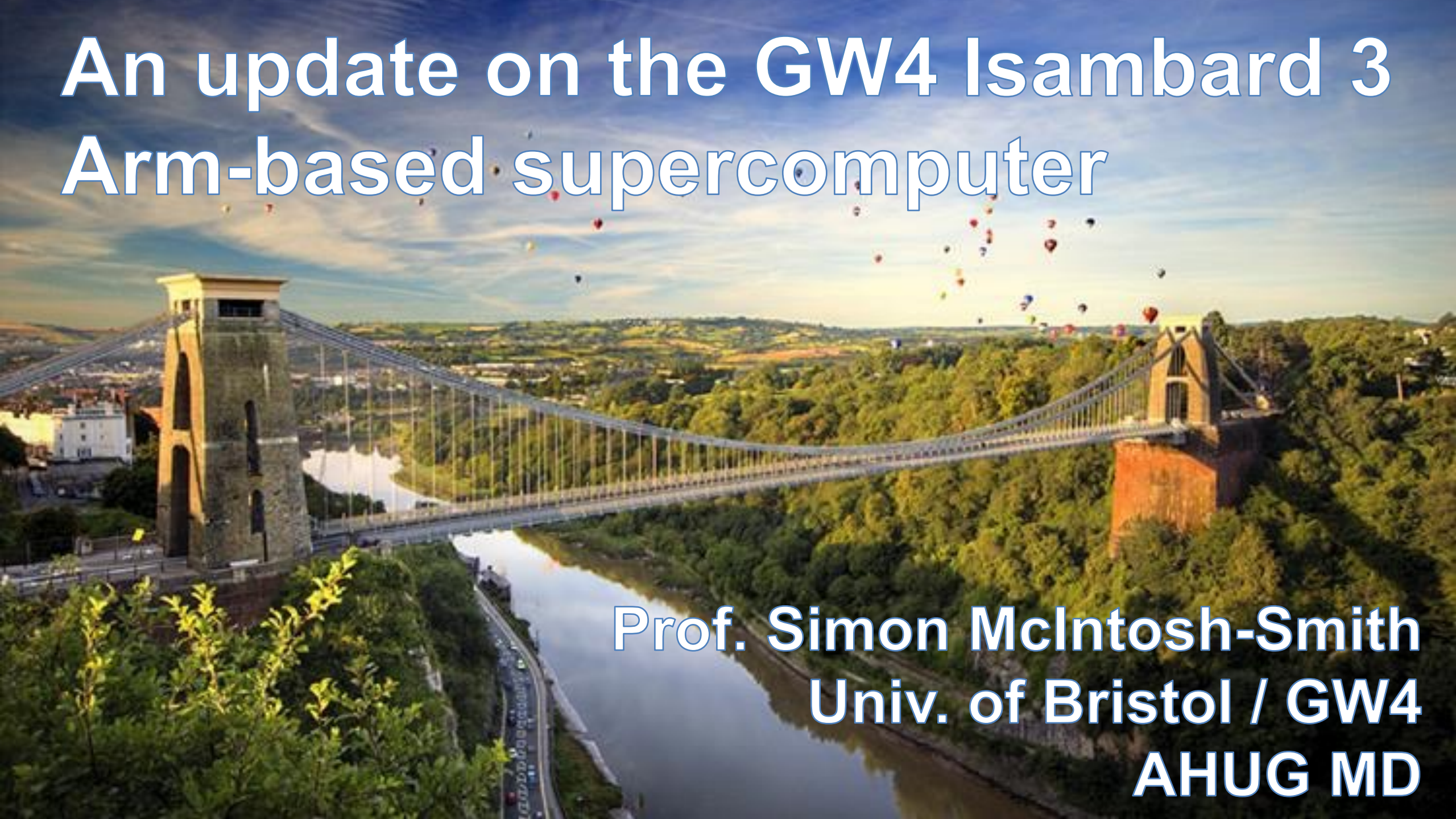
An update on the GW4 Isambard 3 Arm-based supercomputer



Prof. Simon McIntosh-Smith
Univ. of Bristol / GW4

AHUG MD

GW4 Isambard HPC service

- **Isambard 1 was the 1st production Arm-based HPC service in the world**
  - **Went live Spring 2018**
- **Isambard 2** expanded in 2020 to offer one of the **largest** Arm systems in the world at that time
- **>800 registered users, >400 GW4**
- **£7.7M EPSRC funding to date**
- Hosted by the Met Office in Exeter, UK
- **Multiple awards, best papers,...**

University of BRISTOL

GW4

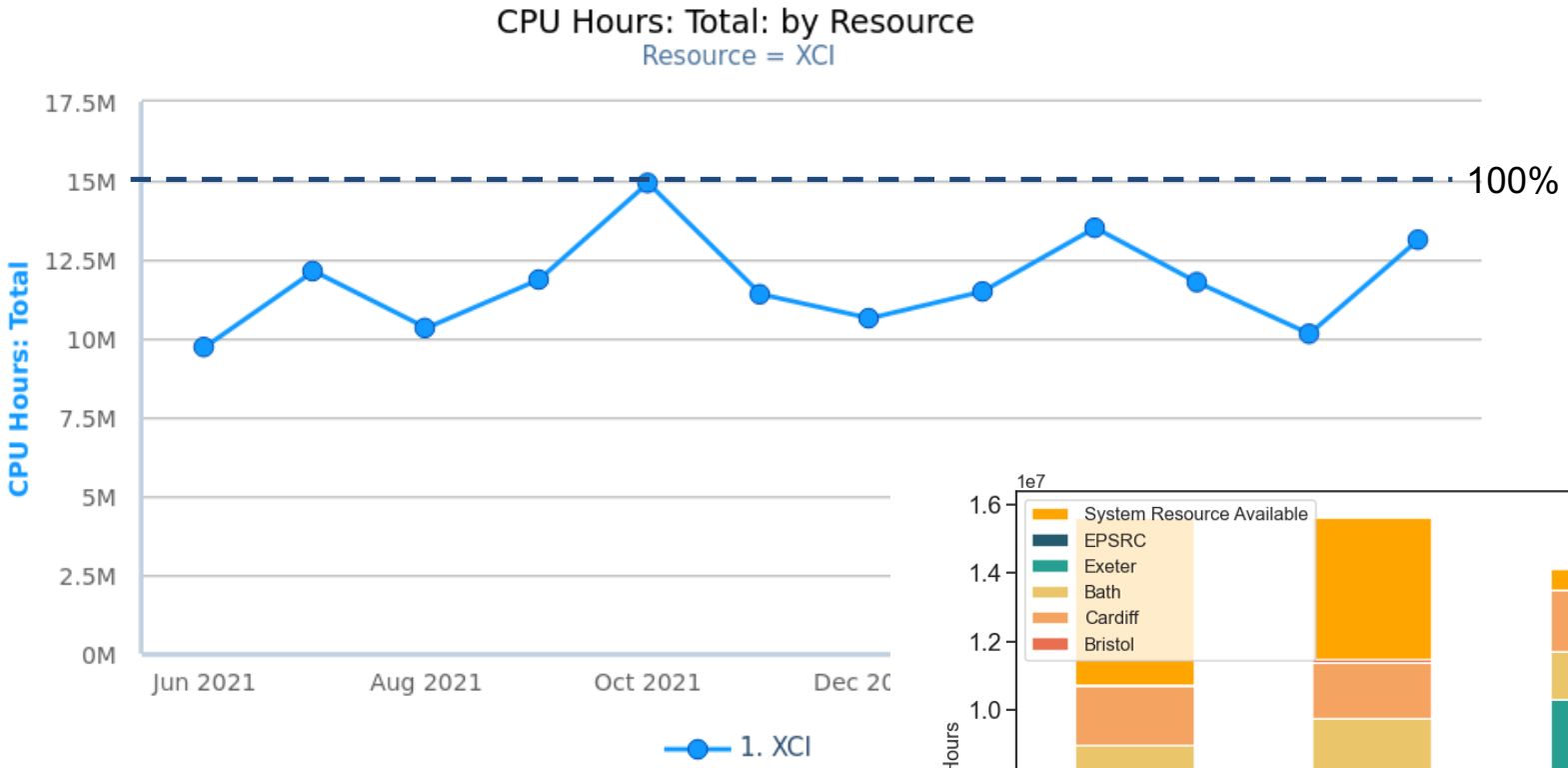
Some of Isambard's achievements to date

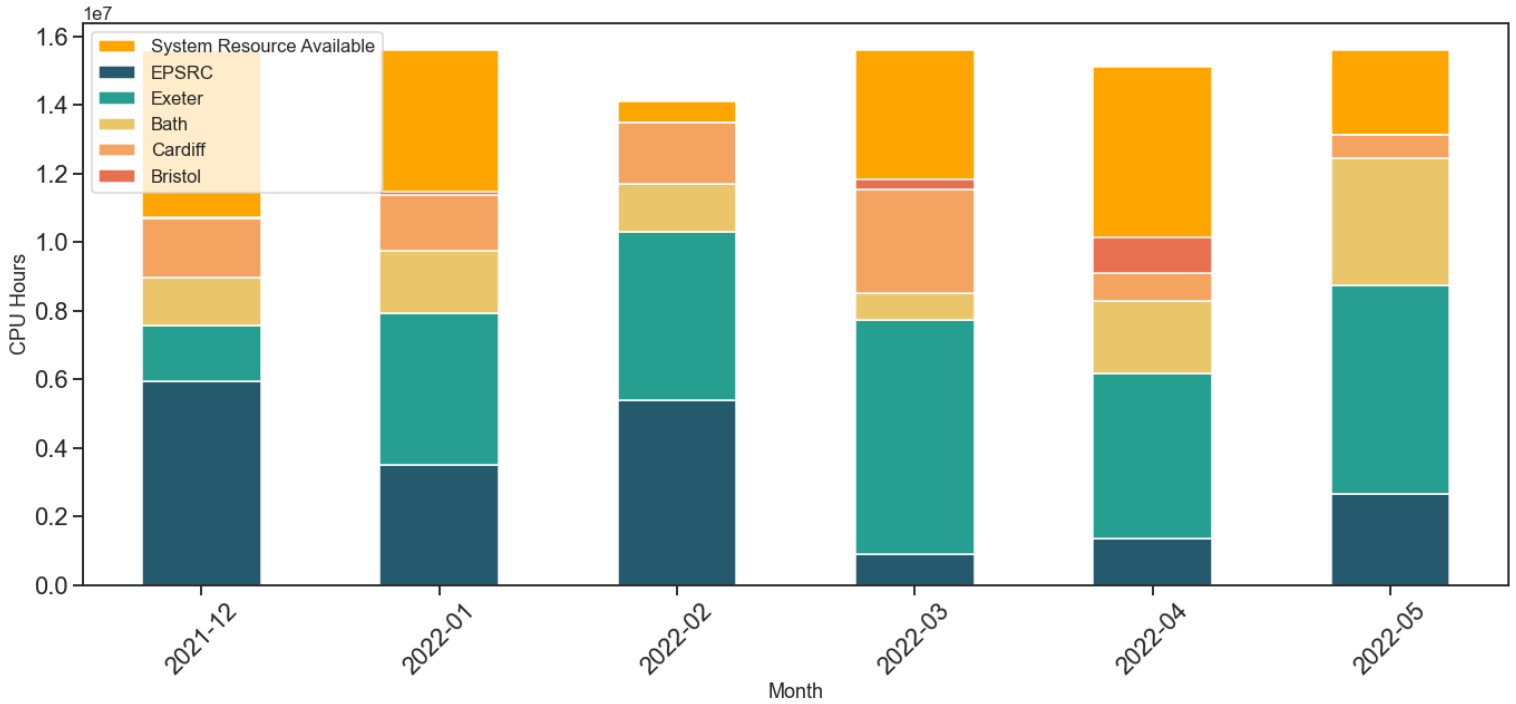
- Delivered **nearly 1B Arm core hours to date**, 20M per month
- Hundreds of scientists and engineers **trained on Arm in HPC**
- Dozens of **hands-on tutorials and hackathons** (SC, ISC, AHUG…)
- **Dozens of HPC codes ported to Arm** for the first time on Isambard
- **Best paper award** at CUG 2019
- World's first hands-on Arm tutorial on production system (SC18)
- **World's first open SVE tutorial on real hardware** (SC20)
- Made significant contributions to the quality and robustness of the main **Arm software toolchains**: LLVM, GNU, Cray, Fujitsu

University of BRISTOL

GW4

Graphs from recent annual report.



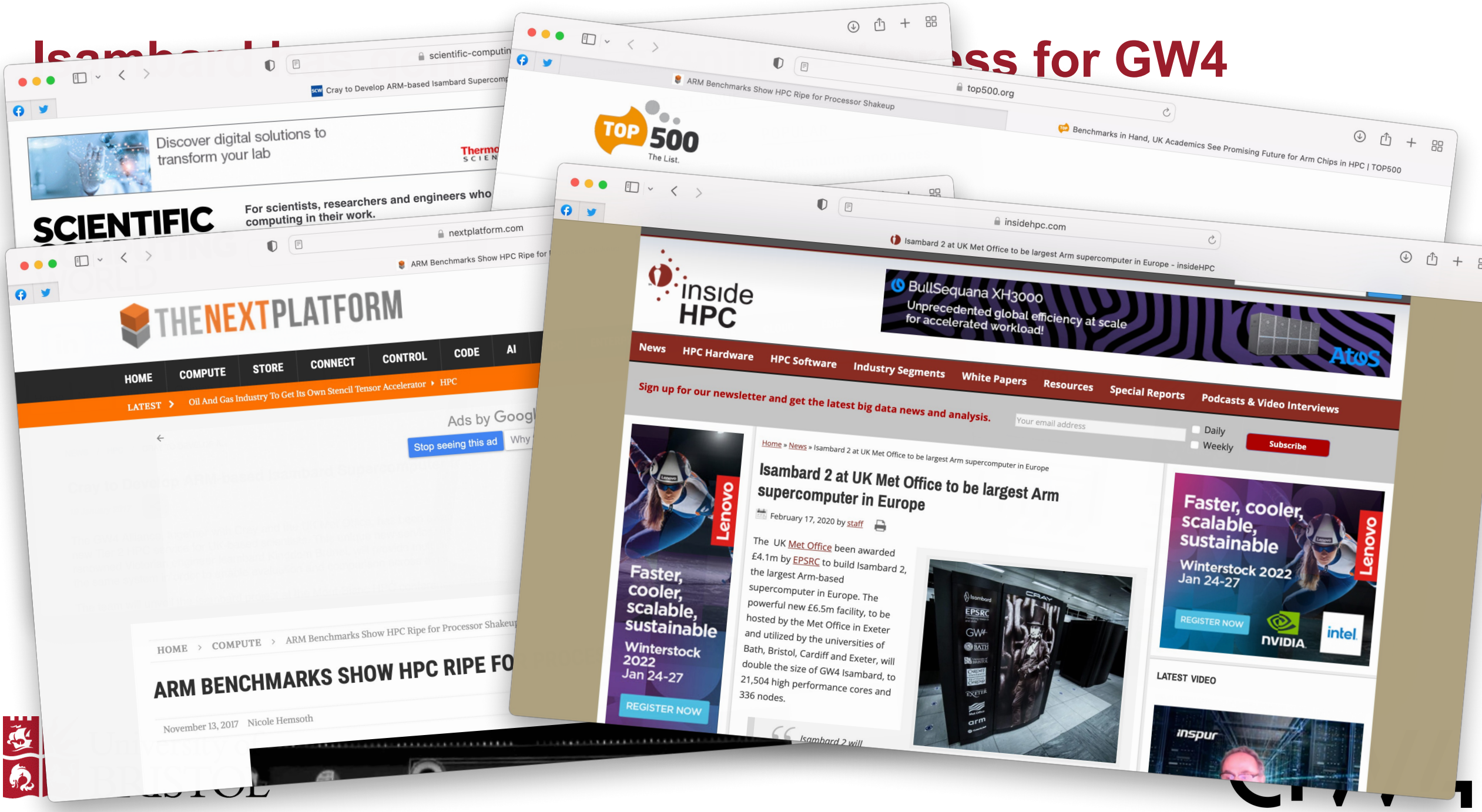


System utilisation typically >90%.
98.9% uptime.

University of BRISTOL

GW4

Isambard ... ess for GW4

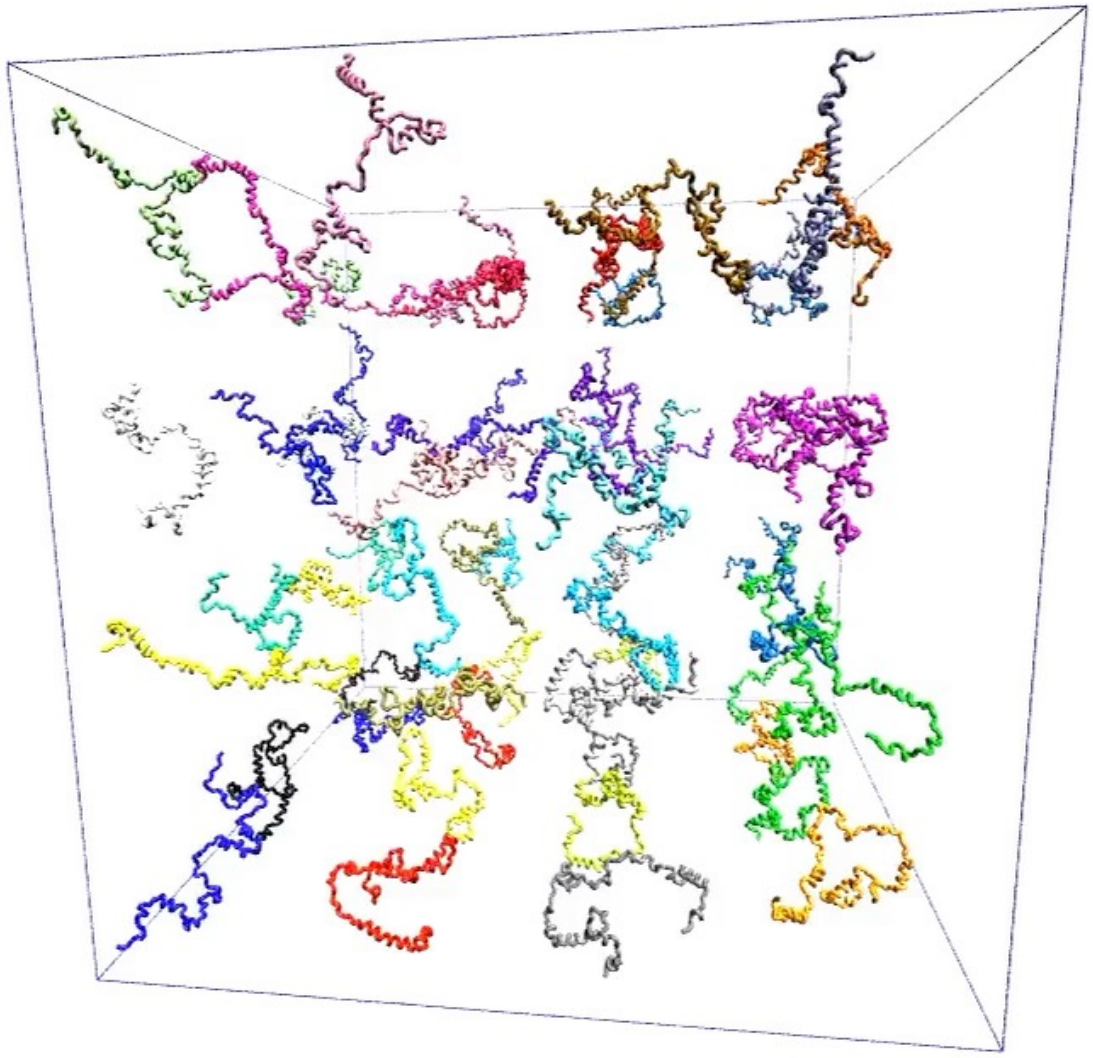


Isambard Arm-based training, hackathons, workshops etc.

- Over 60 tutorials, hackathons and workshops have been run on Isambard since the service's inception, with many at top-tier international conferences, including IEEE/ACM SuperComputing and International Super Computing (ISC)
- To date, over 1,000 international scientists, researchers and developers have attended tutorials, workshops and hackathons run using Isambard

Bristol Isambard case study: molecular simulations of factors behind Parkinson's and osteoporosis

- Bristol researchers have been running molecular level simulations on Isambard to understand the mechanisms behind Parkinson's disease, and to find ways to treat osteoporosis
- Their simulations on Isambard have shown how the alpha-synuclein protein can start to clump together in the human brain, a key feature of Parkinson's disease



University of BRISTOL

Simulations showing how the alpha-synuclein protein can start to clump together in the human brain

GW4


CRAY
CRAY
Isambard
Engineering and Physical Sciences Research Council
GW4
University of BATH
University of BRISTOL
CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD
University of EXETER
Met Office
arm
MARVELL
THE GREAT BRITAIN

- A64fx system of 72 nodes added in Sep 2020
- Enabled porting and optimization for SVE
- Supported world's first open SVE hackathon at SC20

University of BRISTOL

GW4

Isambard 3 coming this year

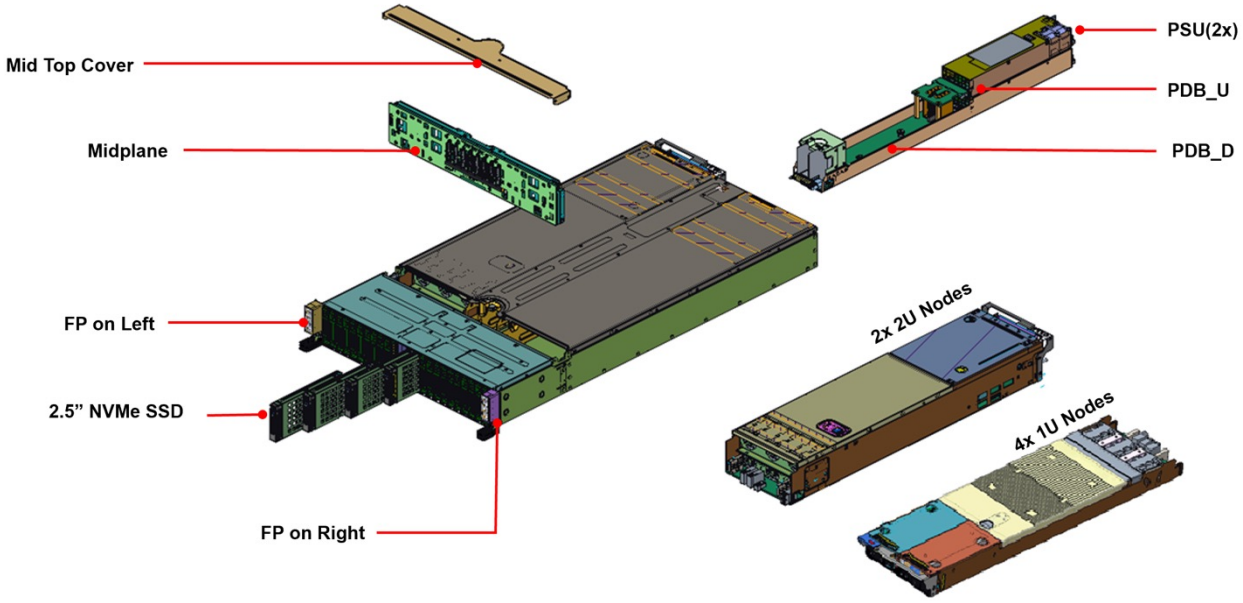
- **£9.2M** CAPEX funding, +£6.1M OPEX for 4 years of operation
  - Significantly expanded support team of 6 RSEs, 2 full-time sys admins
- Isambard 3 will be one of the first supercomputers based on NVIDIA's new '**Grace**' Arm CPUs
- **55,000+** cores, 2-3 PetaFLOP/s, 6X faster and more energy efficient than Isambard 2
- Liquid cooling where possible for a low PUE, waste heat reuse potential
- Each node has 144 cores at 3.5GHz and ~1Tbyte/s memory bandwidth to 256GB of DRAM
- Comes with a complete set of optimized NVIDIA libraries, **including for AI/ML**
- Will also have some **Grace+Hopper GPU nodes**
- On target for installation around the end of 2023

University of BRISTOL

https://www.bristol.ac.uk/news/2023/may/isambard-3-investment.html

GW4

Isambard 3 NVIDIA ‘Grace’ CPU superchip



Competitive with best in class CPUs in 2023 in both performance and energy efficiency.

