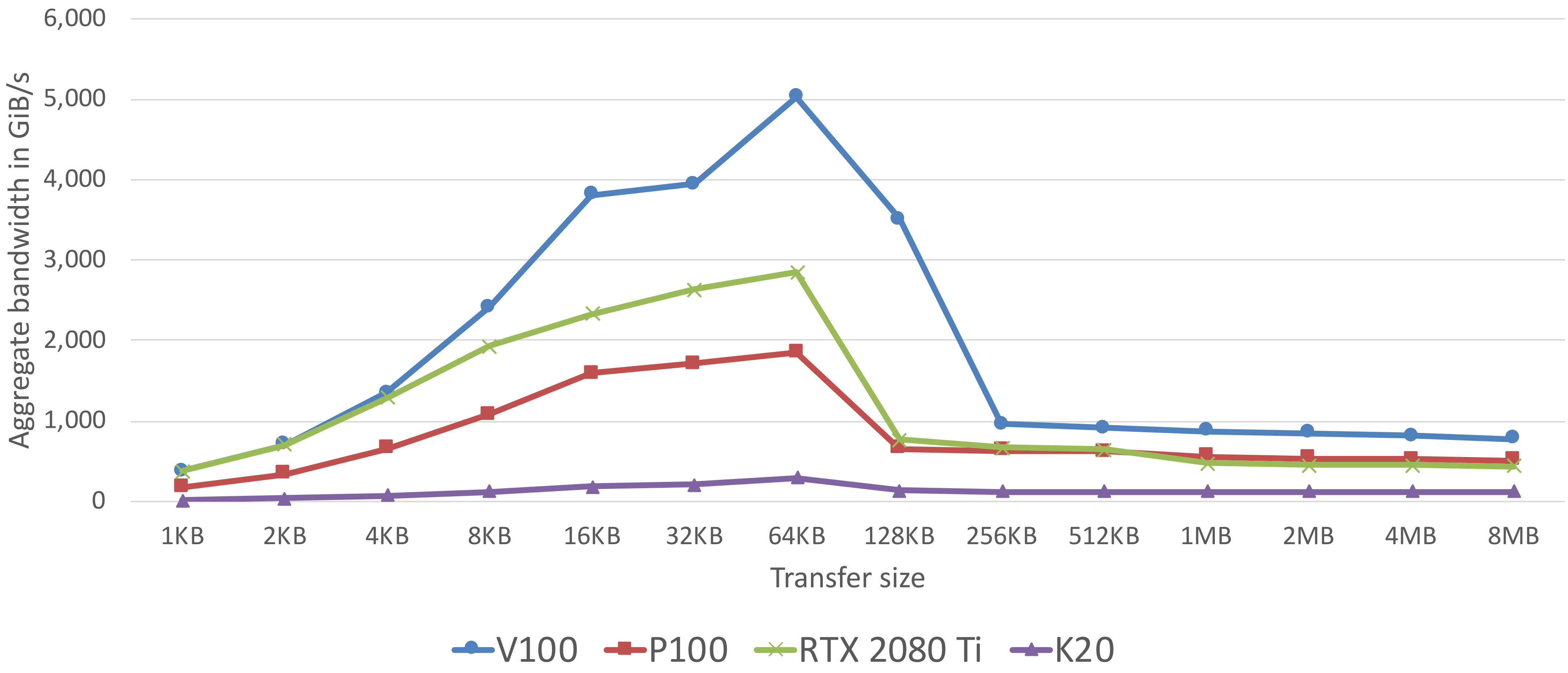


University of BRISTOL

http://uob-hpc.github.io

GW4

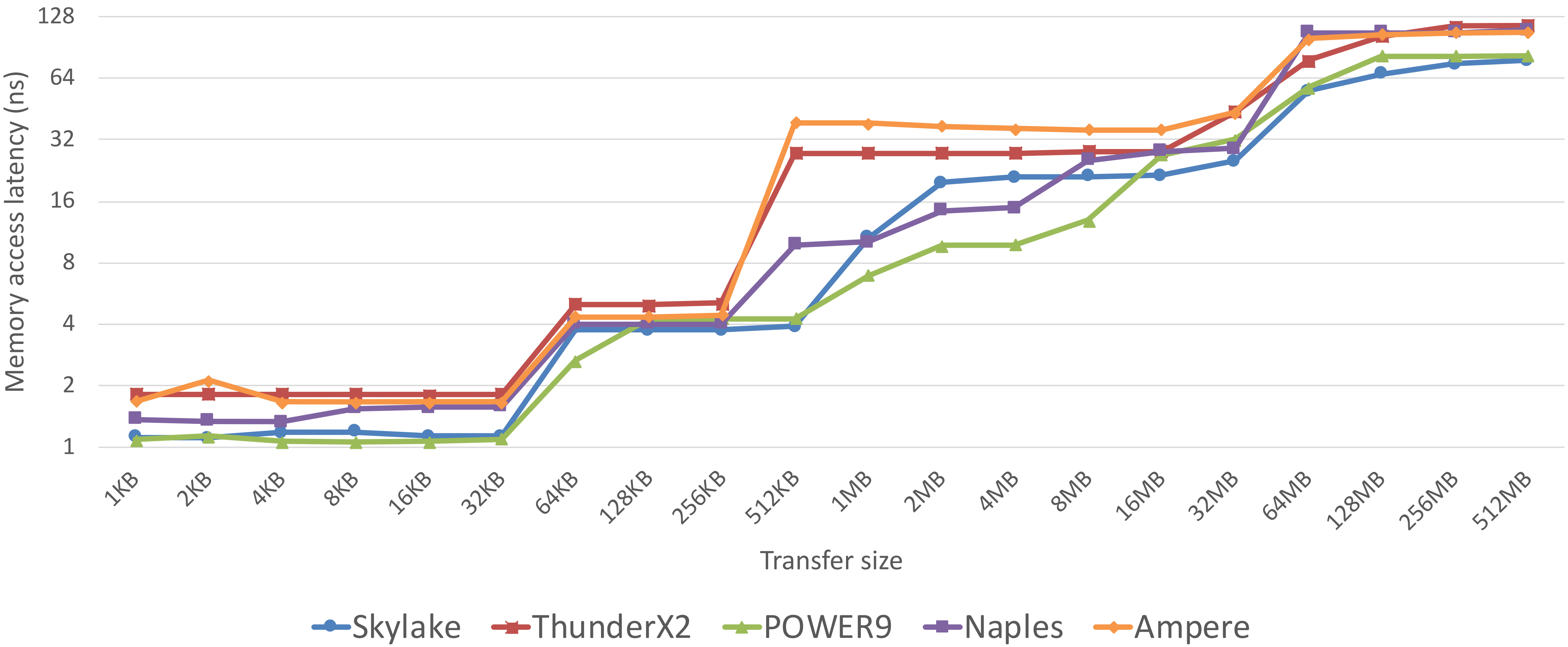
Quantifying performance: GPU memory bandwidth



University of BRISTOL

GW4

Quantifying performance: CPU memory latency



University of BRISTOL

http://uob-hpc.github.io

GW4

Bristol Performance Portability study

Latest results

BabelStream

Achieved bandwidth (GB/s)

Higher is better

	OpenMP	Kokkos	CUDA	OpenACC	OpenCL
Skylake	205	174	-	83	107
KNL	452	304	-	444	286
Power 9	248	250	-	247	-
Naples	240	191	-	257	-
ThunderX2	246	244	-	-	-
Ampere	106	91	-	-	-
NEC Aurora	976	-	-	-	-
K20	144	152	150	-	151
P100	553	557	552	552	551
V100	774	828	833	829	839
Turing	528	554	556	555	554
Radeon VII	-	-	-	-	814

Architectural efficiency (Fraction of hardware peak)