Using NVIDIA’s whitebox air-cooled servers with water cooled doors.

This is the first time that Isambard’s Arm processors will come from a mainstream HPC chip vendor.

University of BRISTOL

GW4

- NVIDIA Grace-Grace 144 core board for Isambard 3
- Approx. A3 in size and 500W
- 384 of these in Isambard 3
- System delivered by HPE

University of BRISTOL

GW4

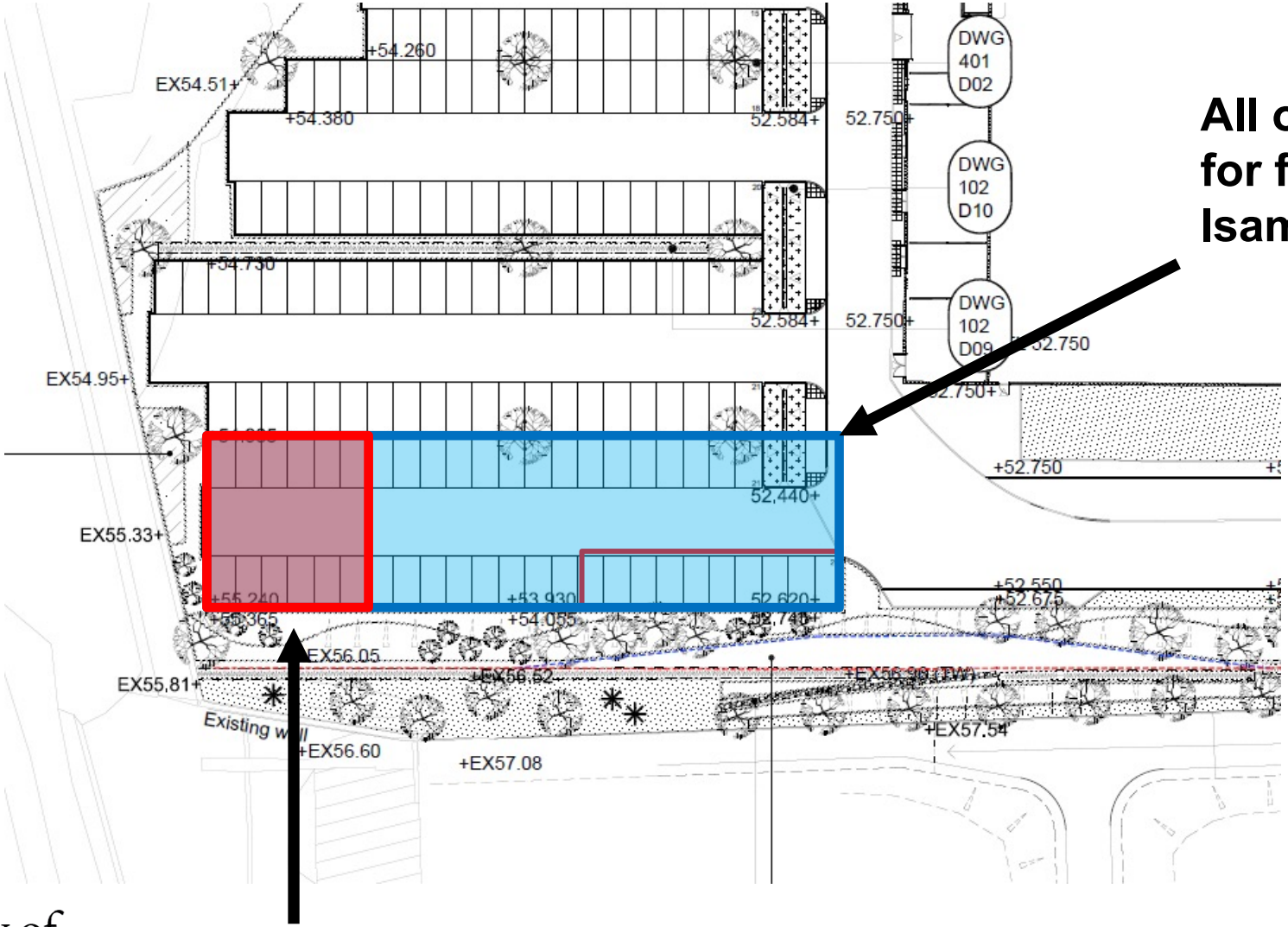
Isambard 3 site: the National Composites Centre in Bristol

University composite materials research centre
Close to M4 and Parkway Station.
Significant room for future expansion.

University of BRISTOL

GW4

NCC Site for Isambard 3



All of this region available for future expansion of the Isambard service.

Area initially used for Isambard 3

University of BRISTOL

GW4

University of BRISTOL

GW4

University of BRISTOL

All of Isambard 3 will fit in a single, agile, energy efficient Modular Data Centre (MDC). Easy to scale up in an agile manner.

GW4


GW4
UK Research and Innovation
NVIDIA
arm

Great Western 4 Isambard 3 summary

- The new service will be one of the most energy efficient CPU-based systems in the world, 5-6X better than Isambard 2
- We expect Grace to be performance competitive with the best x86 processors in 2023/24
- Energy efficiency-wise, should be class leading
- DRAM-sized memory per node, but HBM-like bandwidth
- Very flat NUMA structure should enable excellent ease of use
- On the floor late 2023, in production early 2024

ISC2023- AHUG WORKSHOP

MEMORY PREFETCHING EVALUATION USING GEM5 SIMULATIONS

NAM HO, CARLOS FALQUEZ, ANTONIO PORTERO, ESTELA SUAREZ,

NOVEL SYSTEM ARCHITECTURE DESIGN, JÜLICH SUPERCOMPUTING CENTRE, FORSCHUNGSZENTRUM JÜLICH

DIRK PLEITER

PDC CENTER FOR HIGH PERFORMANCE COMPUTING, KTH ROYAL INSTITUTE OF TECHNOLOGY

OUTLINE

- Motivation
- Hardware memory prefetching
- Simulation methodology
- Application selection and memory access pattern analysis
- Prefetch evaluation
- Conclusion & future study

MOTIVATION

- Emerging HPC applications are demanding advanced computing systems
  - Need both high **processing capability** as well as high **memory bandwidth**
- Recent innovations in the HPC sector
  - Arm-based high-end processors with SVE technology have entered the HPC sector (e.g. Fugaku supercomputer)
  - New memory technologies (e.g. HBMx) make available high bandwidths
- The memory wall
  - Scaling memory bandwidth with processing capability remains a challenge
  - Memory prefetching is well-known technique to hide long-latency memory accesses by predicting the data accesses and preloading data ahead in the cache that is expected to be requested in the near future

- Work objectives
  - Investigate **the effect of hardware prefeching techniques** on scientific applications in recent Arm-based high-end processors
  - Leverage gem5 capabilities and **developed a gem5-model of the latest Arm Neoverse V1 design with HBM2** based on the recently released Graviton3

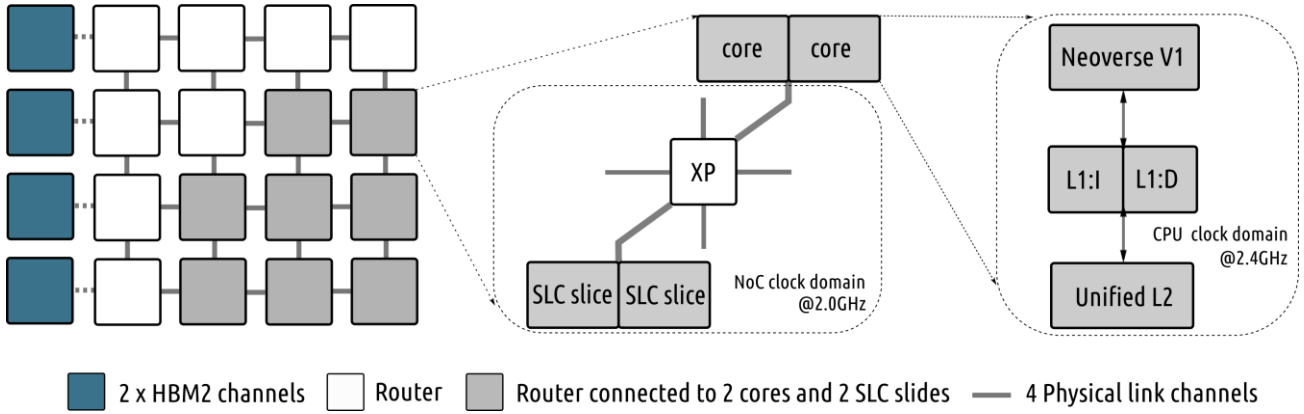
HARDWARE MEMORY PREFETCHING

- A well-known technique to eliminate processor stalls due to long-latency memory accesses
  - Reduce cache misses by predicting the data accesses and fetching ahead data into cache that is expected to be requested in the near future
- Memory access pattern prediction
  - Stride (or delta) access pattern
    - Constant stride sequence: [A,A+k,A+2k,…]
    - Multi-delta sequence: [A,A+k,A+m,…B,B+k,B+m]
  - Address repetition pattern
    - [A,B,C,…,A,B,C]
- Prefetch effectiveness
  - Depend on prediction accuracy, prefetch coverage
  - Aggressive prefetching

** Sparsh Mittal. 2016. A Survey of Recent Prefetching Techniques for Processor Caches. ACM Comput. Surv. 49, 2, Article 35 (Nov. 2016).*

SIMULATION METHODOLOGY: THE GEM5-MODEL OF NEOVERSE V1 ARCHITECTURE

- The 16-core Neoverse V1 architecture (**ExpArch**)
  - NoC:
    - Ruby Garnet + AMBA–CHI
    - 4x4 Mesh
    - 4 physical link channels for 4 VNETs
  - 16 cores + 16x1MB-SLCs
    - 2xSVE engines per core
    - Core's clock: 2.4GHz
    - System/NoC clock: 2.0GHz
  - Memory
    - HBM2 (8 x 38.4GB/s/channels)



North West quadrant of the modeled architecture: The left-most figure shows 16 cores, 16 SLC slices, and 8 HBM2 channels placed in a 2D mesh topology; The middle of the figure outlines a router architecture connecting to 2 cores and 2 SLC slides; The right most figure outlines core's cache hierarchy

Lilia Zaourar et.al. "*Multilevel simulation-based co-design of next generation HPC microprocessors.*" PMBS 2021: 18-29

SIMULATION METHODOLOGY: QUANTITATIVE COMPARISON WITH REAL PLATFORMS

- To get better confidence in the gem5-model, we quantitatively compared the simulation results with N1SDP, and Graviton3
  - Two additional gem5-models
    - Gem5-model (N1) for N1SDP,
    - Gem5-model (G3) for Graviton3

- Microbenchmarking
  - FMLA-bench: A simple benchmark developed to test the maximum number of FP operations on an SVE machine

Overview of three system architectures

	Core-Arch	NoC	Memory
Graviton3	64xNeoverse-V1	AMBA-CHI	DDR5 (8x38.4GB/s/channel)
N1SDP	4xNeoverse-N1	AMBA-CHI	DDR4 (2x25.6GB/s/channel)
ExpArch	16xNeoverse-V1	AMBA-CHI	HBM2-Model (8x38.4GB/s/channel)

FMLA-bench results

Platform	Work size	Loop	CPU clock	Glop/s	Flop/cycle (% of peak)	Cycles	Retired instrs	SIMD instrs
gem5-model (G3)	1000000	5	2.6	41.5	15.96 (99.7)	7.51E+07	1.8E+08	1.5E+08
Graviton3	1000000	5	2.6	41.5	15.96 (99.8)	7.76E+07	1.8E+08	1.5E+08
gem5-model (N1)	1000000	5	2.6	20.8	8 (100)	1.50E+07	3.30E+07	3.00E+07
N1SDP	1000000	5	2.6	20.8	8 (100)	1.50E+07	3.30E+07	3.00E+07
gem5-model (ExpArch)	1000000	5	2.4	38.3	15.96 (99.7)	7.51E+07	1.8E+08	1.5E+08

Simulation results match with that on the real platforms

SIMULATION METHODOLOGY: QUANTITATIVE COMPARISON WITH REAL PLATFORMS (CONT.)

- To get better confidence in the gem5-model, we quantitatively compared the simulation results with N1SDP, and Graviton3

- Microbenchmarking
  - Triad-kernel (from STREAM benchmark)

```
a[i] = S*b[i] + c[i]
```

  - Read-only kernel

```
sum += a[i]
```

N1SDP comparison with prefetchers disabled

Platform	L1D cache miss	L2 cache miss	SLC cache miss	Retired instrs	L1D cache access	Bandwidth (GB/s)
Read-only kernel - With prefetchers disabled						
gem5-model (N1)	4.00E+07	5.03E+06	5.03E+06	1.60E+08	4.04E+07	1.83
N1SDP	5.00E+06(*)	5.00E+06	5.00E+06	1.60E+08	4.00E+07	1.77
Triad kernel - With prefetchers disabled						
gem5-model (N1)	8.61E+07	1.51E+07	1.51E+07	2.80E+08	1.21E+08	2.95
N1SDP	1.00E+07(*)	1.00E+07(**)	1.00E+07(**)	2.80E+08	1.20E+08	2.57
gem5-model (N1) (Stride=4)	2.50E+07	1.51E+07	1.50E+07	7.01E+07	3.10E+07	2.15
N1SDP (Stride=4)	1.50E+07	1.50E+07	1.50E+07	7.00E+07	3.00E+07	1.68

(*) Diff for a factor 8x due to cache-line refill counting in N1SDP
(**) Diff. for a factor of 1.5x due to the effect of write-streaming mode in N1SDP

Three platforms comparison with prefetchers enabled

Platform	1-thread (GB/s)	4-threads (GB/s)	16-threads (GB/s)
gem5-model (N1)	13.1	30.83	-
N1SDP	19.72	29.29	-
gem5-model (G3)	39.11	115.75	173.66
Graviton3	44.86	162.04	214.04
ExpArch	38.01	117.16	169.04

SELECTED HPC APPLICATION KERNELS & THE CHARACTERIZATION

	Description	(Pseudocode) Kernel code (via SVE optimization)	Access pattern

TRIAD	The triad kernel from STREAM benchmark	`// Triad`
		`for i = 0; i < N; i++)`
		`  a[i] <- S* b[i] + c[i]`
			Constant stride

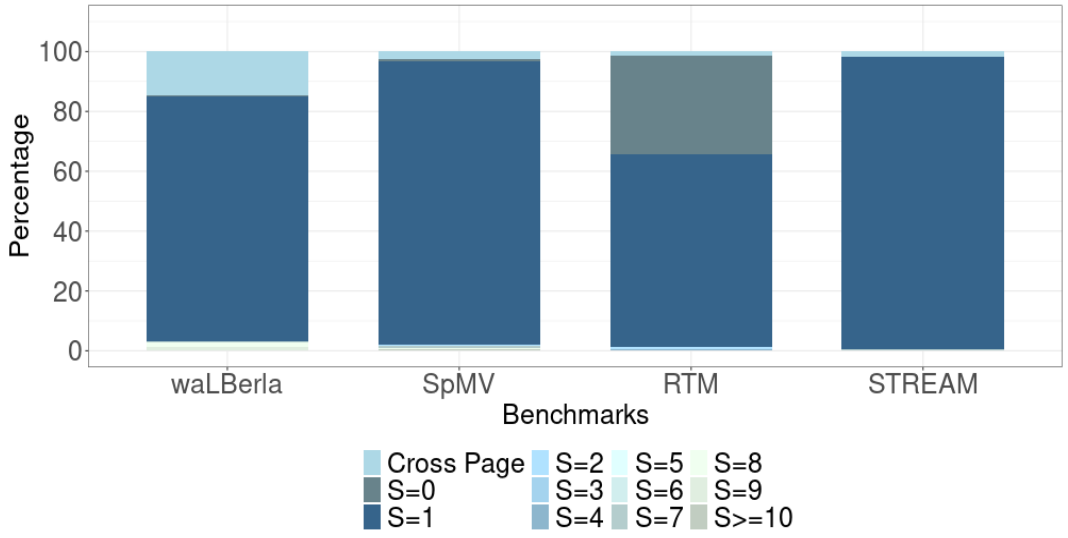
SPMV (MiniFE)	Matrix-vector product $y = y + Ax$, where x and y are dense vectors
	We evaluated the hand-optimized version using SVE intrinsics (*)
	
	(*) Bine Brank et.al. "Porting Applications to Arm-based Processors. In 2020 CLUSTER. 559–566.
		`// Sliced ELLPACK format`
		`for (s = 0; s < N; s += S)`
		`  for (r = s; r < s + S; r++)`
		`  for (i = slice_start[s/S] + r - s;`
		`  i < slice_start[s/S + 1]; i += S)`
		`  y[r] += x[column[i]] * value[i];`
			Constant stride + random access with indirect indexing

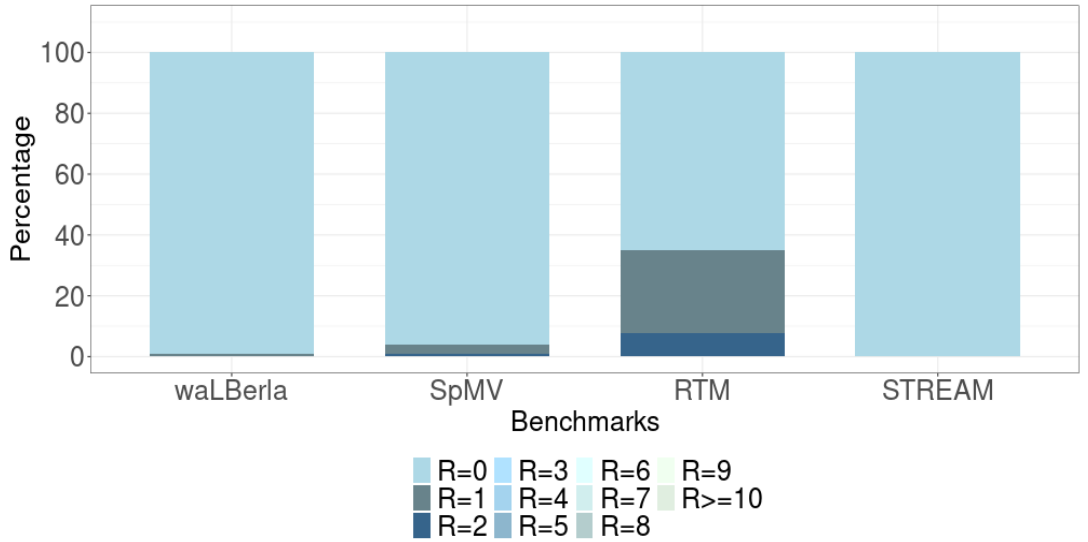
waLBerla	A framework for computational fluid dynamics simulations based on the LBM. We used the waLBerla use-case *UniformGridBenchmark*	`// Stencil code` `D3Q19 model implementation`	Multi-stride

RTM	Reverse-Time Migration (RTM) is a well known method used in the reconstruction of geological subsurface structures.
	We evaluated Isotropic wave equation (ISO) kernel
		`// Stencil code`
		`8th order Laplacian Stencil,`
		`a 3D 27-point stencil`
			Multi-stride

Benchmark	I_{mem} (Byte)	I_{fp} (Flop)	AI= (flops/byte)	Size	Memory footprint (MiB)	Miss-rate(%) L2	Miss-rate(%) SLC	MPKI L2	MPKI SLC	IPC	BW Util. (%peak)
waLBerla	N.76.8	N.189	0.31	N=96.96.96	256	97.53	87.57	46.88	41.05	10.93	31.71
SpMV	N_{nnz}.(8+4)	N_{nnz}.2	0.17	N=63.63.63	80	91.3	92.82	58.1	53.92	8.1	23.6
RTM	16.2	34	2.1	N=128.256.256	128	93.39	44.12	41.17	18.16	12.41	13.57

HPC APPLICATION KERNELS & CHARACTERIZATION (CONT.)





* STRIDE pattern (Spatial Locality [1]):
- Calculation by searching back for a window of **`W`** memory accesses and calculate stride **`s`** as the minimum distance between the current access address and its nearest neighbor among those in the window.
  - **`W`** = 64 (units in 64-byte cache-line).

* REUSE pattern (Temporal Locality[1]):
- Calculation by searching back for a window of **`L`** memory accesses and calculate the number of times access address repeated
  - **`L`** = 64 (units in 64-byte cache-line).

[1] J. *Weinberg, M. O. McCracken, E. Strohmaier and A. Snavely, "Quantifying Locality In The Memory Access Patterns of HPC Applications," SC '05, pp. 50-50, doi: 10.1109/SC.2005.59.*

HARDWARE PREFETCHERS USED FOR EVALUATION

Hardware prefetcher configuration at L2 for evaluation						
HW Prefetcher (*)	Description	Trigger on access	Cache snoop	Queue size	Specific parameters	Aggressive prefetching parameters
Tagged	Next-line prefetching	True	True	64	-	Varying prefetch degree [1..32]
Stride	Constant PC-stride detection				table entries = 128	
AMPM[1]	Delta-pattern detection				hot zone size=2KiB, map table entries = 256	
SPP[2]	Delta-pattern detection				prefetch threshold = 0.25, lookahead threshold=0.01, signature table entries=256, pattern table entries=4096	

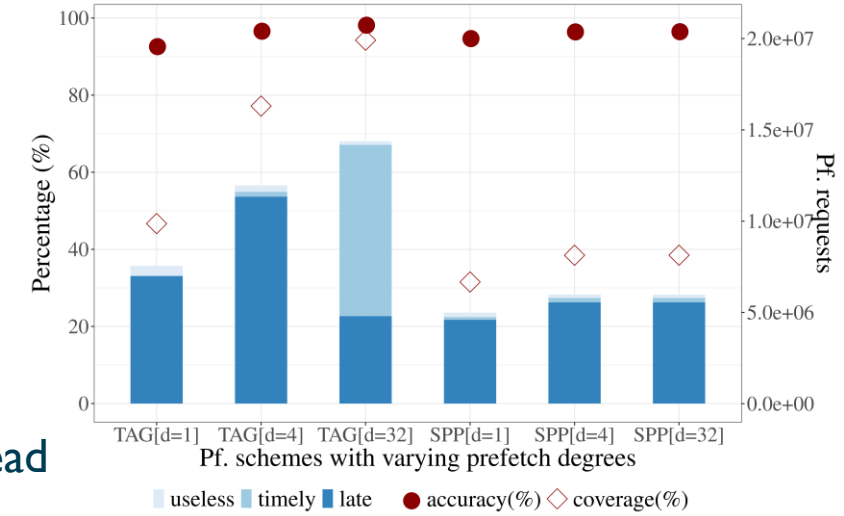
(*) Prefetcher names are taken from the gem5

[1] Ishii, Y. et. al. "Access map pattern matching for high performance data cache prefetch" Journal of Instruction-Level Parallelism, 13, 1-24 (2011).
[2] Kim J. et. al. "Path confidence based lookahead prefetching" MICRO-49, Article 60, 12 (2016).

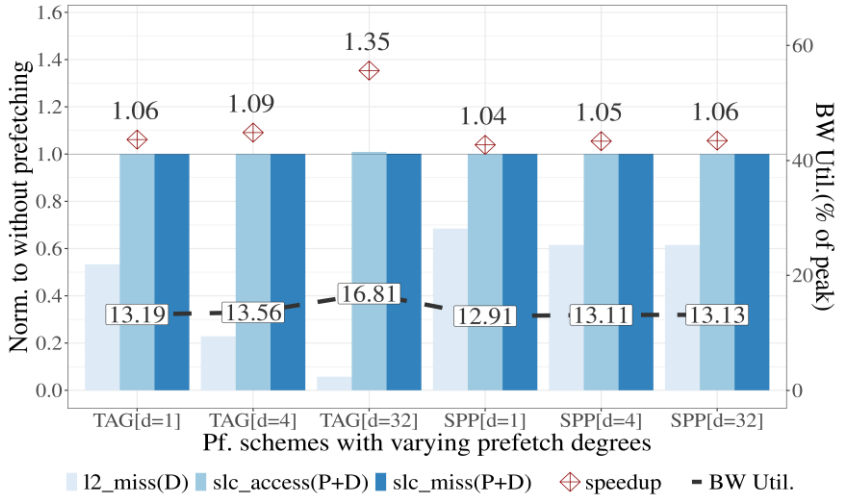
PREFETCH COUNTERS & METRICS

- *Number of prefetches*: A counter tracks number of prefetches actually sent to the next memory level.
- *Late prefetches*: A counter tracks number of accurate prefetch requests are being in the in-flight memory request buffer (TBE) waiting for the returned data by the time needed by the demand miss.
- *Timely prefetches*: A counter tracks number of accurate prefetch requests fetching the needed data by a demand miss in time.
- *Useless prefetches*: A counter tracks number of inaccurate prefetches, for which the fetched data in cache are not used.
- Accuracy: $\frac{timely\ prefetches + late\ prefetches}{number\ of\ prefetches}$
- Coverage: $\frac{timely\ prefetches + late\ prefetches}{timely\ prefetches + late\ prefetches + demand\ misses}$

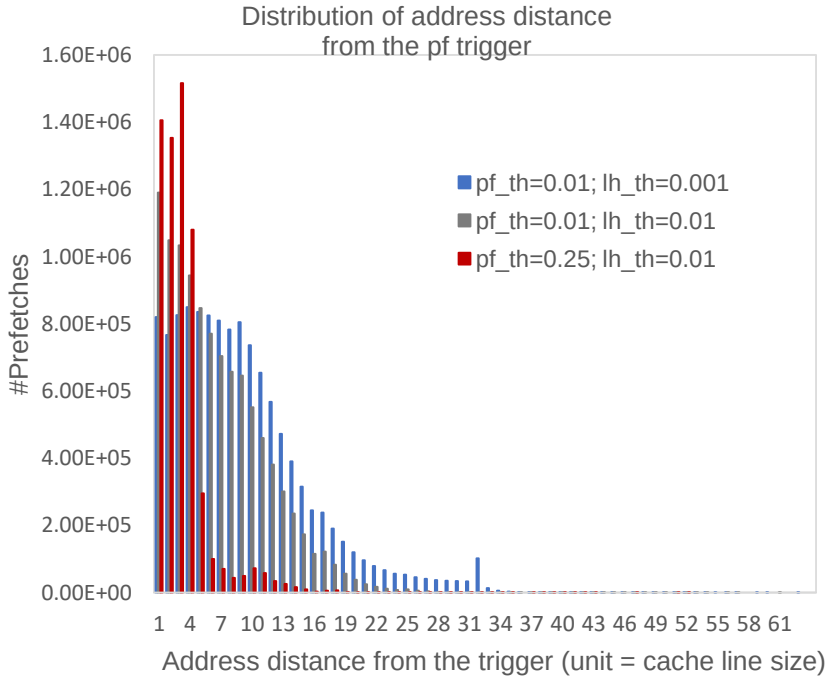
EFFECTS OF AGGRESSIVE PREFETCHING

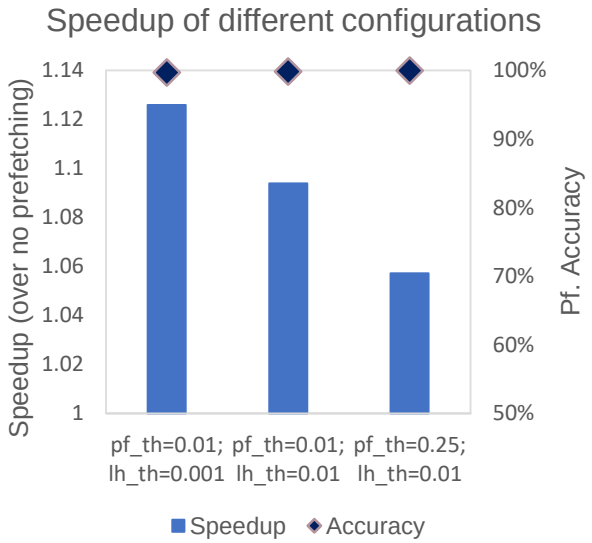


1-thread



- Triad kernel (STREAM benchmark)
  - **BW utilization saturated without prefetching** (74% of peak; **16-thread** evaluation)
  - High aggressiveness shows performance improvement for **single-thread**



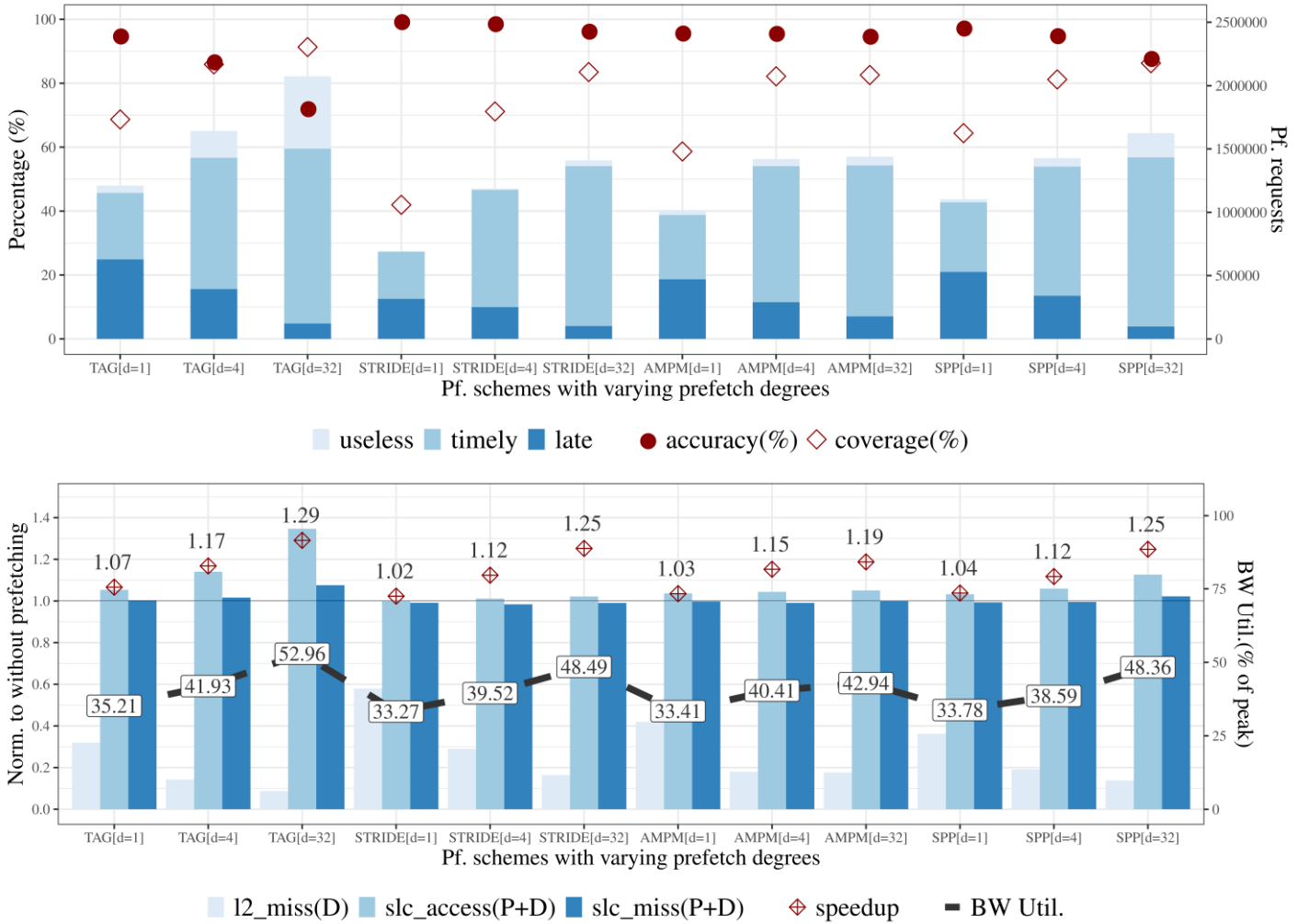


SPP exploration with diff. threshold configurations

EFFECTS OF AGGRESSIVE PREFETCHING (CONT.)

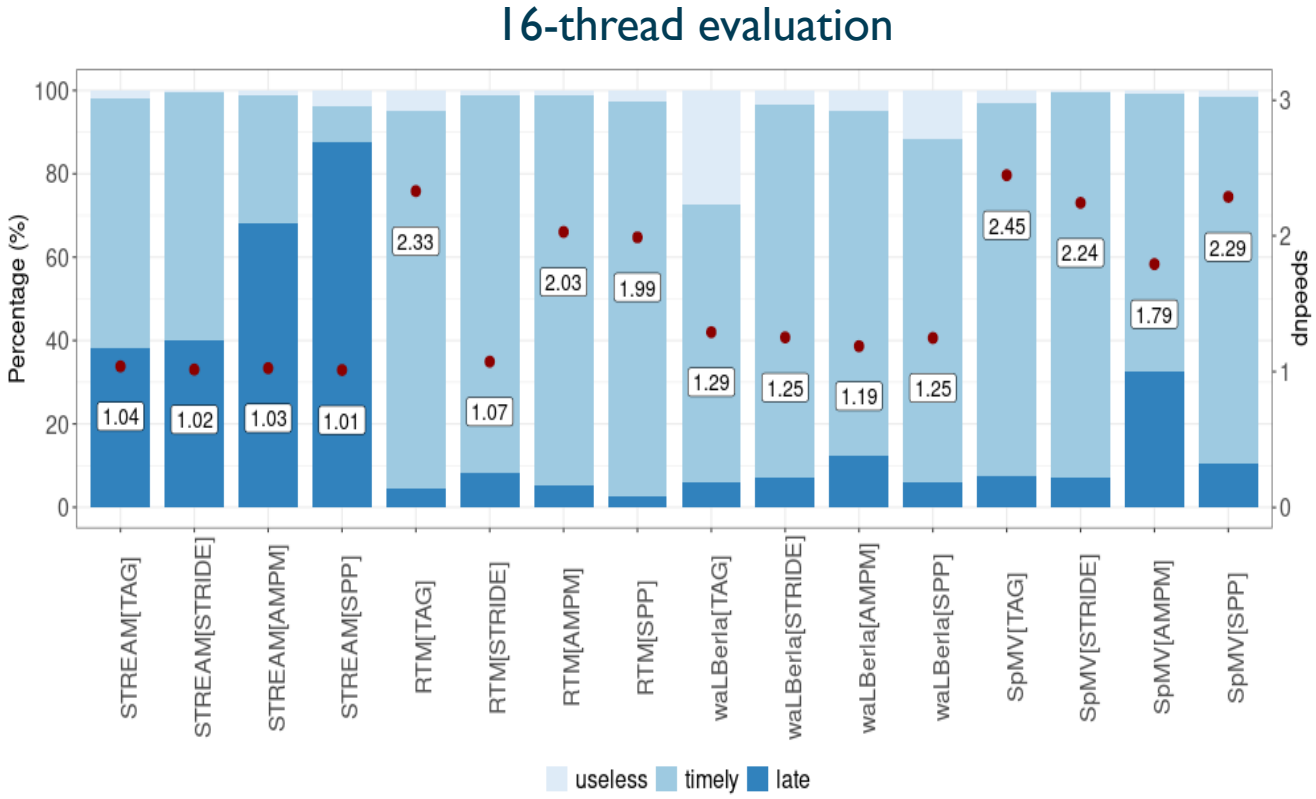
- WaLBerla
  - Strongly benefits from aggressive prefetching
    - Low conservative prefetching (e.g. degree = 1) does not work well (~50% late prefetches)
    - Prefetching with degree = 32, all prefetchers show coverage of over 80%, and the Next-line (TAG) delivers speedup up to 1.29
  - More useless prefetches are observed with the Next-line (TAG)

16-threads

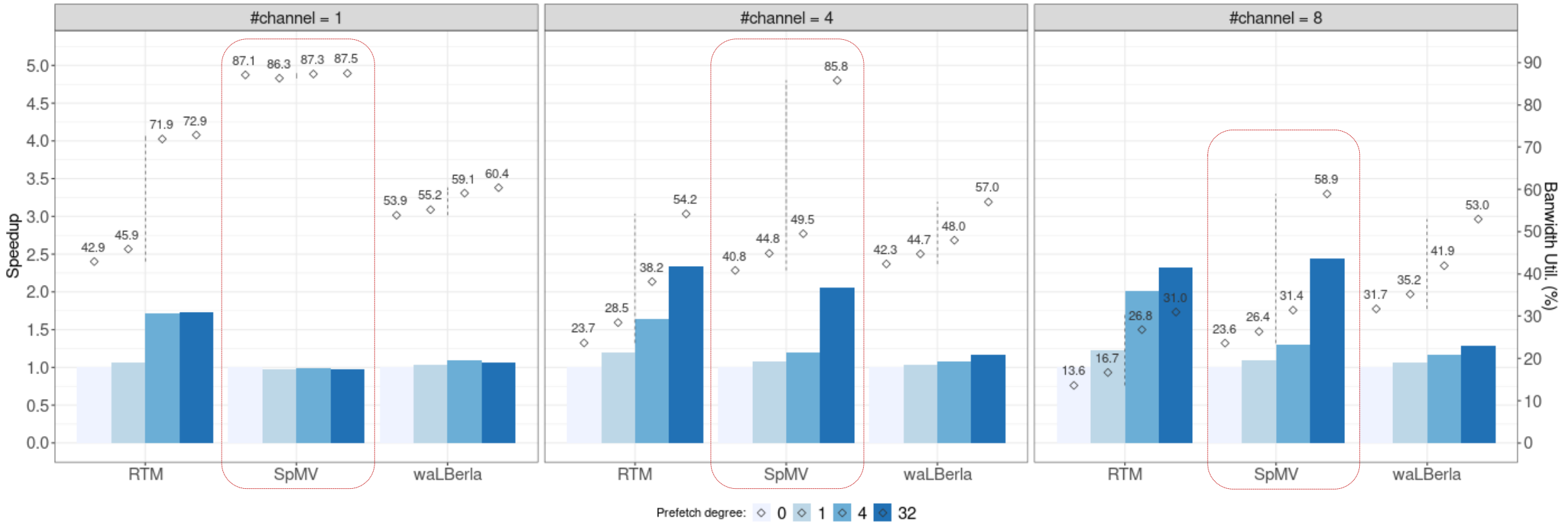


PERFORMANCE IMPROVEMENT

- All selected prefetchers are effective showing speedups that highly correlate with the classified prefetches
- Among prefetchers, Next-line provides the highest effectiveness, although, this prefetcher shows the highest useless prefetches
- SpMV gains the most up to a speedup of 2.45×



SENSITIVITY ANALYSIS-EFFECT OF MEMORY BANDWIDTH



Higher speedup gains when increasing #channels (Benefit of aggressive prefetching)

CONCLUSIONS

- We have evaluated the effects of memory prefetching of scientific application on high-end Arm processors with attached HBM memory using a simulation methodology based on a gem5
- Lessons learnt
  - Memory access patterns from the stencil-style and SpMV codes commonly used in today's scientific applications benefit from spatial address-correlation prefetching schemes
  - Aggressive prefetching evaluated for the kernel codes (e.g. SpMV) is sensitive to memory bandwidth: more available bandwidths lead to higher speedups
- Future studies
  - In-deep analyze memory access patterns (delta, address repetition)
  - Extend the evaluation for other HPC application kernels

- Thanks for your attention!

- Acknowledgements
  - Tiago Muck (ARM Ltd.)
  - Stepan Nassyr, Bine Brank (JSC)

EPI PARTNERS

EPI FUNDING

This project has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 EPI-SGA2. The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

STUDYING DIFFERENT BFS ALGORITHM IMPLEMENTATIONS WITH GEM5

Carlos Falquez[1], Nam Ho[1], Antonio Portero[1], Estela Suarez[1], Dirk Pleiter[2]

[1]Novel System Architectures Design, Jülich Supercomputing Centre, Forschungszentrum Jülich

[2]PDC Center for High Performance Computing, KTH Royal Institute of Technology

AHUG Workshop @ ISC23
May 25, 2023

STUDYING DIFFERENT BFS ALGORITHM IMPLEMENTATIONS WITH GEM5

Outline

- Motivate use of gem5 simulator as tool for developers assesing performance on different architectures
- Study effects of different architectural parameters:
  - Instruction Width, Number of LS Units
  - System Level Cache (SLC) Sizes
  - Memory Controller Latency
- BFS algorithm as illustrative case:
  - Algebraic (vectorized) and Vertex-Based implementations with different memory access patterns
  - Sensitive to different architectural parameters
- Understand differences in performance by comparing results on base and modified architectures.

STUDYING GRAPH ALGORITHMS WITH GEM5

Graph Algorithms

- Graph algorithms fundamental to a wide variety of applications
  - Machine learning
  - Network analysis
  - Bioinformatics
  - ...
- Data-dependent non-sequential access patterns
- Opportunities for optimization

Exploration with gem5

- gem5 is a modular cycle accurate computer architecture simulator.
- Use gem5 to study impact of microarchitectural parameters, cache sizes and memory latency on different implementations.

BREADTH-FIRST SEARCH (BFS)

Edge/Vertex-based approach

- Traverse graph advancing vertex frontier over connecting edges
- Ligra [4] is a state-of-the-art shared memory implementation of graph edge/vertex-traversal, optimized for both sparse and dense frontiers.
- The graph is represented by a list of edges sorted by adjacency. Each vertex stores a pointer to the start of its edges. (Adjacency array format).
- Graph structure determines memory access locality

BREADTH-FIRST SEARCH (BFS)

Algebraic approach

- Graph traversal can be represented as an iterative matrix-vector multiplication over various algebraic semirings [1].
- Graph characteristics determine matrix sparsity.
- Ongoing effort of implementing and optimizing SpMV based graph operations (see e.g. [2]).

Prototype vecBFS implementation

- Vectorized SpMV implementation based on SELL-C-sigma [3] sparse matrix format.
- Hand coded using SVE intrinsics.
- Strided memory access pattern.

VECBFS PERFORMANCE COMPARED TO LIGRA

Single thread speedup for varying problem size on AWS Graviton3 (average)

Scale	vecBFS [cycles]	Ligra BFS [cycles]	Speedup	A size	x size
12	1.88E+05	2.67E+05	1.42	512kB	16kB
13	3.55E+05	5.43E+05	1.53	1MB	25kB
14	7.63E+05	1.08E+06	1.41	2MB	50kB
15	1.63E+06	2.18E+06	1.34	4MB	100kB
16	3.60E+06	4.11E+06	1.14	8MB	180kB
17	7.92E+06	8.34E+06	1.05	16MB	350kB
18	1.82E+07	1.69E+07	0.93	32MB	680kB
19	4.55E+07	3.53E+07	0.78	64MB	1.3MB

- Measured average time-to-solution (TTS) for single-thread benchmark on c7g.metal
- Speedup relative to Ligra drops when matrix size goes beyond available L3 size (32MB).

GEM5 O3CPU MODEL

O3 Pipeline Model and Functional Units

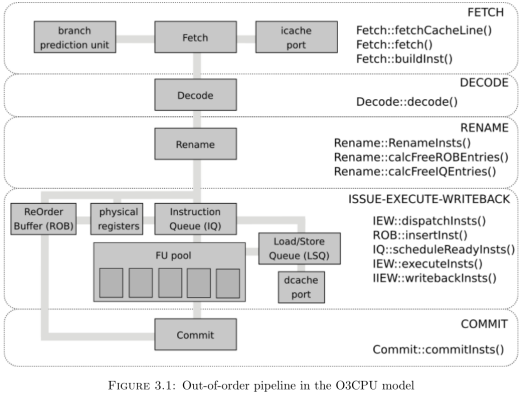


FIGURE 3.1: Out-of-order pipeline in the O3CPU model

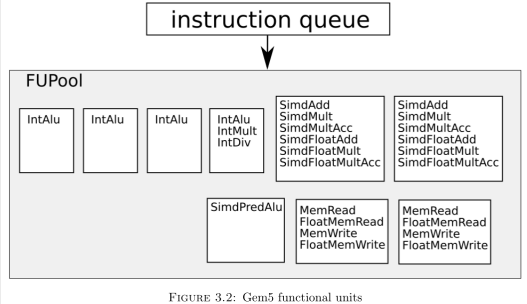
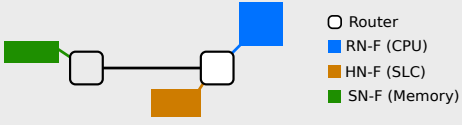


FIGURE 3.2: Gem5 functional units

Figures taken from: Bine Brank, PhD Thesis (2023)

GEM5 MODEL FOR SINGLE THREAD BENCHMARKS

Base configuration



CPU	1x 64-bit ARM (AArch64)
SVE	2x256
Private L1	64kB 4-Way
Private L2	1MB 8-Way
SLC	1x2MB 16-Way
Memory	30GB/s (Simple Memory)
NOC	Arm CHI Ruby + Garnet

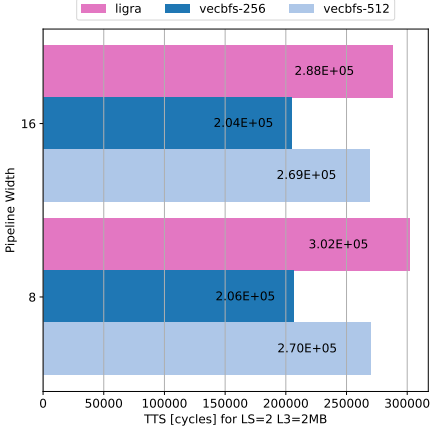
Simulation Parameters

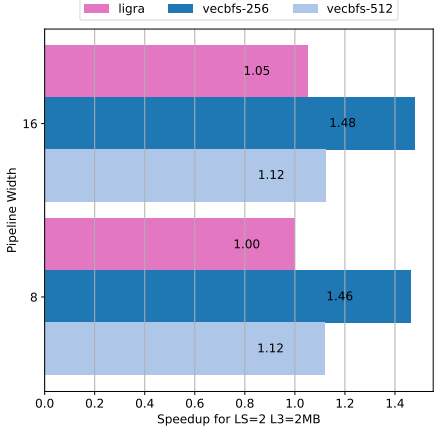
SVE Vector length	256,512
Pipeline Width	8,16
Load/Store Units	2,4
SLC Sizes	2,4,8 MB
Memory Latency	5..300ns

- Workloads: Scale 12,15 Kronecker Graphs (around 512kb and 4MB respectively)

PERFORMANCE COMPARISON: PIPELINE WIDTH

Small Problem Size S=12 (512kB)

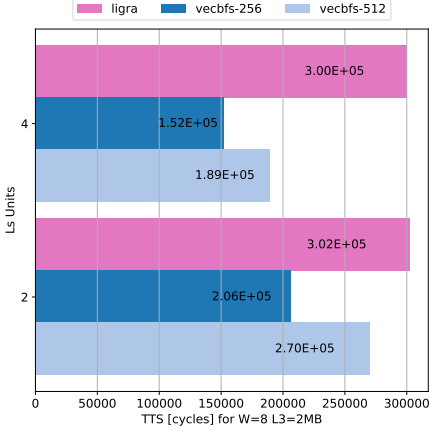


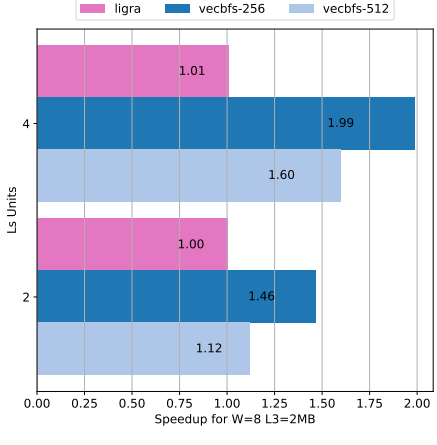


JÜLICH Forschungszentrum

PERFORMANCE COMPARISON: LS UNITS

Small Problem Size S=12 (512kB)