Higher is better

	OpenMP	Kokkos	CUDA	OpenACC	OpenCL
Skylake	80.2%	68.1%	-	32.4%	41.8%
KNL	92.2%	62.1%	-	90.7%	58.4%
Power 9	72.8%	73.6%	-	72.5%	-
Naples	83.4%	66.2%	-	89.3%	-
ThunderX2	85.3%	84.7%	-	-	-
Ampere	66.4%	57.3%	-	-	-
NEC Aurora	81.3%	-	-	-	-
K20	69.2%	72.9%	72.3%	-	72.8%
P100	75.5%	76.1%	75.4%	75.3%	75.3%
V100	86.0%	92.0%	92.6%	92.1%	93.2%
Turing	85.7%	90.0%	90.2%	90.1%	89.9%
Radeon VII	-	-	-	-	79.4%

Observations on BabelStream Performance Portability

- Today, ***no*** language runs successfully on all our platforms
- If we exclude the AMD Radeon GPU, then **OpenMP** successfully runs on all the remaining platforms, with PP = 79.1% (|H|, the number of platforms included in the metric, is 11)
- Excluding the NEC Aurora, then **Kokkos** can run across the remaining set with PP = 72.7% (|H|=10)
- If we further exclude all the Arm CPUs and the K20 GPU, then **OpenACC** runs on the remaining set of platforms, with PP = 68.6% (|H|=7)
- Excluding Power 9 and AMD Naples, **OpenCL** will run with PP = 68.3% (|H|=7)
- Finally, restricting the set of platforms to just NVIDIA GPUs, **CUDA** will run with PP = 81.7% (|H|=4)

TeaLeaf

Runtime in seconds

Lower is better

	OpenMP	Kokkos	CUDA	OpenACC
Skylake	317	370	-	-
KNL	191	885	-	-
Power 9	254	393	-	341
Naples	293	375	-	-
ThunderX2	314	439	-	-
Ampere	793	892	-	-
K20	1605	712	445	629
P100	190	187	122	153
V100	281	127	81	103
Turing	962	181	116	139

Observations on TeaLeaf Performance Portability

- Will use "**Application Efficiency**", efficiency compared to best observed runtime, for TeaLeaf and the remaining codes
- If we exclude the AMD Radeon GPU and the NEC Aurora, then **OpenMP** and **Kokkos** successfully run on all the remaining platforms, with PP = 43.6% and 57.4%, respectively (|H| = 10)
  - OpenMP results on GPU are much slower than with Kokkos, reflected in the scores
  - OpenMP GPU results from LLVM/trunk as not all platforms available with Cray compiler (which generally performs better than LLVM for OpenMP target code; see P100 result)
- When platforms = {Power 9, K20, P100, V100, Turing}, then **OpenACC** achieves P = 77.0% (|H| = 5)
  - OpenACC should work on Intel CPUs, but the code currently segfaults with PGI 18.10

CloverLeaf

Runtime in seconds

Lower is better

	OpenMP	Kokkos	CUDA	OpenACC	OpenCL
Skylake	376	-	-	877	-
KNL	250	-	-	698	-
Power 9	376	-	-	768	-
Naples	327	-	-	337	-
ThunderX2	457	-	-	-	-
Ampere	1309	-	-	-	-
NEC Aurora	323	-	-	-	-
K20	9737	1371	592	-	572
P100	226	182	139	133	149
V100	-	130	88.8	90.1	97.9
Turing	-	228	213	199	213
Radeon VII	-	-	-	-	106

University of BRISTOL

GW4

Observations on CloverLeaf Performance Portability

- A much more broken picture than TeaLeaf, with no approach working across the whole set of platforms
  - Harder to compare PP metric when there's little portability!
- **OpenMP** successfully runs on all the CPU platforms with PP = 100% (|H| = 7), but struggles on the GPUs except where we had the Cray compiler
- **OpenCL** runs on all the GPUs, including AMD Radeon VII, with PP = 94.5% (|H| = 5)
- **OpenACC** runs on all the NVIDIA GPUs except the K20 (fails to build), and all the CPUs except Arm, nor the NEC Aurora. PP = 62.4% (|H| = 7)
- **Kokkos** runs on all the GPUs except AMD Radeon VII, with PP = 62.8% (|H| = 4)