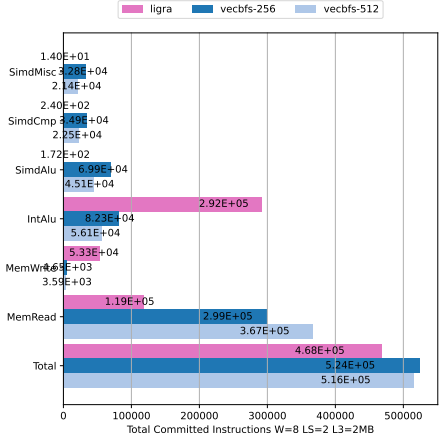


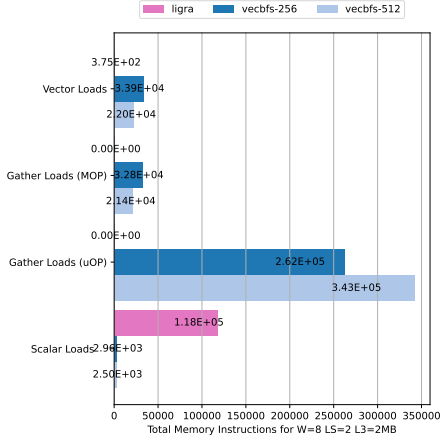


JÜLICH Forschungszentrum

COMMITTED INSTRUCTIONS

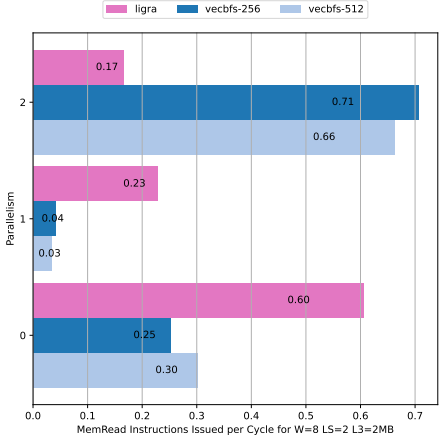
Small Problem Size S=12

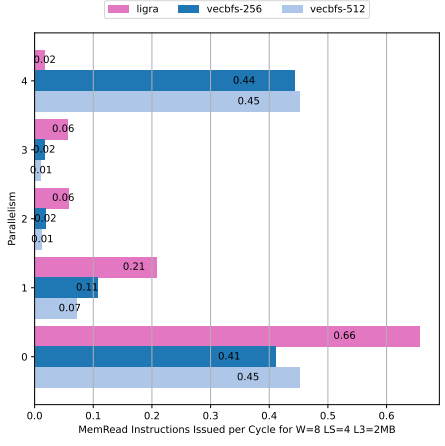




PARALLEL ISSUED MEMREAD INSTRUCTIONS

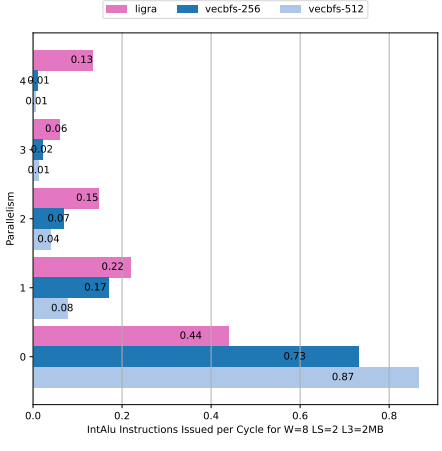
Effect of LS Units on ILP for S=12

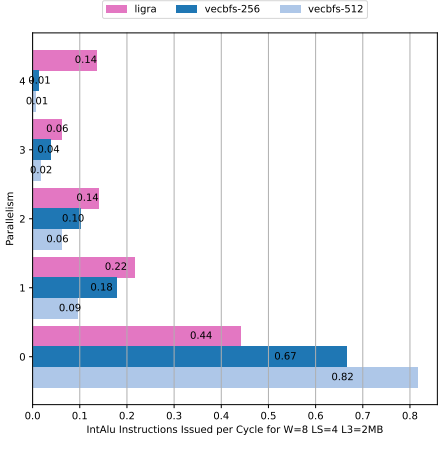




PARALLEL ISSUED INTALU INSTRUCTIONS

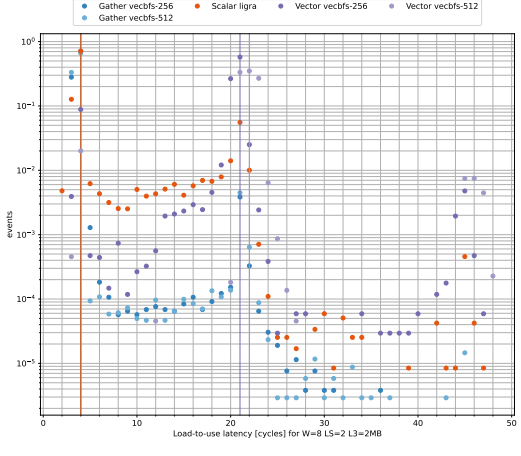
Effect of LS Units on ILP for S=12





LOAD-TO-USE LATENCIES

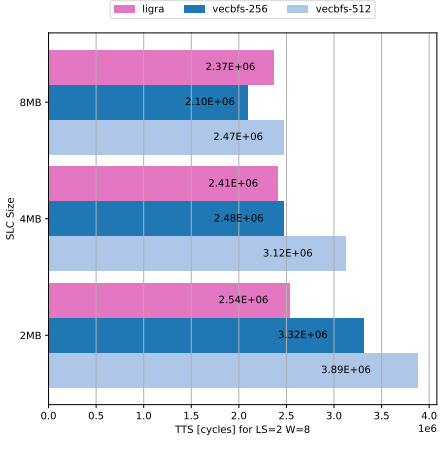
Small Problem Size S=12 (512kB) and SLC=2MB

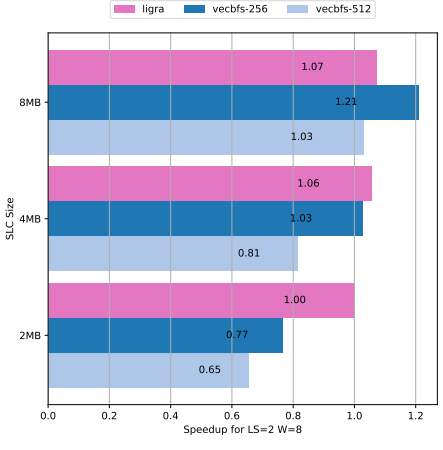


JÜLICH Forschungszentrum

PERFORMANCE COMPARISON: SLC SIZE

Large Problem Size S=15 (4MB)

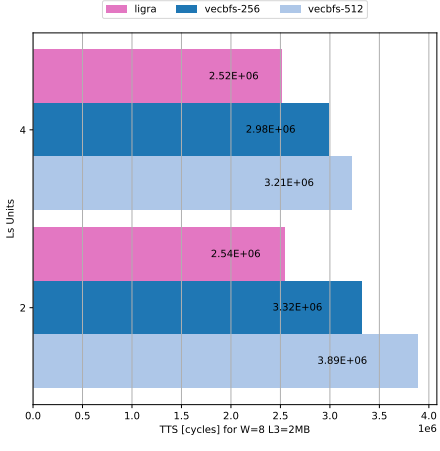


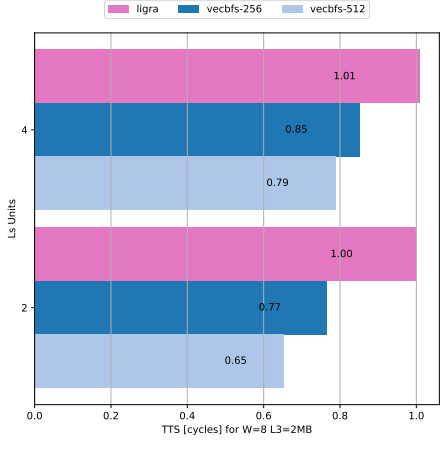


JÜLICH Forschungszentrum

PERFORMANCE COMPARISON: LS UNITS

Large Problem Size S=15 (4MB)

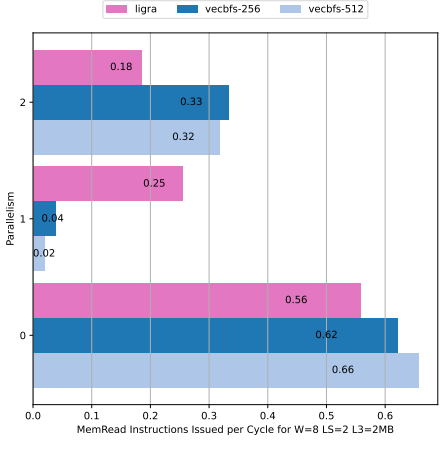


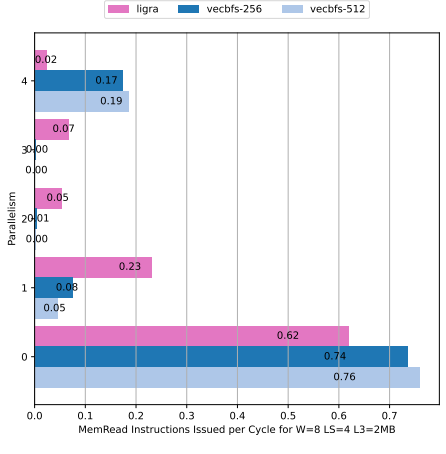


JÜLICH Forschungszentrum

PARALLEL ISSUED MEMREAD INSTRUCTIONS

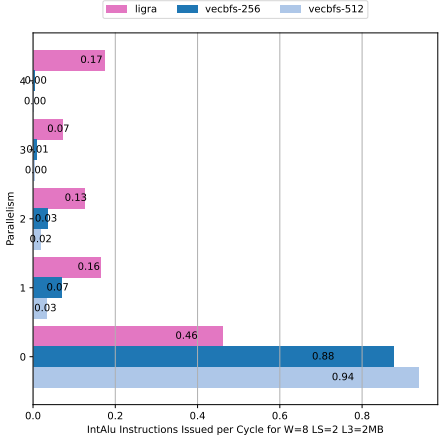
Effect of LS Units on ILP for S=15

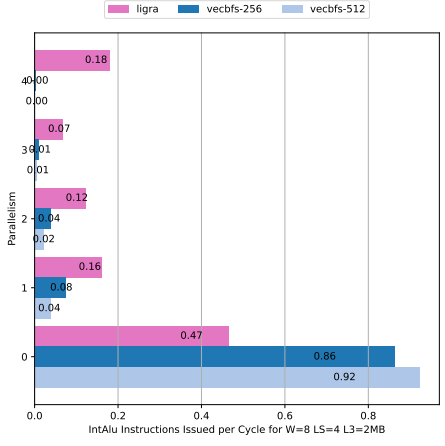




PARALLEL ISSUED INTALU INSTRUCTIONS

Effect of LS Units on ILP for S=15

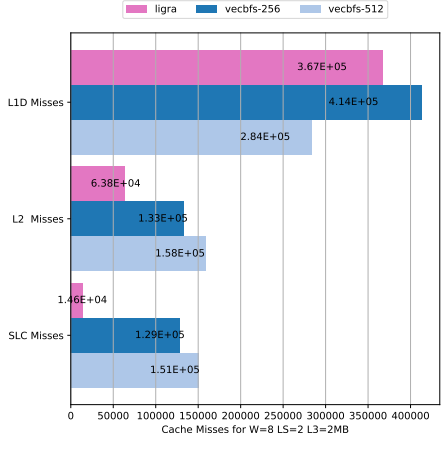


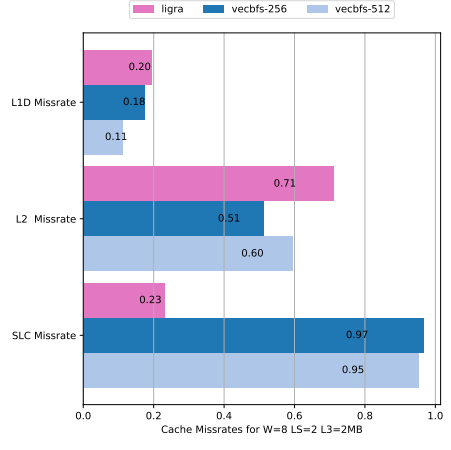


JÜLICH Forschungszentrum

CACHE UTILIZATION

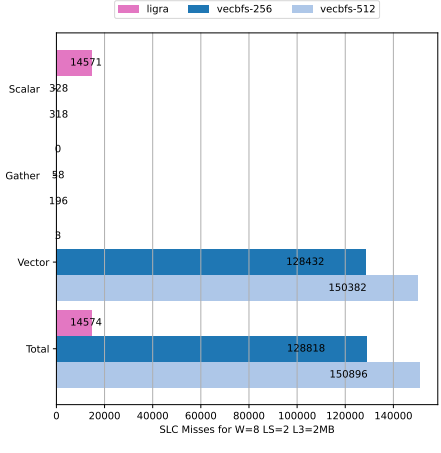
Large Problem Size S=15 (4MB)

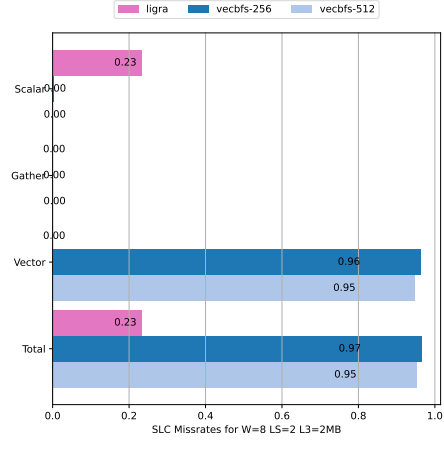




SLC UTILIZATION

Large Problem Size S=15 (4MB)

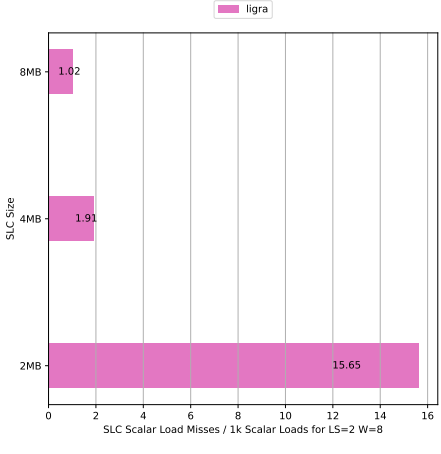


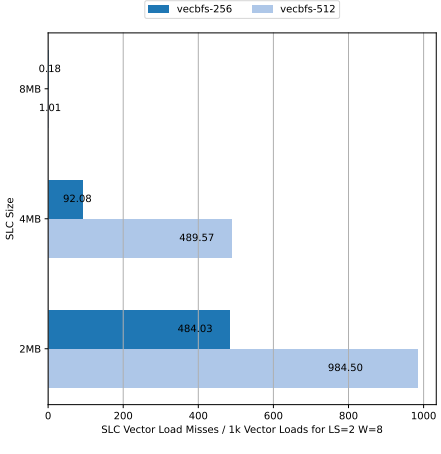


JÜLICH Forschungszentrum

SLC UTILIZATION

Large Problem Size S=15 (4MB)

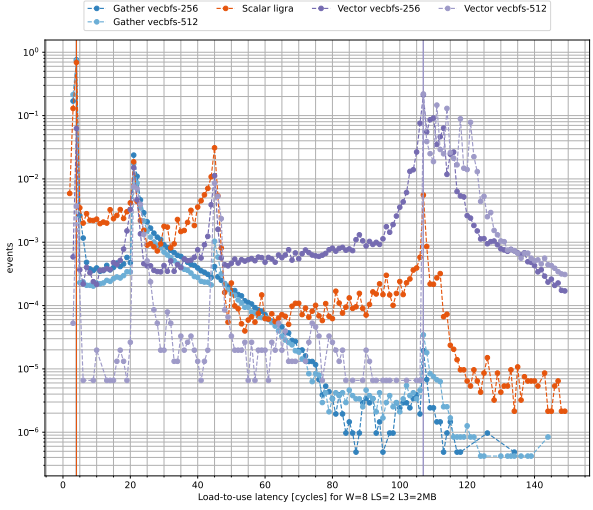




JÜLICH Forschungszentrum

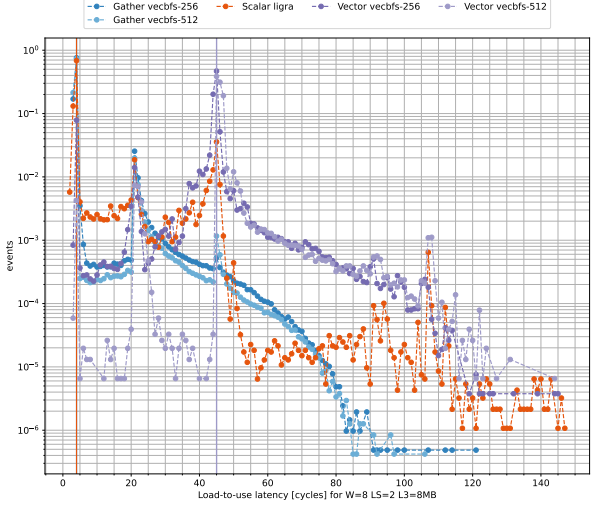
LOAD-TO-USE LATENCIES

Large Problem Size S=15 (4MB) and SLC=2MB



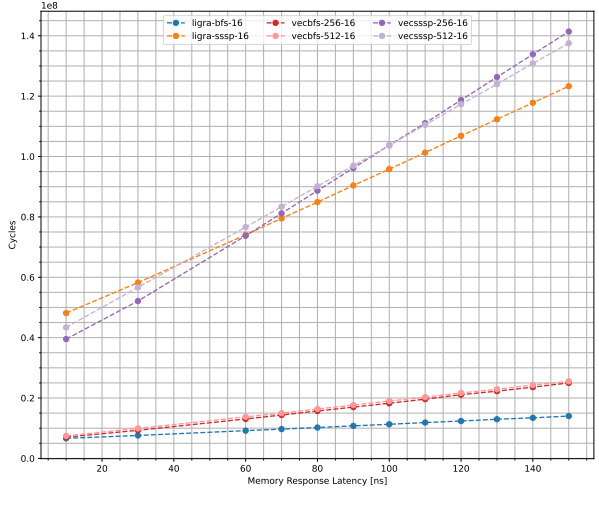
LOAD-TO-USE LATENCIES

Large Problem Size S=15 (4MB) and SLC=8MB



MEMORY LATENCY

Memory Latency sensitivity for S=16



JÜLICH Forschungszentrum

SUMMARY AND CONCLUSIONS

Summary

- Vectorized implementation benefits from high parallelism of gather load micro instructions, as long as the vector loads are utilizing the caches.
- But issue rate is still limited by available LS units, so larger vector lengths will need greater number of cycles to retire vector instructions which depend on the gather micro ops.
- Once the problem size is large enough, SLC misses from vector loads impose large latency penalties.
- Ligra's edge traversal shows low level of instruction parallelism, but much better cache utilization.

SUMMARY AND CONCLUSIONS

Conclusions

- gem5 can be a useful tool not only for architecture research but for software development.
- Need a reference architecture and benchmark to compare performance impact.
- By varying the reference architecture, test how performance changes between benchmarks.
- Don’t optimize for gem5! But use it to gain insight on performance differences between reference and target workloads.

Thank You!

REFERENCES

[1] Jeremy Kepner and John Gilbert, eds. Graph Algorithms in the Language of Linear Algebra. Society for Industrial and Applied Mathematics, 2011. DOI: 10.1137/1.9780898719918.

[2] Jeremy Kepner et al. “Mathematical foundations of the GraphBLAS”. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). 2016, pp. 1–9. DOI: 10.1109/HPEC.2016.7761646.

[3] Moritz Kreutzer et al. “A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units”. In: SIAM Journal on Scientific Computing 36.5 (2014), pp. C401–C423. DOI: 10.1137/130930352.

[4] Julian Shun and Guy E. Blelloch. “Ligra: A Lightweight Graph Processing Framework for Shared Memory”. In: Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2013, pp. 135–146. DOI: 10.1145/2442516.2442530.

High Frequency Performance Monitoring via Architectural Event Measurement
--- on ARM Processors

CHEN LIU

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CLARKSON UNIVERSITY

POTSDAM, NEW YORK, USA

CLIU@CLARKSON.EDU

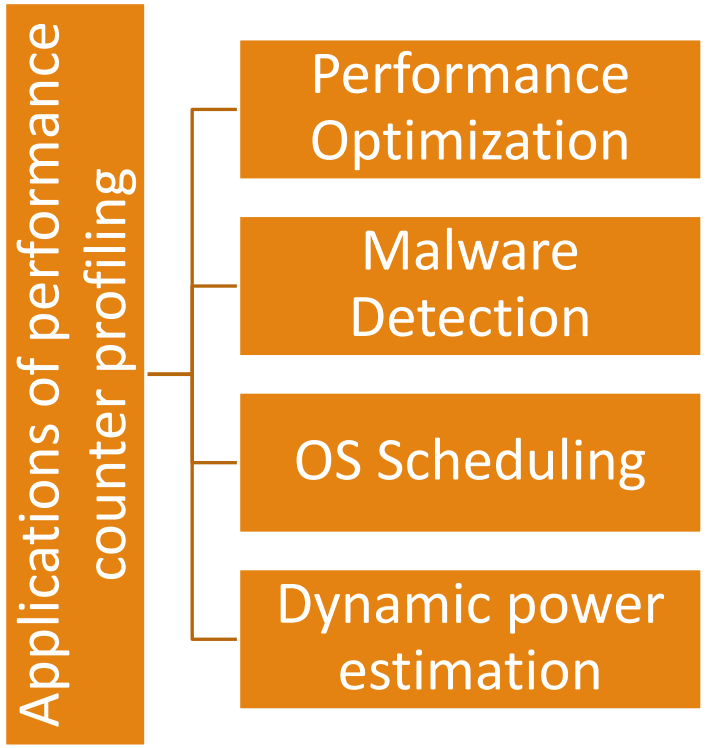
Clarkson™

May 25, 2023 @ AHUG 2023

Motivation

- Significant demand to improve various metrics for modern computing systems
  - performance, energy efficiency, etc.

- Performance monitoring counters (PMCs) built into modern processors
  - Can provide fine-grain performance details
  - ARM Neoverse processor: 1 fixed + 6 programmable performance counters



Motivation

- Many tools have been developed to provide a high-level API to access the low-level performance counters.

Limitations

Kernel - Lineage of Event Behavior (K-LEB)

A performance-counter-based profiling tool that utilizes a kernel space collection system to produce ***precise, non-intrusive, low overhead, high periodicity*** performance counter data.

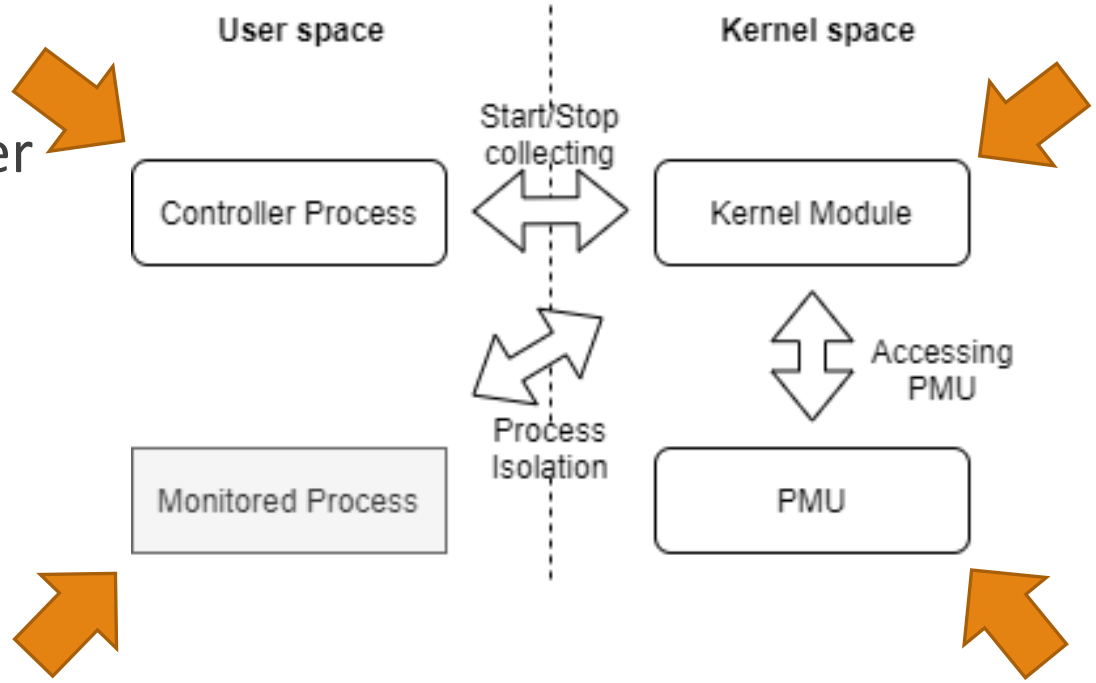
K-LEB System Model

Controller process
- Control the kernel module from user space.

Kernel module
- Access PMU to collect performance counter data.

PMU
- Special hardware unit used to monitor the hardware events.

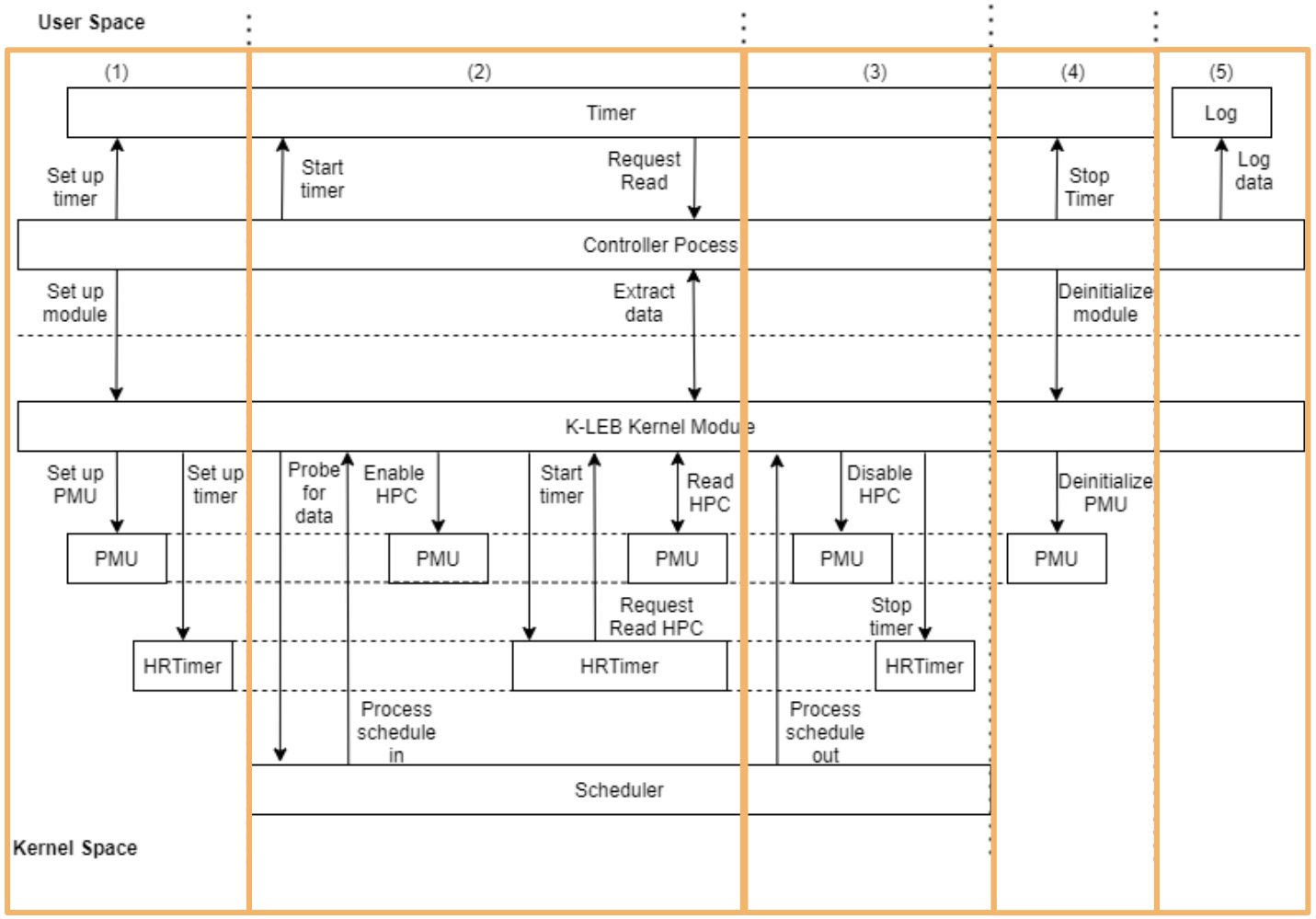
Monitored process
- Process being monitored.



Process Flows

5 phases

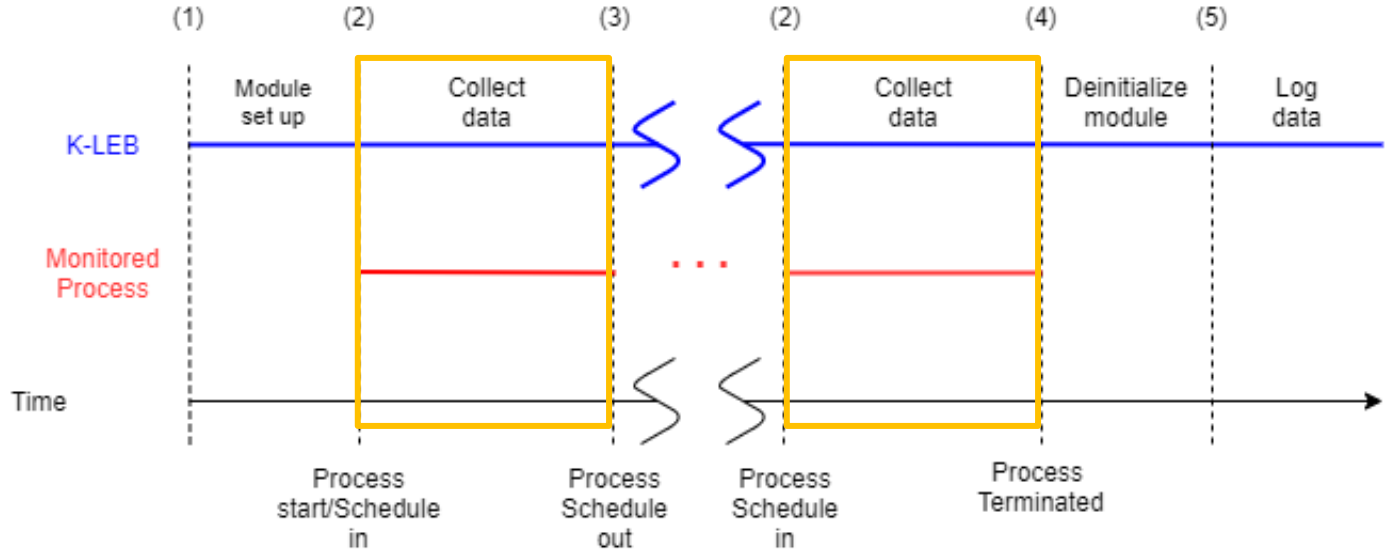
1) Module initialization
2) Start monitoring
3) Stop monitoring
4) Module de-initialization
5) Logging



Process Interaction

Interaction between K-LEB and the monitored process.

- 5 phases
  1) Module initialization
  2) Start monitoring
  3) Stop monitoring
  4) Module de-initialization
  5) Logging



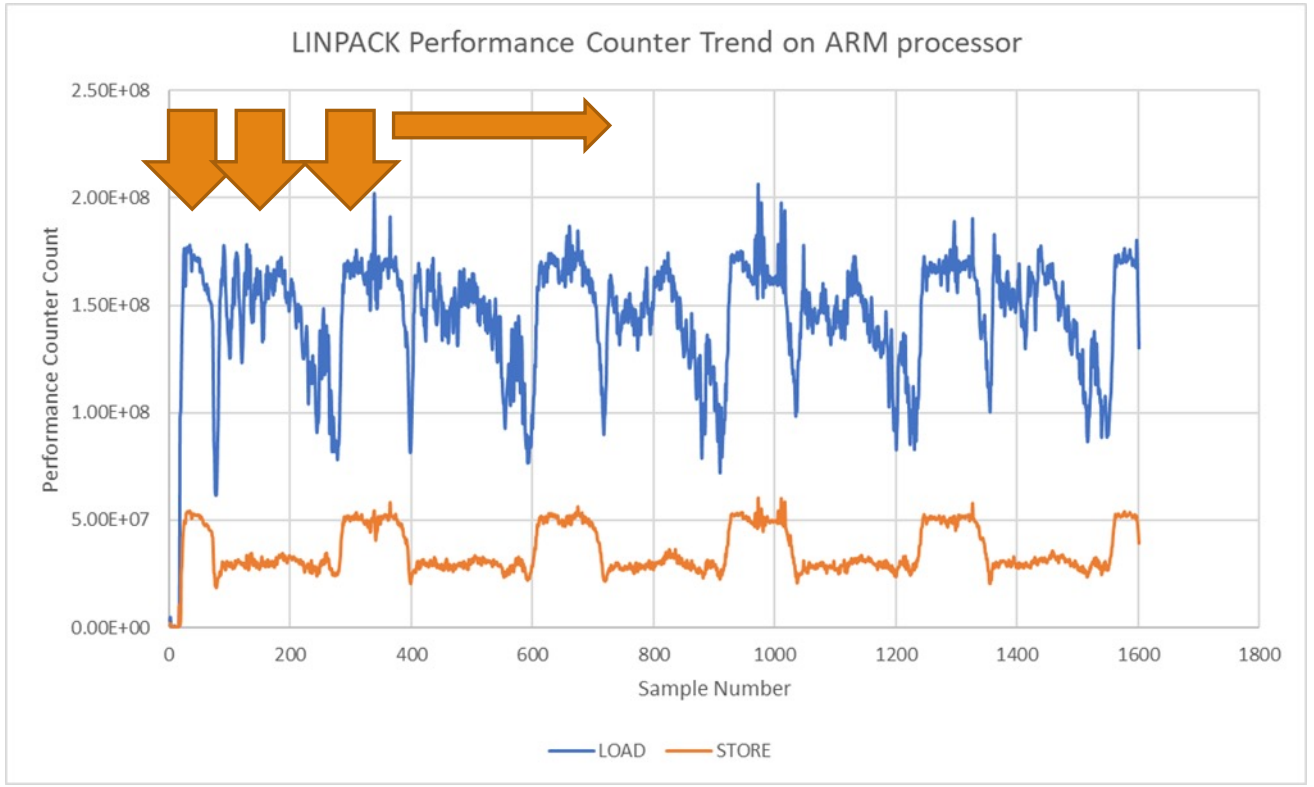
Experiment setup

AWS Graviton Processors feature 64-bit Arm Neoverse running Ubuntu 20.04.2 LTS with Linux kernel version 5.4.0-1041-aws

Raspberry Pi 4 Model B Broadcom BCM2711 @ 1.5GHz processor running Debian 10 with Linux kernel version 5.4.72-v8+

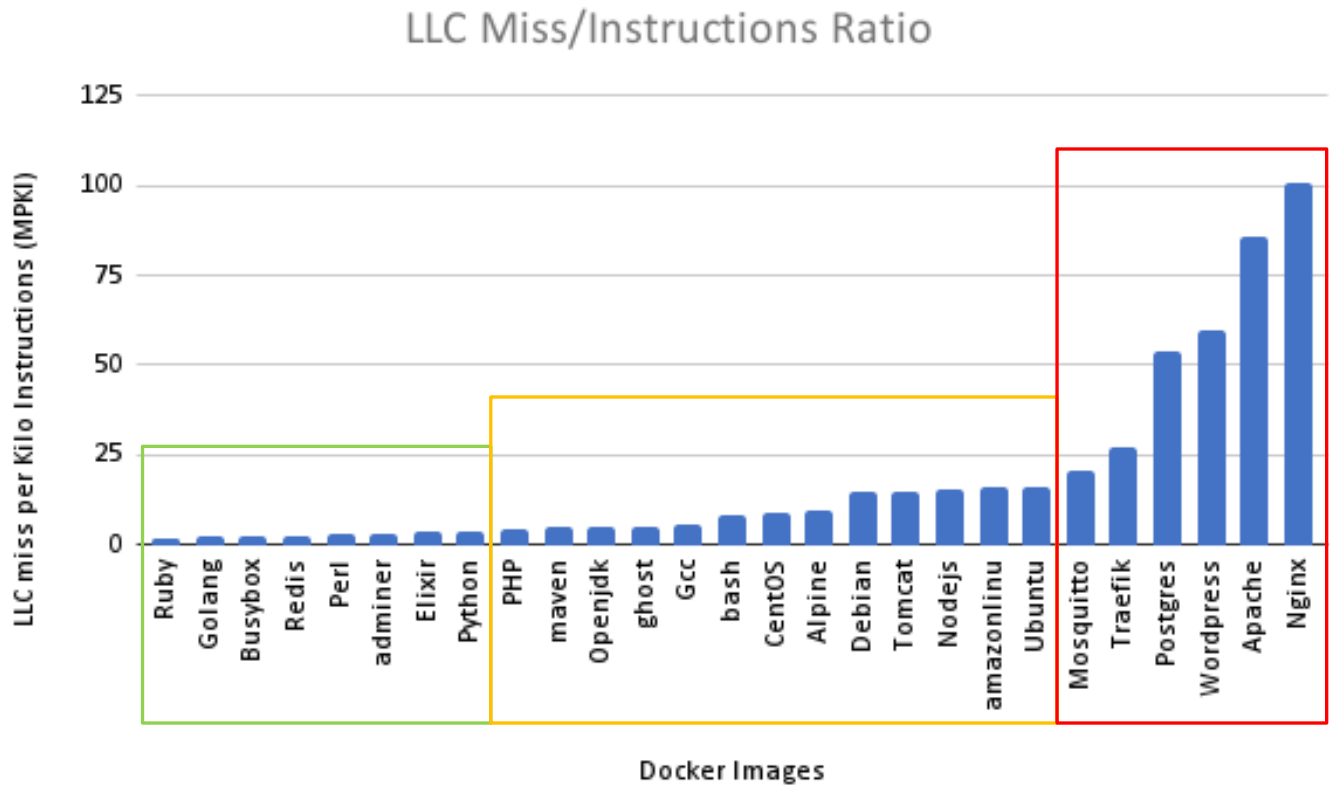
Case study 1: Linpack

- Capture phase behavior.
- K-LEB has a very small FLOPS loss of 0.04% in comparison with 0.98% from Perf.
- No source code require.



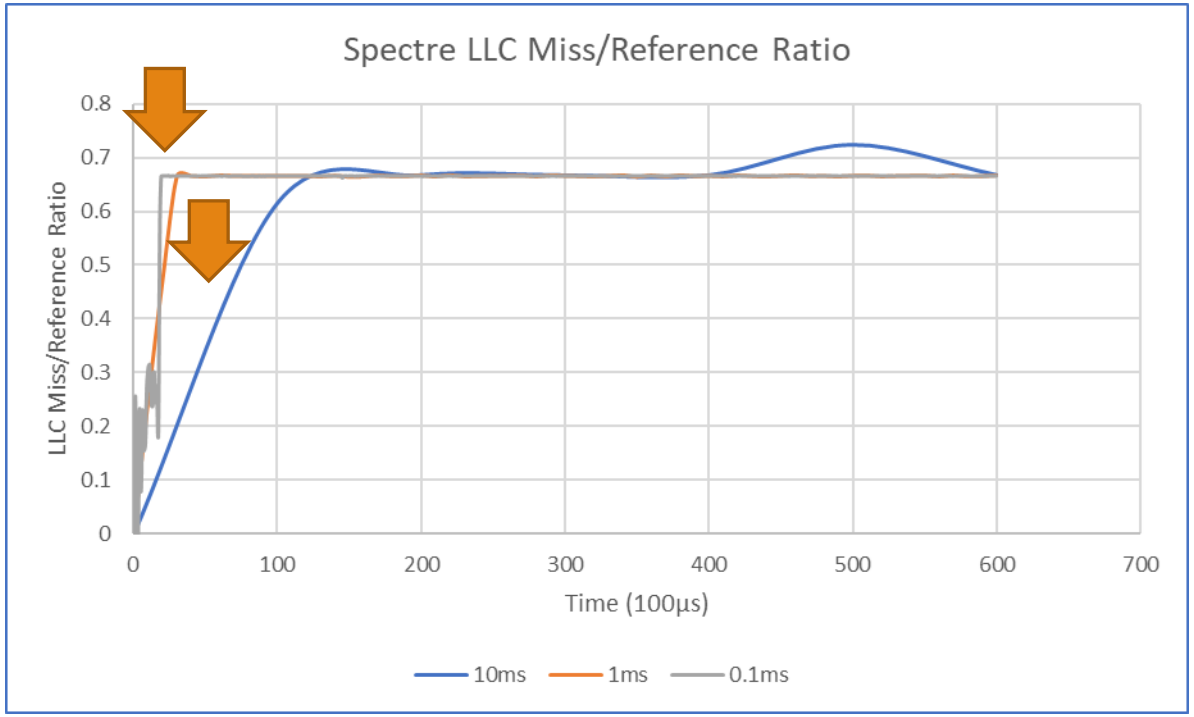
Case study 2: Docker

- Workload characterization.
- Computation/Memory intensive.
- Non-intrusive to a running program.



Case study 3: Spectre Attack

- Anomaly detection.
- High frequency timer.
- Monitor program with short execution time.



Performance overhead comparison

- Test on matrix multiplication program.
- Percentage performance overhead for each profiling tool.

Sample Rate	K-LEB	Perf stat	Perf record
10 ms	0.16	0.41	0.31
1 ms	0.43	N/A	1.52
0.1 ms	2.82	N/A	10.08

Performance overhead comparison

- Test on Linpack benchmark.
- Percentage GFlops loss for each profiling tools.

Sample Rate	K-LEB	Perf stat	Perf record
10 ms	0.04	0.98	0.47
1 ms	0.31	N/A	2.97
0.1 ms	2.18	N/A	15.378

Conclusions

We introduce K-LEB, a kernel module-based approach for performance counter collection with following features:

- Non-intrusive to the program being monitored.
- Provide high frequency sampling rate up to 100µs, which is 100 times finer granularity than current available tools.
- Very low overhead.
- Can benefit program analysis, malware detection and scheduling techniques.

Performance Evaluation of the Ginkgo Sparse Linear Solver Framework on Arm

May. 25th, 2023 – AHUG workshop at ISC conference, Hamburg, Germany

Luka STANISIC, Robert MIJAKOVIC, Matthias GRIES

HUAWEI

Scope

Huawei's investment in HPC

- Many products: storage (main focus of ISC 2023 booth E518), cloud, traditional HPC servers, AI accelerators, software, etc.
- Our team concentrates on Arm software ecosystem enablement for HPC & AI (Munich research center, funded by HiSilicon)
- Recognized current and future importance of sparse linear algebra for performance of top HPC applications

Ginkgo sparse linear solver framework (https://ginkgo-project.github.io/)

- Offering set of iterative solvers and preconditioners under BSD 3-clause permissive open-source license
- Comes with built-in benchmarks for spmv, matrix conversion and solver / preconditioner
- Recent, fresh approach, started in 2017 – already integrated with MFEM, deal.II, xSDK
- Main developers and maintainers located in Germany at Karlsruhe Institute of Technology (KIT)
- Most programming efforts spent on kernels for Nvidia, AMD and Intel accelerators
- Very competitive performance

Focus of the current research project

- Improving the OpenMP CPU execution target for Huawei Kunpeng 920 AArch64 chipset
- Relying on GNU GCC compilation
- First step: performance characterization of Ginkgo sparse linear solver on Arm & x86

Experimental Setup

Evaluated several thousand unique experimental setups

- Ginkgo options: 3 benchmarks (spmv, conversions, solver), 9 preconditioners (jacobi, paric, parict, parilu, parilut, paric-isai, parict-isai, parilu-isai, parilut-isai, or none), 5 sparse matrix formats (csr, coo, ell, hybrid, sellp), etc.
- Huawei Kunpeng 920 system options: GCC compilers (versions 10, 11, 12), OpenMP threads scaling (24, 48, 96), etc.
- Additional investigations (in backup slides): active Working Set Size (WSS), memory frequency scaling (1600/2133/2933 MHz), hardware prefetching (on/off), benchmark precision, I/O, C++ overhead, etc.