Neutral

Runtime in seconds

Lower is better

	OpenMP	Kokkos	CUDA	OpenACC	OpenCL
Skylake	8.0	13.0	-	-	-
KNL	23.8	28.1	-	-	-
Power 9	8.3	10.0	-	-	-
Naples	15.3	17.5	-	-	-
ThunderX2	12.6	13.5	-	-	-
Ampere	39.4	43.9	-	-	-
K20	-	52.7	41.6	88.4	29.7
P100	-	9.5	4.4	9.5	3.9
V100	-	5.6	2.8	3.7	3.3
Turing	-	9.3	6.9	8.7	6.7
Radeon VII	-	-	-	-	3.7

University of BRISTOL

GW4

Observations on Neutral Performance Portability

- **Kokkos** in the best condition here, running on all platforms except NEC Aurora and AMD Radeon VII, with P = 66.8% (|H| = 10)
  - For CPUs, **Kokkos** achieves PP = 81.7% (|H| = 6)
- **OpenMP** successfully runs on all the CPU platforms with PP = 100%, no target version yet for GPUs (|H| = 6)
- **OpenCL** runs on all the GPUs, including AMD Radeon VII, with PP = 96.8% (|H| = 5)
  - Will add Intel CPU results in the future
- **OpenACC** runs on all the NVIDIA GPUs with PP = 49.8% (|H| = 4).
  - Kokkos achieves PP = 52.5% for these GPUs
  - Will add OpenACC results for x86 and POWER CPUs in the future
- **CUDA** runs on all the NVIDIA GPUs with PP = 87.6%

MiniFMM

Runtime in seconds

Lower is better

	OpenMP	Kokkos	CUDA
Skylake	8.7	12.9	-
KNL	11.4	20.2	-
Power 9	23.6	38.5	-
Naples	15.4	19.6	-
ThunderX2	21.9	30.6	-
Ampere	116	127	-
K20	-	28.2	17.3
P100	-	4.7	3.5
V100	-	4.4	2.5
Turing	-	4.2	2.3

Observations on MiniFMM Performance Portability

- **Kokkos** again does well here, running on all platforms except NEC Aurora and AMD Radeon VII, with PP = 65.6% (|H| = 10)
  - MiniFMM uses identical code on CPUs and GPUs using shared memory
- **OpenMP** runs on all the CPU platforms with PP = 100% (|H| = 6)
  - On this same set of platforms, Kokkos achieves PP = 69.3%
  - No OpenMP target version yet for GPUs
- **CUDA** runs on all the NVIDIA GPUs with PP = 100% (|H| = 4)
  - Kokkos runs with PP = 60.6% here
- **Kokkos** does similarly well on CPU, GPU and combined groups
  - Higher PP score than TeaLeaf

PP measurements across the set of codes

- There are three platform groups of interest:
  - **CPU** = {Skylake, KNL, Power 9, Naples, TX2}
  - **GPU** = {K20, P100, V100, Turing}
  - **All** = {Skylake, KNL, Power9, Naples, ThunderX2, K20, P100, V100, Turing}

- This leaves out the three least mature / well covered platforms in our total set of 12:
  - **Deferred** = {Ampere, NEC aurora, AMD Radeon VII}

PP measurements across the the three platform groups

Higher is better

	BabelStream	TeaLeaf	CloverLeaf	Neutral	MiniFMM
OpenMP CPU	98.4%	100.0%	100.0%	100.0%	100.0%
Kokkos CPU	83.0%	49.8%	0.0%	80.2%	66.1%
OpenMP GPU	95.3%	23.6%	0.0%	0.0%	0.0%
Kokkos GPU	99.6%	63.8%	62.8%	52.5%	60.6%
OpenMP all	97.0%	41.0%	0.0%	0.0%	0.0%
Kokkos all	89.7%	55.2%	0.0%	65.0%	63.6%

Useful observations reading across the rows:

- On **CPUs**, **OpenMP** gets the best performance, with **Kokkos** 17-50% slower
- On **GPUs**, the support for a robust OpenMP offload across all platforms is lacking. Kokkos generally does better than OpenMP on GPUs
- **OpenMP all**: The lack of widespread support of OpenMP on GPUs means overall performance portability is lacking as of today
- **Kokkos all**: only CloverLeaf on CPUs a problem today. This shows performance portability is possible, with our Kokkos results generally being within 33% of the “best” for a given platform.