Evaluation with 10 real-world examples from SuiteSparse collection

- Ranked and selected matrices by studying ~200 sources (publications, whitepapers, products, frameworks) from industry and academia

Machines

- 96-cores 2x Huawei Kunpeng 920-4826 (launched in January 2019)
  - 128-cores version of Kunpeng 920 accessible via OEHI
- 64-cores Amazon Graviton 3 on AWS EC2 Ireland (remote access)
- 48-cores 2x Intel 3rd gen Xeon Scalable Processors Gold 6342 (IceLake)
- 48-cores 2x AMD 3rd gen EPYC 7413 (Milan)

Selected use cases from SuiteSparse matrix collection

Name	Group	Rows	Cols	Non-zeros	Sparsity	Kind	Date	Ref. count
thermal2	Schmid	1,228,045	1,228,045	8,580,313	0.0006%	Thermal Problem	2006	9
Serena	Janna	1,391,349	1,391,349	64,131,971	0.0033%	Structural Problem	2011	9
bone010	Oberwolfach	986,703	986,703	47,851,783	0.0049%	Model Reduction Problem	2006	7
atmosmodd	Bourchtein	1,270,432	1,270,432	8,814,880	0.0005%	Computational Fluid Dynamics	2009	5
offshore	Um	259,789	259,789	4,242,673	0.0063%	Electromagnetics Problem	2010	5
G3_circuit	AMD	1,585,478	1,585,478	7,660,826	0.0003%	Circuit Simulation	2006	5
consph	Williams	83,334	83,334	6,010,480	0.0865%	2D/3D Problem	2008	5
pdb1HYS	Williams	36,417	36,417	4,344,765	0.3276%	Undirected Graph	2008	5
Hardesty3	Hardesty	8,217,820	7,591,564	40,451,632	0.0001%	Computer Graphics/Vision	2015	0
analytics	Precima	303,813	303,813	2,006,126	0.0022%	Data Analytics Problem	2018	0

http://sparse.tamu.edu/

Overview

- **Ginkgo performance analysis**
  - Roofline
  - Instruction mix and top-down
  - Hot kernels
  - OpenMP imbalance
  - Scalability
  - Overall performance

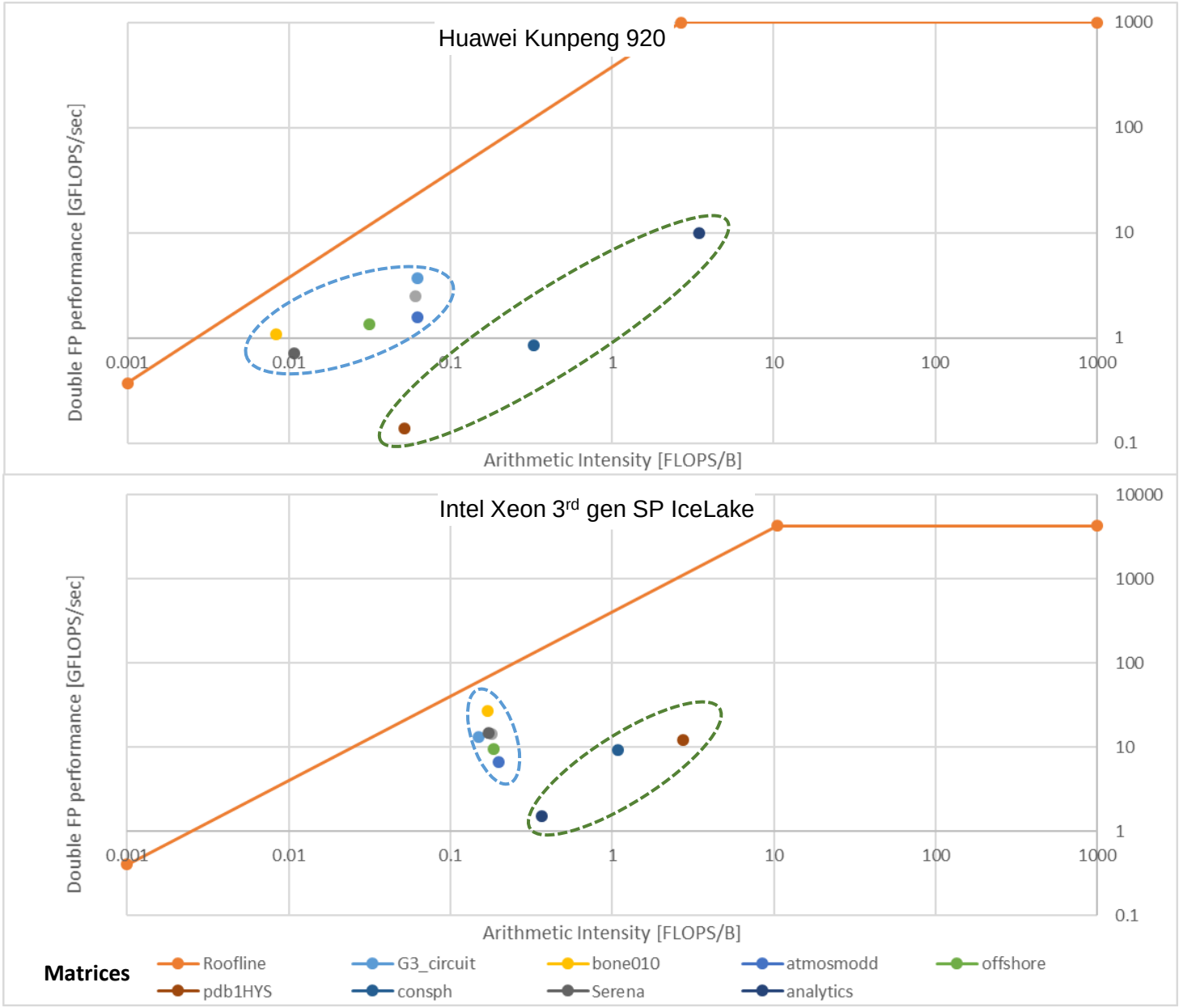
- **Concluding Remarks**
  - Pitfalls and next steps

Roofline Analysis

- Best preconditioner + solver combination per matrix applied, at double-precision

- In memory-bound region of the roof, with large dynamic range for arithmetic intensity
- Significant distance to the roof due to control flow, partial cacheline use, non-consecutive memory accesses, lack of SIMD use, etc.

- FLOPS efficiency thus below 1% (similar to, e.g., HPCG)
- Behavior depending on number of non-zeros (NNZ) and sparsity patterns of matrices



Instruction Mix and Top-down Analysis

Statistics for complete preconditioner + solver runs on Kunpeng 920 summarized for all matrices

- Low fraction of floating point and SIMD instructions
- High amount of integer operations
- Mainly limited by memory backend (LLC and main memory)
- Limitation by LLC or main memory strongly depends on matrix

Additional experiments across Arm & x86 machines confirmed minimal performance impact of advanced SIMD support coming from GCC 12 auto-vectorization

To be further analyzed and tuned

	Sub-category	AVERAGE	MIN	MAX
Instruction Mix	Memory (%)	21.84	16.72	24.99
	Integer (%)	48.38	45.25	53.27
	Floating Point (%)	2.65	0.42	4.33
	Advanced SIMD (%)	1.27	0.04	6.06
	Not Retired (%)	2.22	0.36	4.75
Top-down	Retiring (%)	30.83	16.46	49.60
	Backend Bound (%)	62.98	39.48	79.62
	-> Memory Bound (%)	40.46	15.66	62.05
	--> L1 Bound (%)	6.55	3.13	10.10
	--> L2 Bound (%)	0.72	0.16	1.56
	--> L3 or DRAM Bound (%)	33.13	9.68	54.94
	--> Store Bound (%)	0.06	0.02	0.15
	-> Core Bound (%)	22.51	17.57	27.41
	Frontend Bound (%)	5.41	2.50	9.32
	Bad Speculation (%)	0.79	0.09	1.99
Memory subsystem	Average DRAM Bandwidth (GB/s)	34.08	2.76	68.54
	-> Read (GB/s)	28.82	2.32	65.03
	-> Write (GB/s)	5.27	0.44	13.05
	L3 By-Pass ratio (%)	8.39	0.83	22.19
	L3 miss ratio (%)	62.13	24.29	82.98
	L3 Utilization Efficiency (%)	79.51	36.37	96.85
	Within Socket Bandwidth (GB/s)	1.07	0.25	3.35
	Inter Socket Bandwidth (GB/s)	1.79	0.33	5.73

Hot Compute Kernels on Kunpeng 920

Example: spmv kernel for csr format

- Iterating over rows
- Non-consecutive column accesses
- Multiply-add operation

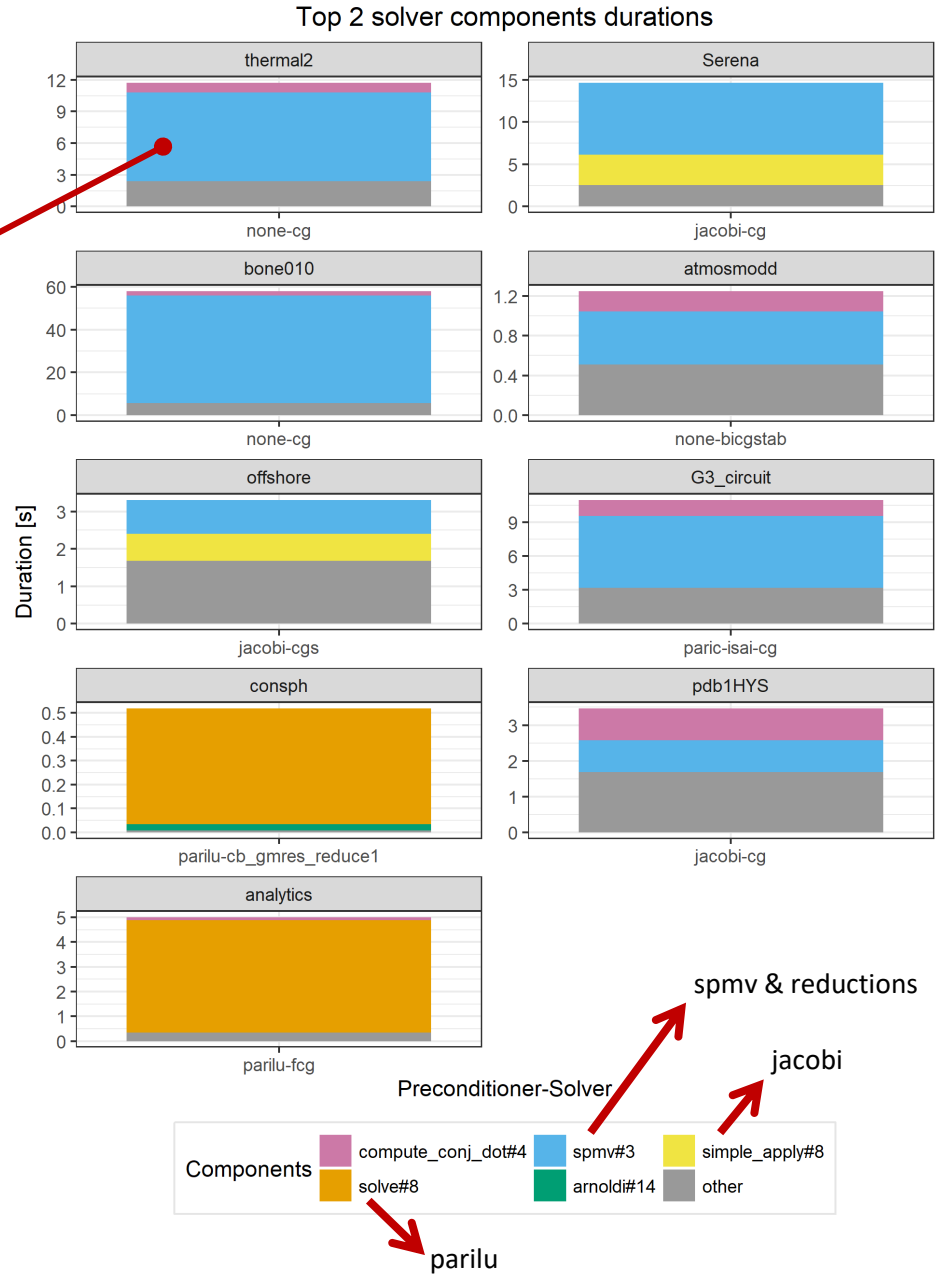
```
#pragma omp parallel for
    for (size_type row = 0; row < a->get_size()[0]; ++row) {
        for (size_type j = 0; j < c->get_size()[1]; ++j) {
            c->at(row, j) = zero<ValueType>();
        }
        for (size_type k = row_ptrs[row];
             k < static_cast<size_type>(row_ptrs[row + 1]); ++k) {
            auto val = vals[k];
            auto col = col_idxs[k];
            for (size_type j = 0; j < c->get_size()[1]; ++j) {
                c->at(row, j) += val * b->at(col, j);
            }
        }
    }
```

Limitations

- Cannot take advantage of SIMD for multiply-add
- Memory bound because of non-consecutive accesses

Further hotspots identified

- Column reduction operations
- Jacobi preconditioner
- ParILU (Parallel Incomplete LU) preconditioner



OpenMP Imbalance Examples: Kunpeng 920, Preconditioner + Solver

- Considerable parallel time, but with noticeable imbalance between 96 OpenMP threads

Matrix	Total execution (s)	Serial time (s)	Serial time (%)	Parallel time (s)	Parallel time (%)	Imbalance (s)	Imbalance (%)
bone010	27.98	6.32	22.61%	21.65	77.39%	9.08	41.96%
Top parallel region	Elapsed time (s)	Perc. of total (%)	Perc. of parallel (%)	Average (ms)	Count	Imbalance (s)	Imbalance (%)
csr_kernels.cpp:84	8.35	29.86%	38.58%	0.82	10200	1.90	22.78%
kernel_launch_reduction.hpp:335	2.42	8.66%	11.19%	0.12	20403	1.34	55.13%
kernel_launch.hpp:83	2.27	8.11%	10.48%	0.22	10200	0.66	29.18%
kernel_launch_reduction.hpp:353	1.64	5.85%	7.56%	0.08	20403	1.33	81.45%

- High serial time, but low imbalance in parallel regions

Matrix	Total execution (s)	Serial time (s)	Serial time (%)	Parallel time (s)	Parallel time (%)	Imbalance (s)	Imbalance (%)
Serena	42.54	26.10	61.35%	16.44	38.65%	1.64	9.99%
Top parallel region	Elapsed time (s)	Perc. of total (%)	Perc. of parallel (%)	Average (ms)	Count	Imbalance (s)	Imbalance (%)
sellp_kernels.cpp:67	9.63	22.65%	58.60%	5.65	1706	0.8117	8.43%
jacobi_kernels.cpp:564	4.09	9.61%	24.87%	2.39	1709	0.3982	9.74%
kernel_launch_reduction.hpp:335	0.62	1.47%	3.80%	0.18	3415	0.0856	13.71%

- Preconditioner dominating, high imbalance (kernels de facto sequential)

Matrix	Total execution (s)	Serial time (s)	Serial time (%)	Parallel time (s)	Parallel time (%)	Imbalance (s)	Imbalance (%)
consph	23.32	3.26	13.97%	20.06	86.03%	16.80	83.76%
Top parallel region	Elapsed time (s)	Perc. of total (%)	Perc. of parallel (%)	Average (ms)	Count	Imbalance (s)	Imbalance (%)
upper_trs_kernels.cpp:101	9.58	41.07%	47.74%	28.93	331	9.45	98.73%
lower_trs_kernels.cpp:101	6.36	27.27%	31.70%	19.21	331	6.28	98.85%

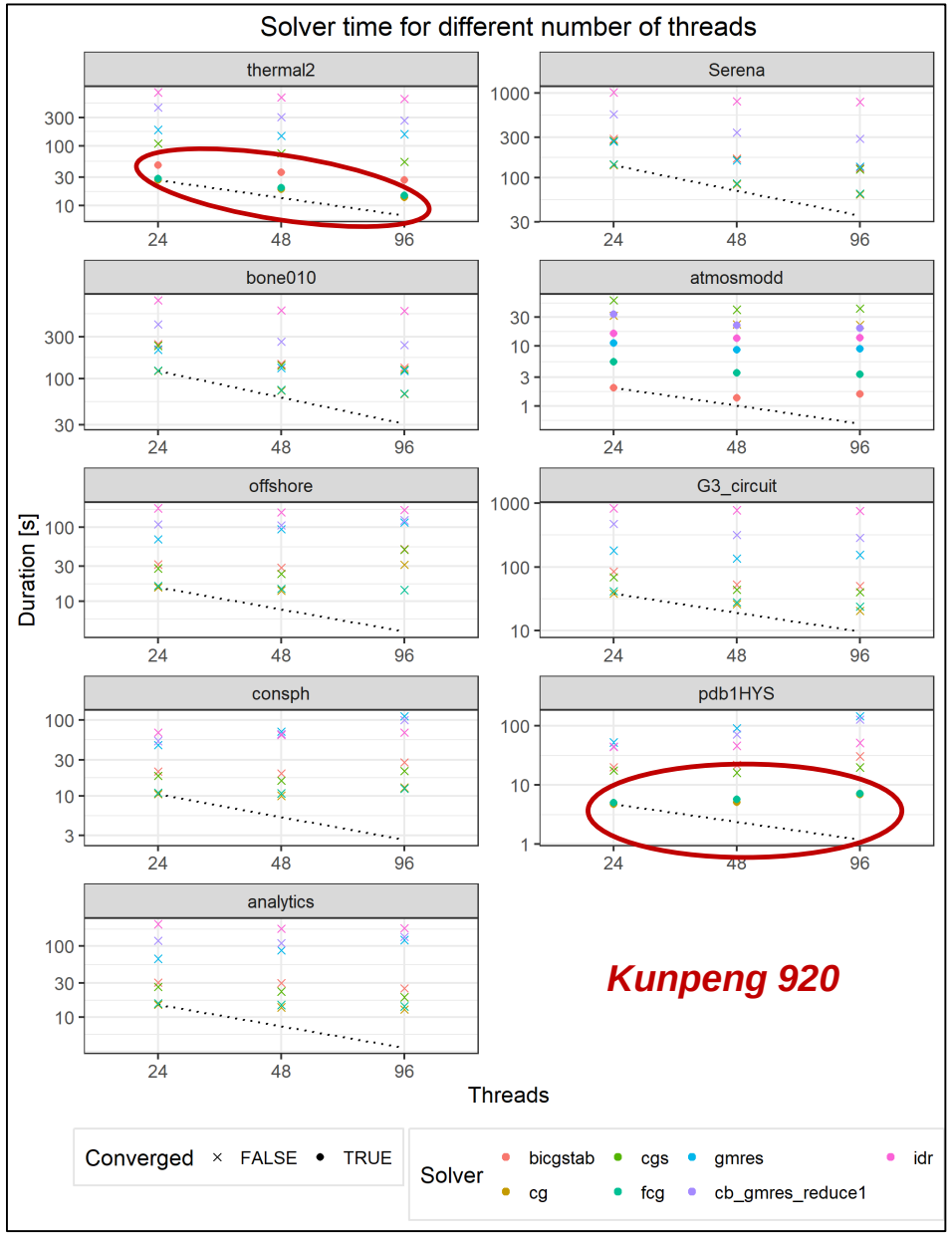
To be further analyzed and tuned

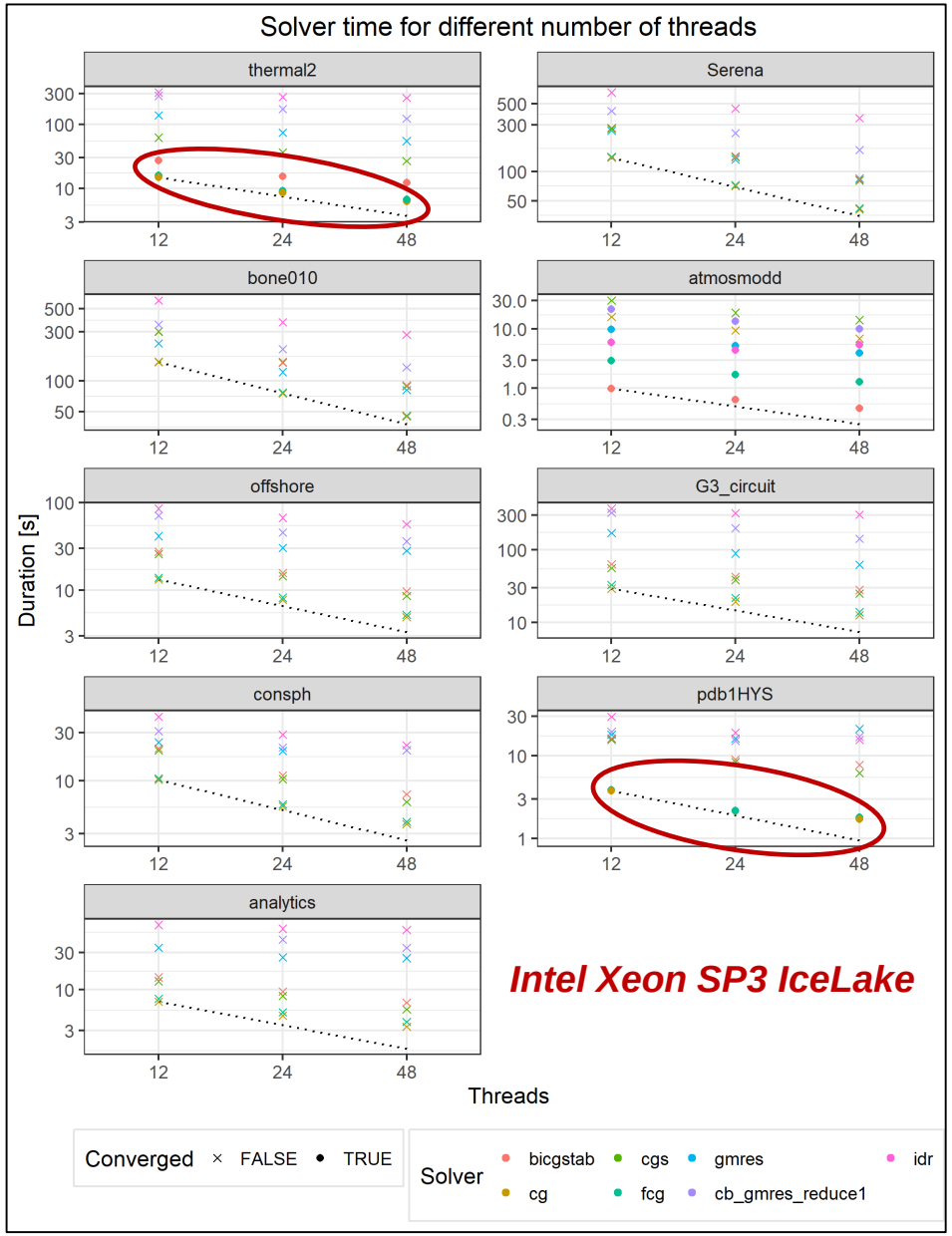
Scalability

- Solver benchmark
- Log vertical axis
- Results again very dependent on matrix

- Kunpeng 920 has 96 cores and often cannot take advantage of complete node

- Mediocre scaling on other machines (48 and 64 cores)
- No benefits from hyperthreading

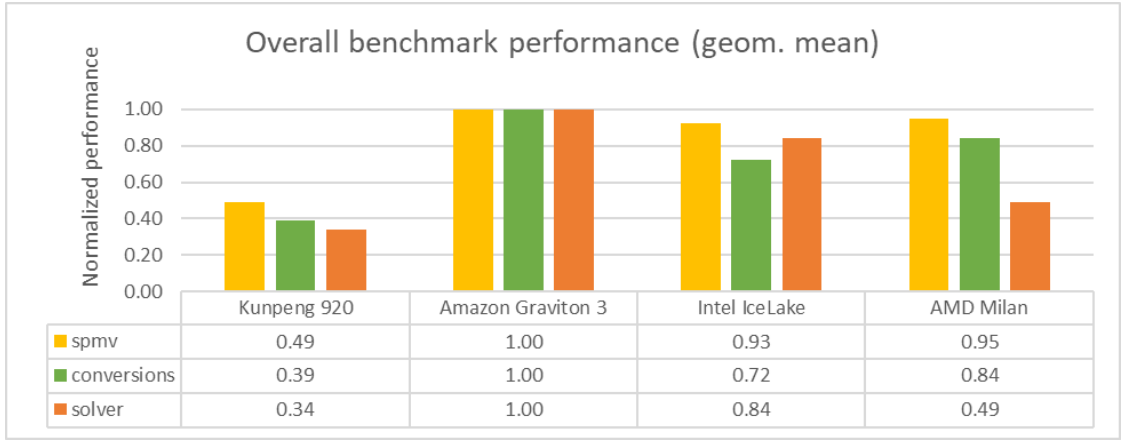




Overall Performance

Geometric mean across matrices, default config ranking:

1. 64-cores Amazon Graviton 3 on AWS EC2 - *2022*
2. 48-cores 2x Intel Xeon SP3 Gold 6342 (IceLake) - *2021*
3. 48-cores 2x AMD 3rd gen EPYC 7413 (Milan) - *2021*
4. 96-cores 2x Huawei Kunpeng 920-4826 - *2019*



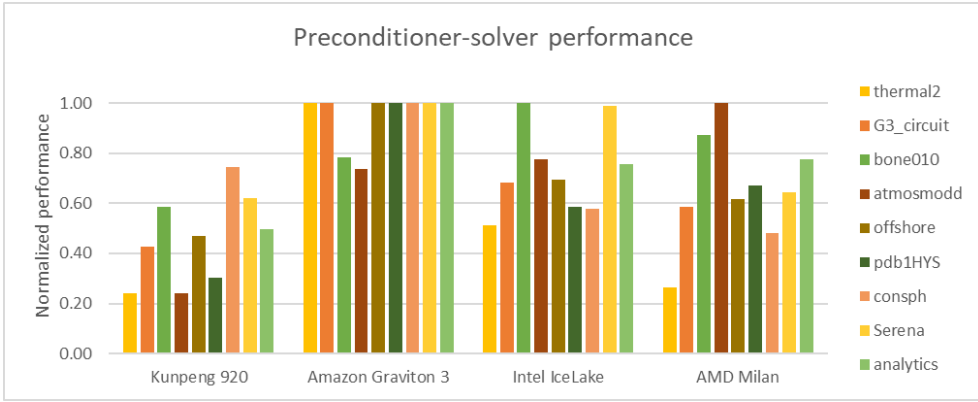
Preconditioner + solver selection

- Simple Jacobi preconditioner often the best, followed by not using a preconditioner at all
- Selected cases where one of the parallel incomplete Cholesky/LU variants allows much faster convergence
- Conjugate Gradient (CG) solver algorithm often the best

Matrix	Kunpeng 920		Amazon Graviton 3		Intel IceLake		AMD Milan	
	Precond.	Solver	Precond.	Solver	Precond.	Solver	Precond.	Solver
thermal2	none	cg	none	cg	none	cg	jacobi	cg
G3_circuit	paric-isai	cg	paric-isai	cg	paric-isai	cg	paric-isai	cg
bone010	none	cg	none	cg	none	cg	none	cg
atmosmodd	none	bicgstab	none	bicgstab	none	bicgstab	none	bicgstab
offshore	jacobi	cgs	parilu-isai	cgs	parilu-isai	cgs	jacobi	cgs
pdb1HYS	jacobi	cg	jacobi	cg	jacobi	cg	jacobi	cg
consph	parilu	cb_gmres	jacobi	cg	jacobi	cgs	jacobi	cg
Serena	jacobi	cg	jacobi	cg	jacobi	cg	jacobi	cg
analytics	parilu	fcg	parilu	gmres	parilu	cg	parilu	cb_gmres_re

Preconditioner + solver performance per matrix

- Graviton 3 mostly performs the best (7 out of 9 matrices)
- Performance on AMD Milan greatly depends on matrix



Overview

- **Ginkgo performance analysis**
  - Roofline
  - Instruction mix and top-down
  - Hot kernels
  - OpenMP imbalance
  - Scalability
  - Overall performance

- **Concluding Remarks**
  - Pitfalls and next steps

Concluding Remarks

Pitfalls

- Variability of repeated runs with the same settings can be significant
  - Mixture of OpenMP overhead, NUMA effects, startup calibrations by Ginkgo for selecting matrix format, nondeterministic preconditioner algorithms, responsiveness of memory subsystem under load

- Extra investigation of OpenMP parallelism
  - On Kunpeng 920, when a subset of cores is used: the best choice between “spread” and “close” mappings is matrix dependent, and can have a significant impact on the overall performance
  - Using dynamic and guided OpenMP scheduling strategies decreases performance, especially for smaller chunks

Next steps

- Performance analysis of direct sparse linear solvers (e.g., MUMPS framework)
- Ginkgo contributions related to micro-architecture level optimization (e.g., SVE) for selected algorithms

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.

References and Setups

References

- H. Anzt, T. Cojean, G. Flegar et al.: *Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing*, ACM Transactions on Mathematical Software, volume 48(1), 2022
  https://ginkgo-project.github.io/
- T.A. Davis, Yifan Hu: *The university of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, Volume 38(1), 2011
  SuiteSparse matrix collection: http://sparse.tamu.edu/
- Jing Xia, Chuanning Cheng, Xiping Zhou et al.: *Kunpeng 920: The First 7-nm Chiplet-Based 64-Core ARM SoC for Cloud Services*, IEEE Micro, volume 41(5), 2021
- Brendan Gregg, *How To Measure the Working Set Size on Linux*, https://www.brendangregg.com/blog/2018-01-17/measure-working-set-size.html (accessed May 2023)

Machine setups

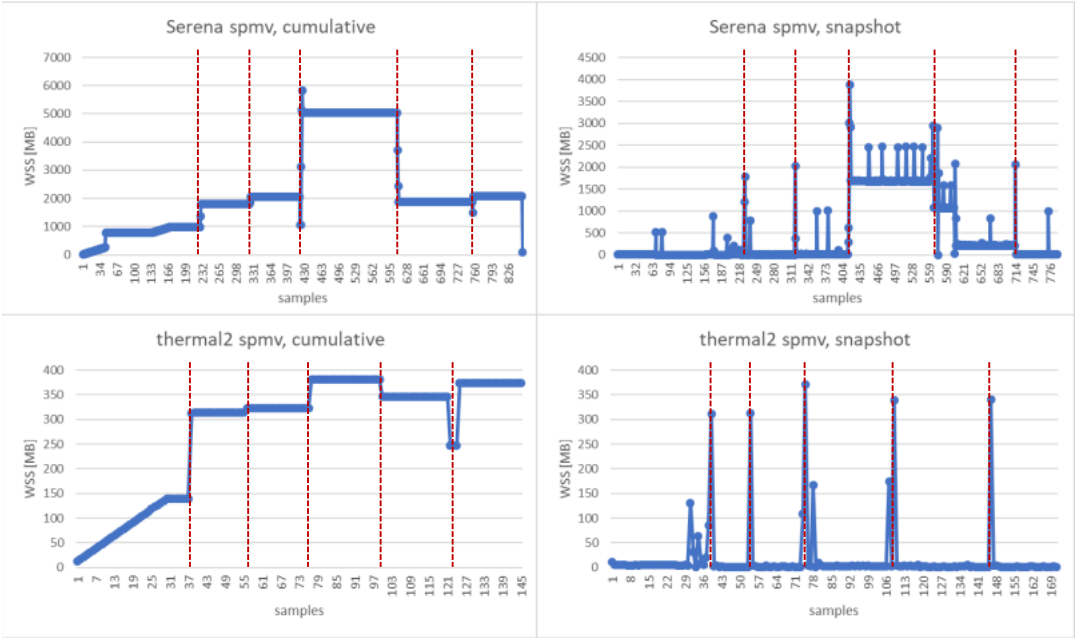
- 2x Kunpeng 920-4826 (48 cores) at 2.6 GHz, 48 MB LLC, 128 bit NEON vectorization support (one FMA unit enabled for double precision, two FMA units enabled for single precision), 16 x 32 GB DDR4-2933 memory DIMMs (single DIMM per channel), Ubuntu 18.04.6 LTS (Linux 5.4.0-136-generic aarch64)
- 2x AMD EPYC 7413 (24 cores) at 2.65 GHz base frequency, 128 MB LLC, AVX2 vectorization support (two FMA units), 16 x 16 GB DDR4-3200 memory DIMMs (single DIMM per channel), Ubuntu 18.04.6 LTS (Linux 5.4.0-90-generic x86_64)
- 2x Intel Xeon SP3 Gold 6342 (24 cores) at 2.8 GHz base frequency, 36 MB LLC, AVX512 vectorization support (two FMA units), 16 x 16 GB DDR4-3200 memory DIMMs (single DIMM per channel), Ubuntu 18.04.6 LTS (Linux 5.15.31-051531-generic x86_64)
- AWS EC2 c7g.16xlarge instance: 1x Amazon Graviton 3 (64 cores) at likely 2.6 GHz, likely 64 MB LLC, SVE vectorization support (two FMA units enabled for SVE256, 4 FMA units enabled for NEON), likely 8 channels at DDR5-4800 (128 GB total), Ubuntu 22.04.1 LTS (Linux 5.15.0-1027-aws aarch64)

Active Working Set Size (WSS)

- WSS depends on matrix characteristics
- 3 matrices need several GBs (served from main memory)
- 3 matrices take up to 0.5 GB
- 4 matrices take up to 0.25 GB and can leverage the LLC

Working set size [MB]	spmv	conversion	Best preconditioner + solver
thermal2	380.4	451.3	202.2
G3_circuit	359.6	398.1	279.7
bone010	2251.6	3143.4	3034.7
atmosmodd	354.2	415.4	291.5
offshore	198.5	244.0	237.5
pdb1HYS	188.5	249.8	117.4
consph	239.5	283.6	238.9
Serena	5839.3	5965.3	2637.1
Hardesty3	1717.1	1881.9	--
analytics	179.2	89348.9	88.1

- After initialization, WSS cumulative traces of spmv benchmark show 5 distinct phases for 5 evaluated matrix formats
- Corresponding WSS snapshot traces show peak memory requirements at the beginning of each phase



Memory Bandwidth Characteristics

Top-down analysis showed that Ginkgo solver execution is backend bounded mainly by memory

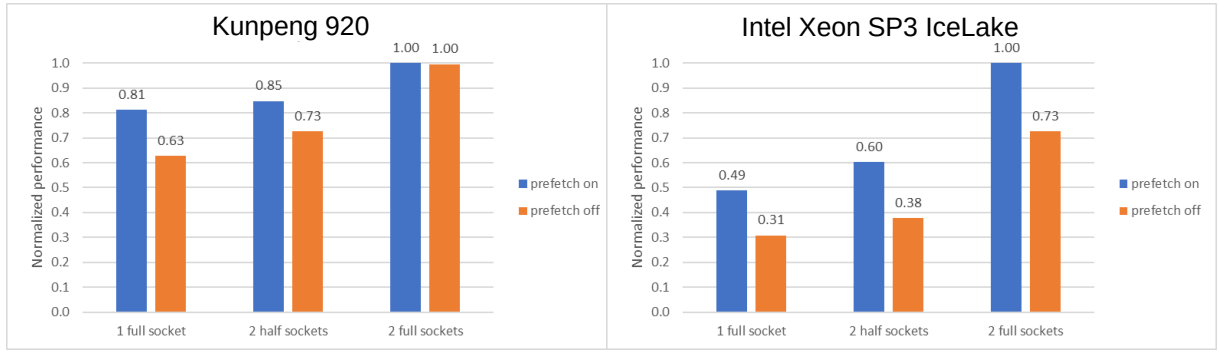
Hardware prefetcher (example: spmv benchmark)

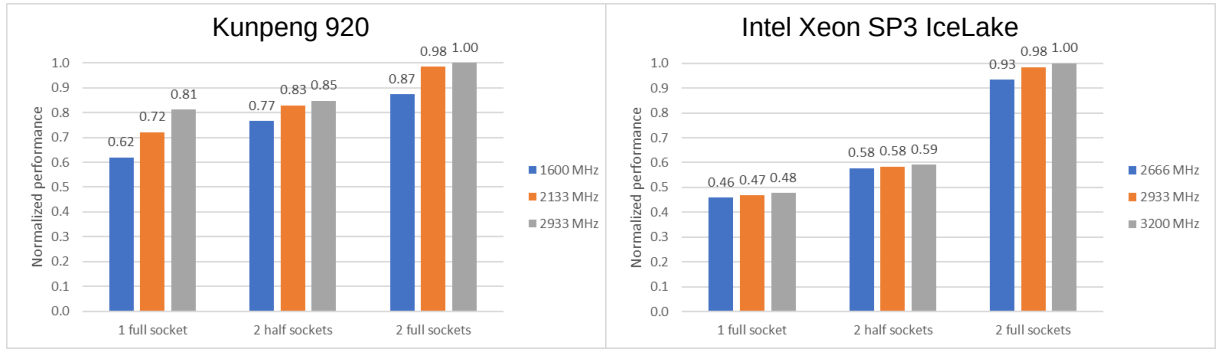
- Intel IceLake's prefetchers show the highest impact, reducing performance up to 38% by switching them off
- Followed by Kunpeng 920 (23%) and AMD Milan (13%)

Impact of DRAM DIMM speed (example: spmv benchmark)

- On Kunpeng 920, performance drops by up to 24% (mem frequency reduced by 45%, DDR4-2933 → 1600)
- On AMD & Intel, performance goes down by 3% and 7% (mem frequency reduced by 17%, DDR4-3200 → 2666)
- Overall, proportional dependency on memory bandwidth (performance partially limited by memory latency as well)

Normalized performance (higher is better)

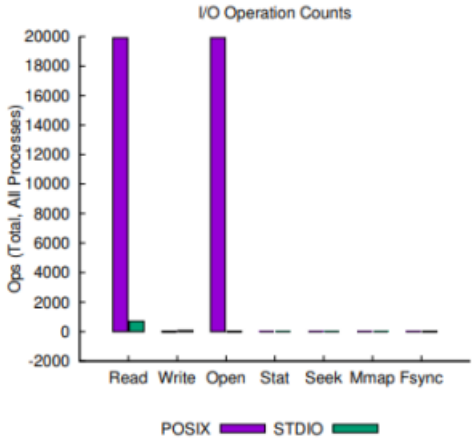




Additional Investigations

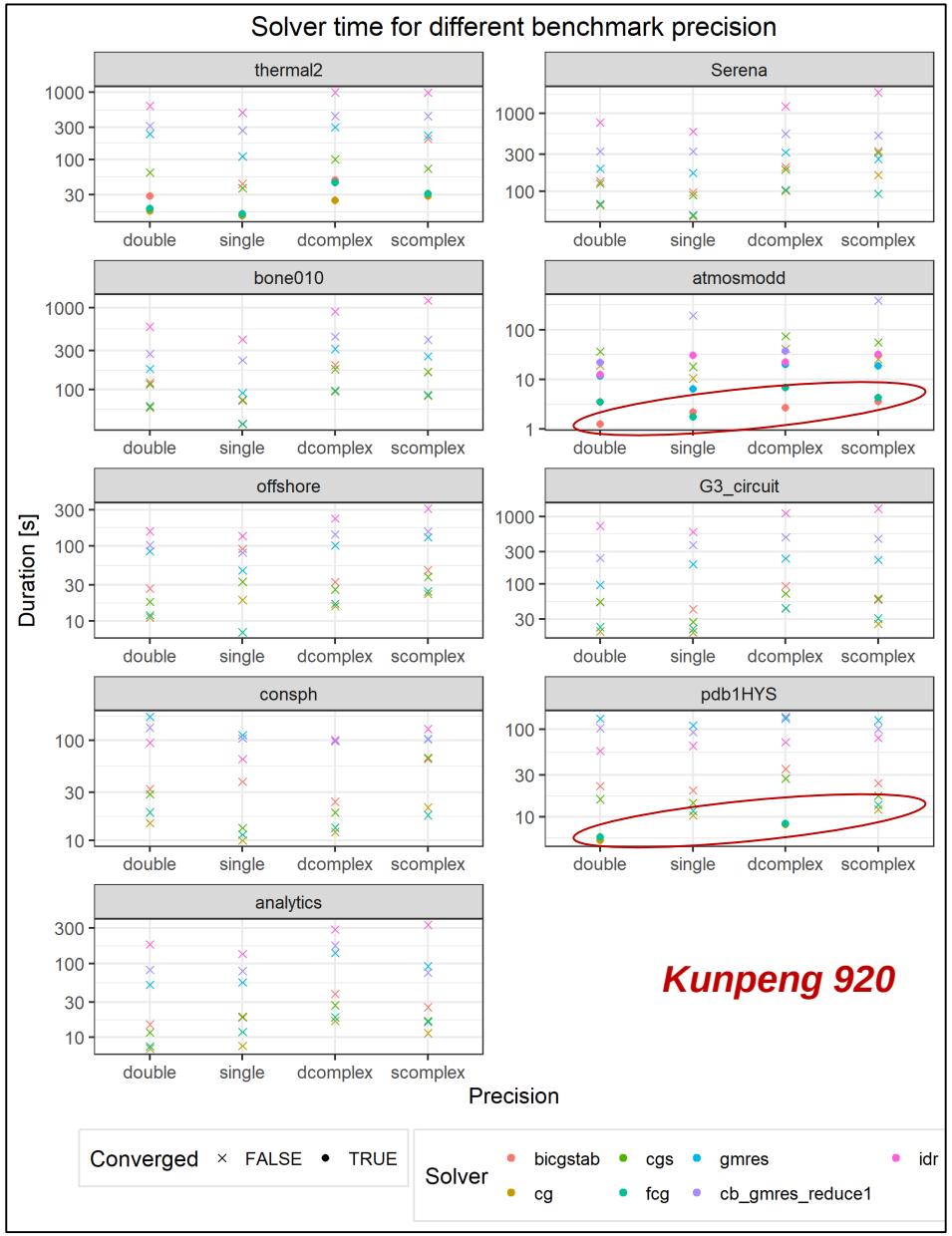
- In most cases lower precision single kernels are faster than double, and scomplex kernels faster than dcomplex
- For atmosmodd and pdb1HYS matrices higher precision is faster than lower due to faster convergence

- Darshan traces confirmed that Ginkgo executions have almost no I/O footprint (except initial reading of the matrix), similar to other sparse linear solvers



- Measured negligible overhead of C++ runtime polymorphism

```
Running 1000000 iterations of the CG solver took a total of 4.56531 seconds.
        Average library overhead:     4565.31 [nanoseconds / iteration]
```



Acknowledgement

Chutitep Woralert, Ph.D. candidate, Clarkson University

James Bruska

Dr. Lok Yan

Dr. Gidlo Torres

C. Woralert, J. Bruska, C. Liu and L. Yan, "High Frequency Performance Monitoring via Architectural Event Measurement," *2020 IEEE International Symposium on Workload Characterization (IISWC)*, Beijing, China, 2020, pp. 114-122, doi: 10.1109/IISWC50251.2020.00020.

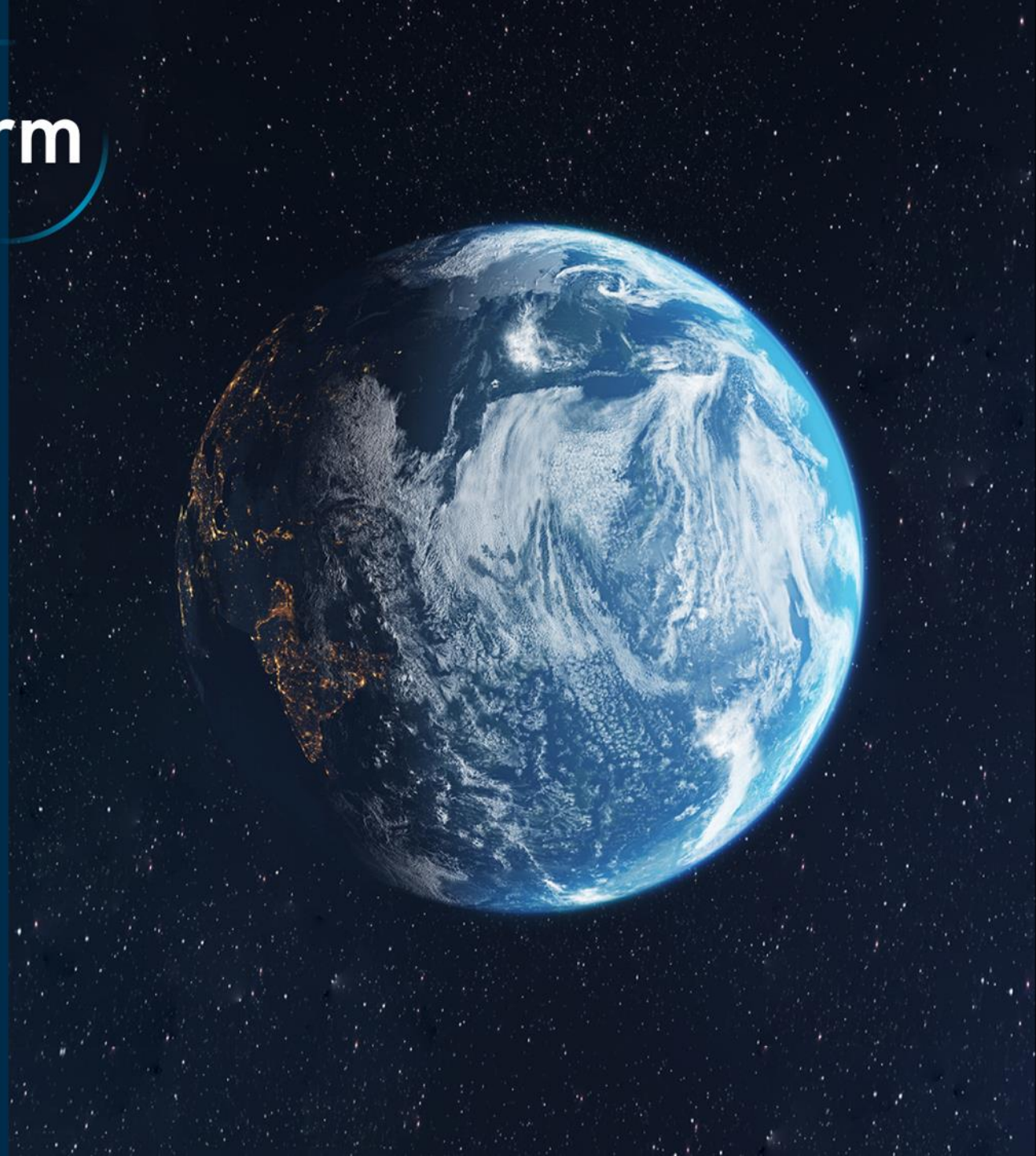
https://camel.clarkson.edu/

Teratec Hackathon

AHUG Workshop
ISC 2023

Gilles Tourpe - HPC Business Developer - AWS
gtourpe@amazon.fr
Conrad Hillairet - Staff HPC Engineer - Arm
conrad.hillairet@arm.com

25th May 2023



Overview and genesis

TERATEC

- AWS and TERATEC
- Proposal / Objectives
  - Educate next HPC talents
  - Think HPC differently / Reality check
  - Promote/Support arm64 ecosystem
    - Accelerate adoption
    - Accelerate tools maturation

Big love to…

Modus operati

Provide students with a Cloud HPC infrastructure to use AWS c7g instances using AWS Graviton3

Provide them a 1st hand experience on using new ARM processors, new architectures, new tools to optimise a mini-app provided bv CGG and port a production grade code on a HPC Cloud infrastructure

100% of the participating teams (10) were to port CGG Stencil code (50% of the evaluation) and then either choose to port Code Telemac or Code Saturne from EDF R&D (50% of the evaluation)

AWS Graviton3

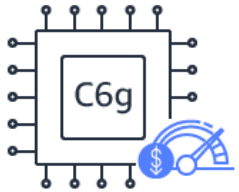
Hardware based on Arm technologies

Graviton2 Processor

Frequency
2.5 GHz

Peak Flops
1280 GFlops

Peak Memory B/W
204 GB/s



Many core
architecture
64 cores

Non-NUMA

7 nm

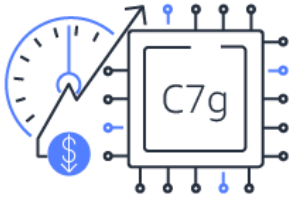
Arm Neoverse N1

Graviton3 Processor

Frequency
2.6 GHz

Peak Flops
2662 GFlops

Peak Memory B/W
307 GB/s



Many core
architecture
64 cores

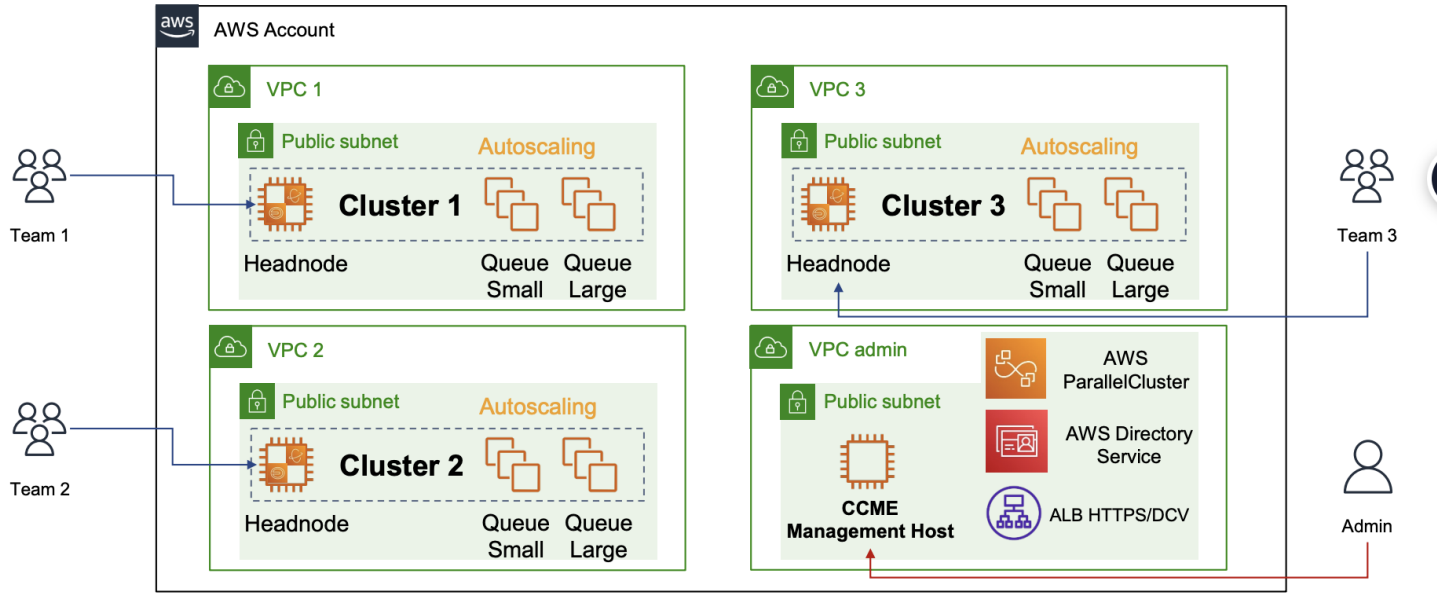
Non-NUMA

Energy
efficiency
5 nm

Arm Neoverse V1

Architecture overview and UCit platform used to deliver it

UCIT Hackathon : The Architecture



- Each cluster is fully isolated from the others (network, accounts…)
- Job scheduler: SLURM
- Compute: 2 partitions
  - Small: c7g.4xlarge (4 vCPUs, 32 GiB) – limit 4 instances
  - Large: c7g.16xlarge (64 vCPUs, 128 GiB) – limit 4 instances
  - ➢ 272 vCPUs available
- Storage: Shared NFS – 500GiB
- Remote access
  - SSH connection to frontend node through login/password
  - Web portal EnginFrame + remote desktop on frontend node

Results

- 10 teams registered
- 7 actively participated

En s'appuyant sur ces codes industriels, ce **hackathon HPC organisé par Teratec et AWS** avec le soutien d'**ARM** et d'**UCit** a permis aux étudiants d'accroitre leur compréhension des enjeux industriels autour de la simulation haute performance et de se familiariser à l'utilisation du Cloud Computing pour le développement, l'analyse de performance et l'exécution de codes de calculs dits HPC.