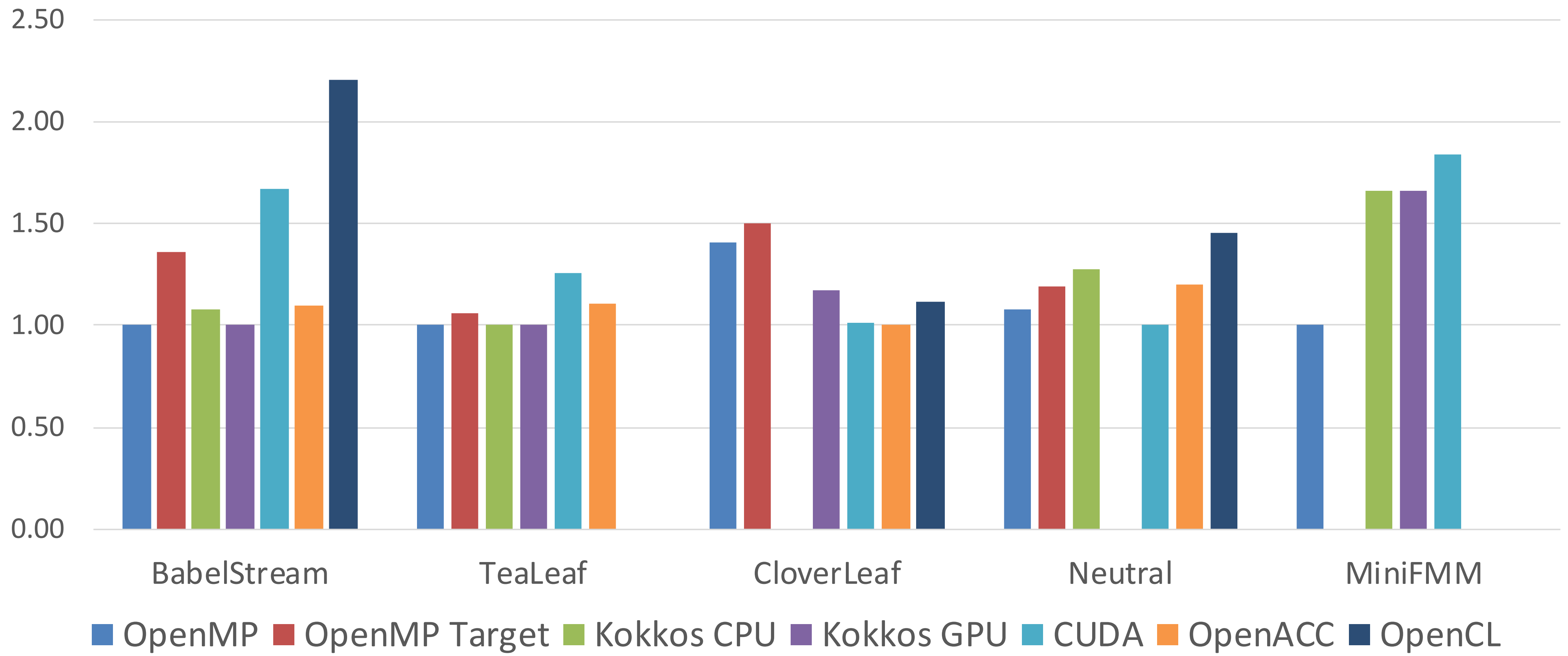
Overall Performance Portability observations thus far

- A very mixed bag
- A language may do well on one code, then poorly on the next
- Big differences between compilers for PP (esp. OpenMP target)
- OpenMP often achieving the best platform coverage
- Kokkos also achieving reasonable coverage
- OpenACC struggling for coverage on the CPUs (x86. A64fx? TX4?)
- OpenCL enjoying a resurgence with fast AMD GPUs re-emerging, Intel HPC GPUs on the horizon, and portability across some CPUs

Lessons learned about achieving performance portability

1. **Use open (standard) parallel programming languages** supported by multiple vendors across multiple hardware platforms
   - **E.g.** OpenMP, Kokkos, Raja, SYCL, …?
2. **Expose maximal parallelism** at all levels of the algorithm and application
3. **Avoid over-optimising** for any one platform
   - Optimise for at least two different platforms at once
4. **Multi-objective autotuning** can significantly improve performance
   - **Autotune for more than one target at once**
   - See: **Exploiting auto-tuning to analyze and improve performance portability on many-core architectures**, J.Price and S. McIntosh-Smith, P^3MA, ISC'17

Lines of code (normalized to lowest)



Recommendations and call to arms – I

- The current state of PP is not good enough and radical intervention is required
- Set up a long-term Performance Portability improvement program
  - **3 M's: Mandate it, Measure it, Maintain it**
- Need to select a broad-enough set of target platforms and codes, and **mandate a PP score of at least 80%** for this set
- Driven by users, with buy-in from PP solutions providers and platform vendors
- Must be led by an ***independent*** party

Recommendations and call to arms – II

- **Performance Portability must be elevated to a mandatory requirement** for future procurements, Exascale programs etc.
  - Add requirements that are ***objective*** and ***measurable***, just like benchmark results
  - E.g. a set of codes (real and mini-apps) must hit the PP application efficiency metric of at least 80% across the platform set consisting of Volta GPUs from Summit/Sierra and Xe GPUs in Aurora. Sensible to include Rome, A64fx, ThunderX4. Chose a set of codes from ECP.
- Bristol's contribution is to open source our "BabelSuite" of codes in as many languages and on as many platforms as we can, complete with build and run scripts

The Bristol HPC team doing this work

Tom Deakin

Patrick Atkinson

Andrei Poenaru

James Price

Also: Matt Martineau (now at NVIDIA), Codrin Popa and Justin Salmon

For more information

Bristol HPC group: https://uob-hpc.github.io/

Build & run scripts: https://github.com/UoB-HPC/benchmarks

Isambard: http://gw4.ac.uk/isambard/

Twitter: **@simonmcs**

- **High Performance *in silico* Virtual Drug Screening on Many-Core Processors**
  S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, IJHPCA 2014
- **On the performance portability of structured grid codes on many-core computer architectures**
  S.N. McIntosh-Smith, M. Boulton, D. Curran, & J.R. Price
  ISC, Leipzig, June 2014. DOI: 10.1007/978-3-319-07518-1_4
- **Assessing the Performance Portability of Modern Parallel Programming Models using TeaLeaf**
  Martineau, M., McIntosh-Smith, S. & Gaudin, W.
  Concurrency and Computation: Practice and Experience (Apr 2016)
- **GPU-STREAM v2.0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models**
  Deakin, T. J., Price, J., Martineau, M. J. & McIntosh-Smith, S. N.
  First International Workshop on Performance Portable Programming Models for Accelerators (P3MA), ISC 2016
- **The Productivity, Portability and Performance of OpenMP 4.5 for Scientific Applications Targeting Intel CPUs, IBM CPUs, and NVIDIA GPUs**
  M. Martineau and S. McIntosh-Smith, IWOMP 2017, Stony Brook, USA.

- **Evaluating Attainable Memory Bandwidth of Parallel Programming Models via BabelStream**
  Deakin, T, Price, J, Martineau, M, and McIntosh-Smith, S
  International Journal of Computational Science and Engineering (special issue), vol 17., 2018
- **Pragmatic Performance Portability with OpenMP 4.x**
  Martineau, Matt, Price, James, McIntosh-Smith, Simon, and Gaudin, Wayne
  Proceedings of the 12th International Workshop on OpenMP, 2016
- **Performance Analysis and Optimization of Clang's OpenMP 4.5 GPU Support**
  Martineau, Matt, McIntosh-Smith, Simon, Bertolli, Carlo, et al
  Proceedings of the International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2016, SC'16
- **Exploiting auto-tuning to analyze and improve performance portability on many-core architectures**
  Price, J. & McIntosh-Smith, S., P^3MA, ISC High Performance 2017 International Workshops, Revised Selected Papers. Springer, Cham, p.538-556, vol. 10524 LNCS