L'UVSQ a remporté cette compétition en inscrivant 3 équipes sur le Podium. Bravo à l'équipe de Hugo BATTISTON, Guillaume BIGAND, Mathys JAM et Benjamin LOZES qui termine première.
Les équipes "The Assembler" et "Arm yourself" se partagent la seconde place.

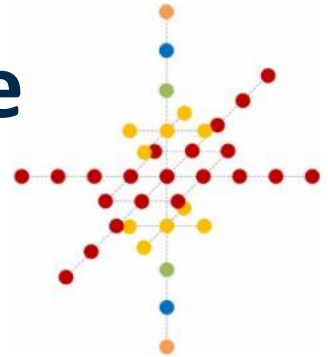
Félicitations à ces trois équipes qui pourront présenter leur travail et échanger avec la communauté HPC lors du Forum Teratec 2023 qui se tiendra au Parc Floral de Paris les 31 mai et 1er juin 2023. En attendant, l'équipe vainqueure s'est vue offrir 4 Macbok

Teratec Hackathon

Two codes – Representative of industrial challenges

code_saturne

Stencil code



EDF

CGG

Teratec Hackathon

You have until Friday to « hack it » !
Good luck.

The Organising Committee.

CGG Stencils

What did they do ?

Math functions calls (pow)

OpenMP parallelization

Limit number of divisions

Vectorization (Neon, SVE)

Compiler Flags

Compiler optimization remarks

Tools : MAP, MAQAO

Remove unnecessary matrix copies

Reordering & unrolling loops

Cache blocking

Intrinsics

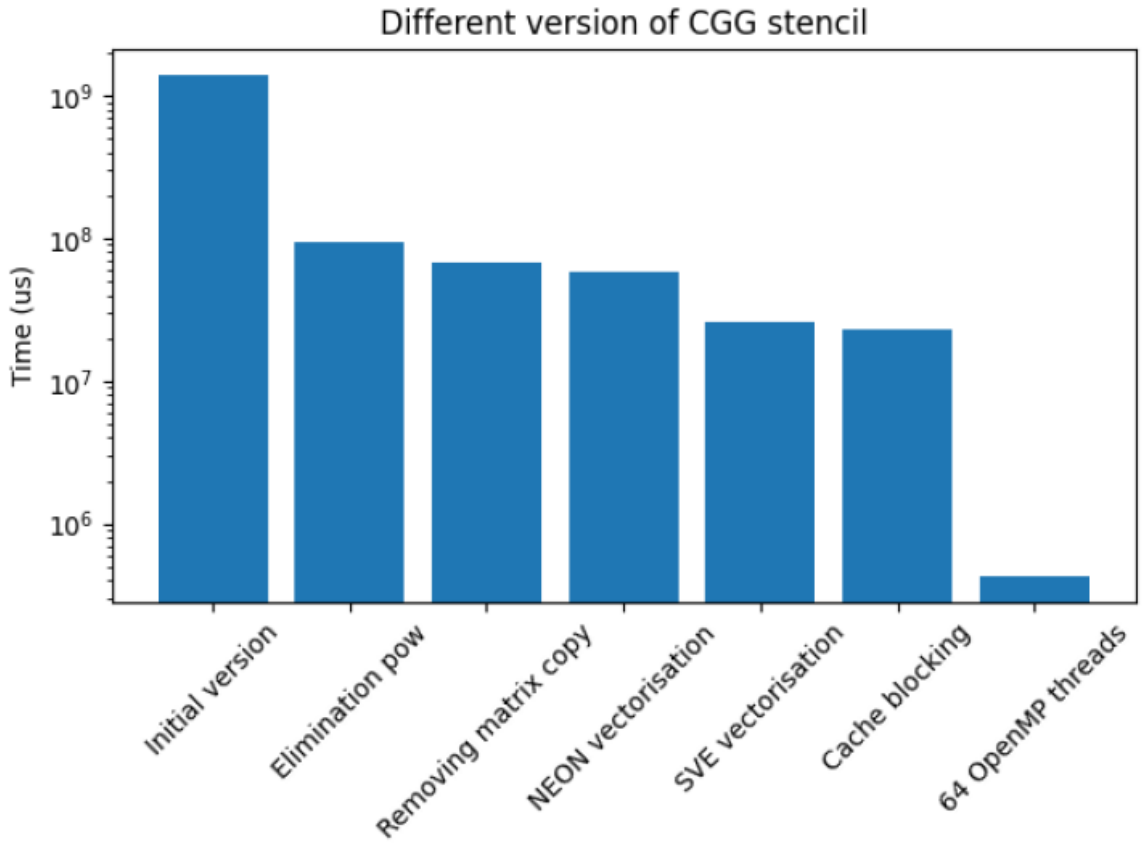


FIGURE 6 – Histogram of the different optimized versions

Best speed-up
7176x

Code_Saturne

Reference - Build instructions on x86 with Intel toolchain

Install

- Download Code_Saturne 7.2.0 (https://github.com/code-saturne/code_saturne/archive/refs/tags/v7.2.0.tar.gz)
- Use the archive
- Install the code locally
  - source ~/intel/oneapi/setvars.sh
  - CC="mpiicc"
  - CXX="mpiicpc"
  - FC="ifort"
  - DEST=$HOME/code_saturne_7.2.0
  - mkdir build
  - cd build
  - ../configure CC="$CC" CXX="$CXX" FC="$FC" \

    --with-blas=$MKLROOT --prefix=$DEST \

    --disable-gui --without-med \

    --without-hdf5 --without-cgns \

    --without-metis --disable-salome \

    --without-salome --without-eos \

    --disable-static --enable-long-gnum
  - make
  - make install

Run

- Download test cases https://github.com/code-saturne/saturne-open-cases
- Go inside folder BUNDLE
- Two test cases available : C016 et F128 , the main difference is the size of the mesh, C016 allows very fast computations (single core) and F128 more precise simulations (single node)
- First step : generate the meshes that will be used later
  - cd $REPBASE/BENCH_C016_PREPROCESS/DATA
  - $DEST/bin/code_saturne run
    - Outputted data are in $REPBASE/BENCH_C016_PREPROCESS/RESU/extrude_16
    - C016 : has 1 024 cells as an input and 17 408 as the output
  - cd $REPBASE/BENCH_F128_PREPROCESS/DATA
  - $DEST/bin/code_saturne run
    - Outputted data are in $REPBASE/BENCH_F128_PREPROCESS/RESU/extrude_128
    - F128 : 100 040 cells in input, 12 905 160 cells in output
- 2 ème étape : run testcase C016 on one node with 48 cores and 1 thread per MPI task :
  - cd $REPBASE/BENCH_C016_01/DATA
  - $DEST/bin/code_saturne submit --wckey dtsi:undefined --nodes=1 -n 48 --exclusive --ntasks-per-node=48 --cpus-per-task=1

Code_Saturne

Reference results for F128 on Cronos supercomputer

CRONOS (https://top500.org/system/179899/)

			reference CRONOS								
			F128_01			F128_02			F128_04		
nombre de nœuds	nombre de cœurs	nombre threads	User CPU time	Total CPU time	Elapsed time	User CPU time	Total CPU time	Elapsed time	User CPU time	Total CPU time	Elapsed time
1	48	1	241	12017	258	1019	50643	1069			
2	96	1	131	13184	142	534	53274	565			
4	192	1	89	17908	q97	295	58426	313	1103	220235	1168
8	384	1	50	20602	56	155	61143	165	599	234654	627
16	768	1	38	32345	45	141	111944	155	333	262441	360
32	1536	1	32	53595	38	59	99592	69	205	321137	229

Adapt configure command to your environment

x86 to aarch64

```
CC="mpiicc"
CXX="mpiicpc"
FC="ifort"
DEST=$HOME/code_saturne_7.2.0
mkdir build
cd build
../configure CC="$CC" CXX="$CXX" FC="$FC" \
             --with-blas=$MKLROOT --prefix=$DEST \
             --disable-gui --without-med \
             --without-hdf5 --without-cgns \
             --without-metis --disable-salome \
             --without-salome --without-eos \
             --disable-static --enable-long-gnum
make
make install
```

```
CC=mpicc CXX=mpicxx FC=armflang ./configure \
  --with-blas=$ARMPL_DIR --prefix=$PWD/build \
  --disable-gui --without-med                \
  --without-hdf5 --without-cgns              \
  --without-metis --disable-Salome           \
  --without-salome --without-eos             \
--disable-static --enable-long-gnum
```

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for python3 extension module
directory...
${exec_prefix}/lib64/python3.7/site-packages
checking for dlopen... dlopen
checking for MKL libraries... no
checking for threaded ATLAS BLAS... no
checking for ATLAS BLAS... no
checking for legacy C BLAS... no
configure: error: in `/home/ec2-
user/code_saturne-7.2.0':
configure: error: BLAS support is requested,
but test for BLAS failed!
See `config.log' for more details
```

- Need to introduce support for ArmPL
- Modify one file
- Adapt ATLAS and MKL detection mechanisms

Adapt configure command to your environment

x86 to aarch64

```
[...]
${exec_prefix}/lib64/python3.7/site-packages
checking for dlopen... dlopen
checking for MKL libraries... no
checking for threaded ATLAS BLAS... no
checking for ATLAS BLAS... no
checking for legacy C BLAS... no
checking for ArmPL libraries... yes
checking for MPI (MPI compiler wrapper
test)... yes
checking for MPI I/O... yes
checking for MPI in place... yes
checking for MPI nonblocking barrier... yes
CCMIO headers not found
[...]
```

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for a sed that does not truncate
output... /usr/bin/sed
checking for ar... ar
checking the archiver (ar) interface... ar
configure: sourcing config/cs_auto_flags.sh
(test for known compilers)
compiler 'mpicc' is NVIDIA compiler
compiler 'mpicxx' is NVIDIA compiler
compiler 'armflang' is NVIDIA compiler
checking how to print strings... printf
[...]
```

- Bug in compiler detection mechanism
- Arm Compiler for Linux detection overwritten by NVIDIA detection
- Two files to modify (~10-20 lines)

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for ar... ar
checking the archiver (ar) interface... ar
configure: sourcing config/cs_auto_flags.sh
(test for known compilers)
compiler 'mpicc' is Arm C compiler
compiler 'mpicxx' is Arm C++
compiler 'armflang' is Arm Fortran compiler
checking how to print strings... printf
[...]
```

Compile

x86 to aarch64

```
make -j 4 > resComp >&1
make install


grep -i err ./resComp
```

No problem, compiles OoB

Flat MPI version

x86 to aarch64

```
CC=mpicc CXX=mpicxx FC=gfortran ./configure --
prefix=$PWD/build-noomp --disable-gui --without-hdf5 --
without-cgns --without-med --without-metis --disable-static
--disable-salome --without-salome --without-eos  --enable-
long-gnum --with-blas=$ARMPL_DIR --disable-openmp > resConf
2>&1

grep -i openmp ./resConf
[…]
AArch64_RHEL-7_aarch64-linux/bin/../lib/gcc/aarch64-linux-
gnu/11.2.0/../../.. -lgfortran -lm
checking for OpenMP (C)... yes
checking for OpenMP (Fortran)... yes
checking for Fortran libraries of gfortran... (cached) OpenMP
support: no
 OpenMP support: yes
 OpenMP accelerator support: no
 OpenMP Fortran support: yes
 Auto load environment modules:  binutils/11.2.0 gnu/11.2.0
armpl/22.1.0 openmpi/gcc/4.1.4
```

- Bug in OpenMP configuration
  not Arm specific
  also happens on x86

- One file to modify (~1 line)

Flat MPI version

x86 to aarch64

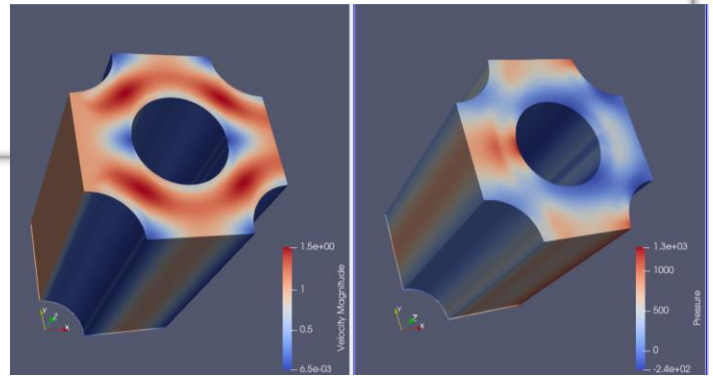
```
CC=mpicc CXX=mpicxx FC=gfortran ./configure --
prefix=$PWD/build-noomp --disable-gui --without-hdf5 --
without-cgns --without-med --without-metis --disable-static
--disable-salome --without-salome --without-eos  --enable-
long-gnum --with-blas=$ARMPL_DIR --disable-openmp > resConf
2>&1

grep -i openmp ./resConf
[…]
OpenMP support: no
OpenMP support: no
 Auto load environment modules:  binutils/11.2.0 gnu/11.2.0
armpl/22.1.0 openmpi/gcc/4.1.4


make > resComp 2>&1
grep fopenmp ./resComp | wc -l
0
```

Run

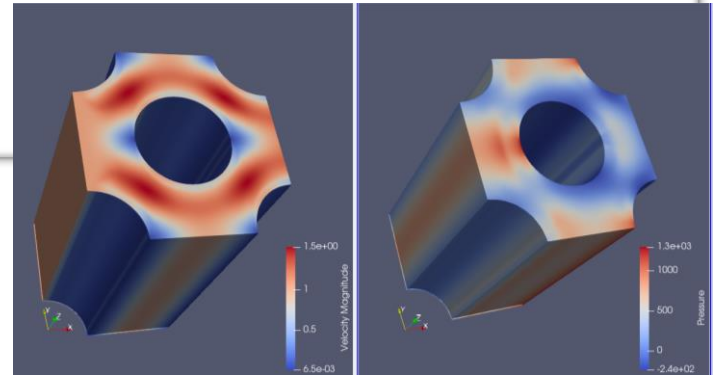
x86 to aarch64

No problem at run time

```
First step : generate the meshes that will be used later
cd $REPBASE/BENCH_F128_PREPROCESS/DATA
$DEST/bin/code_saturne run

Second step: run testcase C016 on one node with 48 cores and 1
thread per MPI task :
cd $REPBASE/BENCH_C016_01/DATA
$DEST/bin/code_saturne submit --wckey dtsi:undefined --nodes=1 -n
48 --exclusive --ntasks-per-node=48 --cpus-per-task=1
```



```
First step : generate the meshes that will be used later
cd $REPBASE/BENCH_F128_PREPROCESS/DATA
$DEST/bin/code_saturne run

Second step: run testcase C016 on one node with 64 cores and 1
thread per MPI task :
cd $REPBASE/BENCH_C016_01/DATA
$DEST/bin/code_saturne submit -n 64 --cpus-per-task=1
```

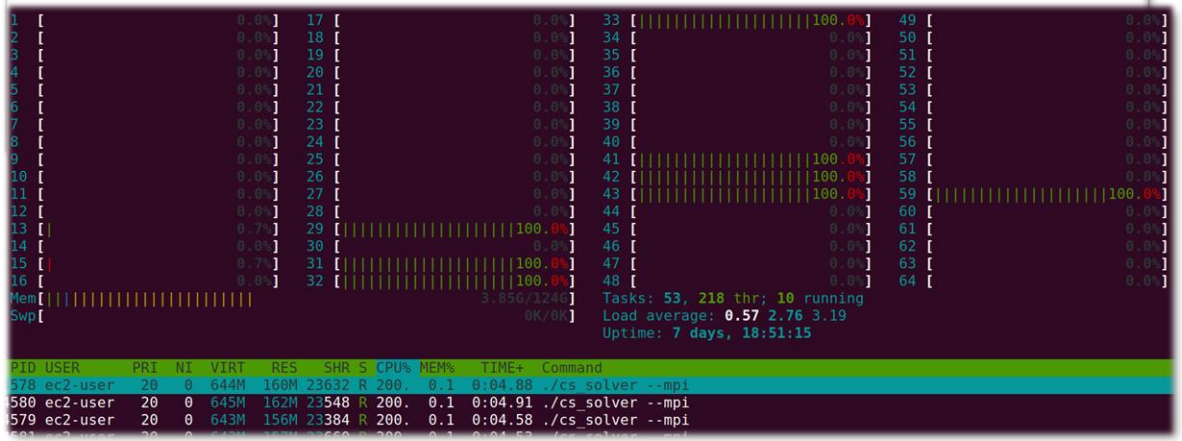


Another run with MPI & OpenMP

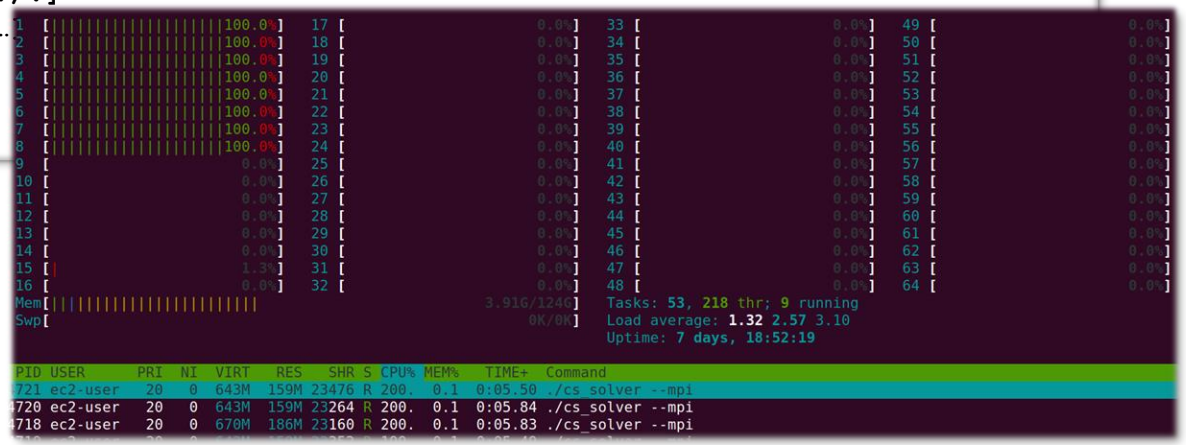
x86 to aarch64

Not Arm specific

```
code_saturne submit -n 4 --nt 2 --cpus-per-task=2
```

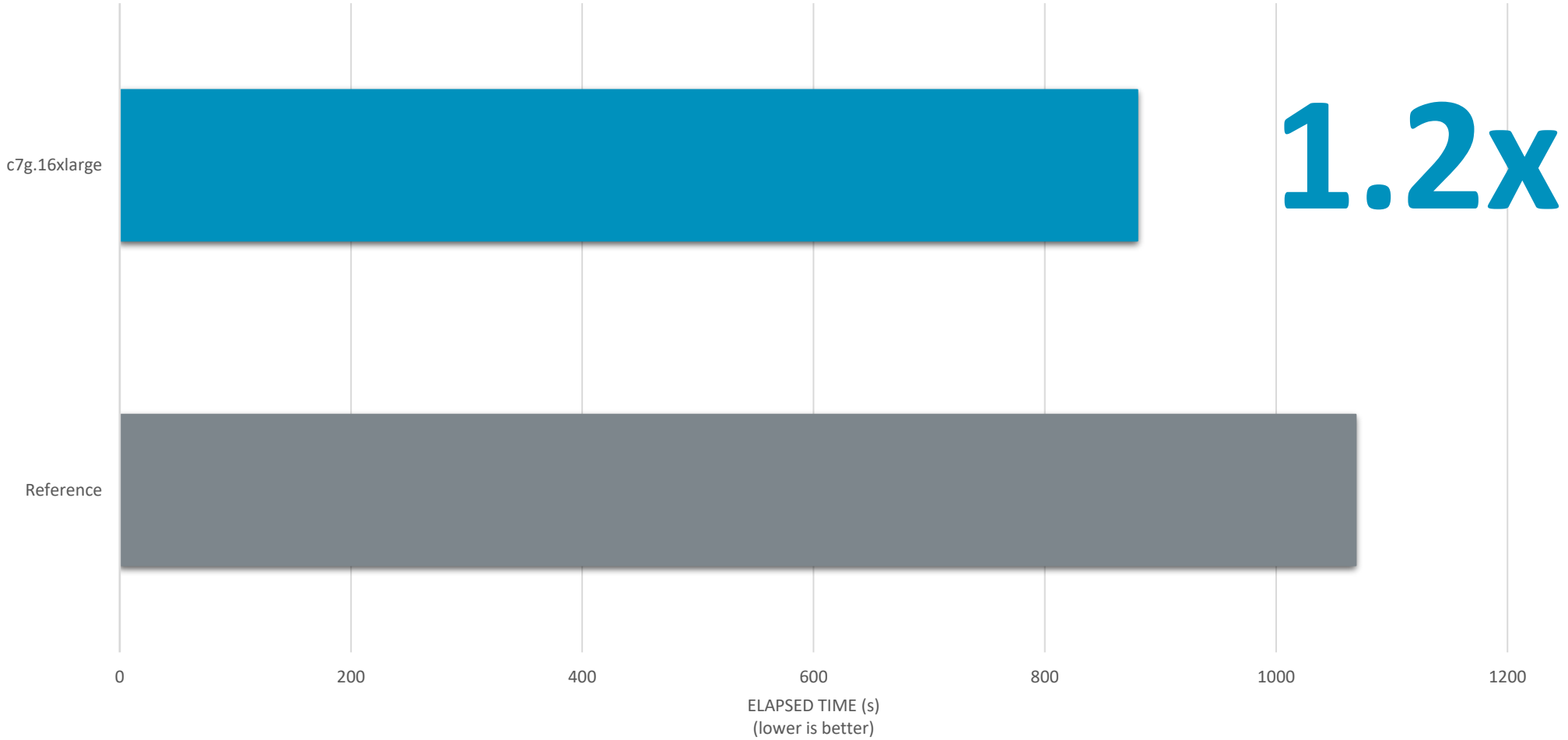



```
export CS_MPIEXEC_OPTIONS="--map-by ppr:4:node:PE=2 --report-
bindings"
code_saturne submit -n 4 --nt 2
[ip-172-31-77-39:04714] MCW rank 0 bound to socket 0[core
0[hwt 0]], socket 0[core 1[hwt 0]]:
[B/B/././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././.
/./.]
[ip-172-31-77-39:04714] MCW rank 1 bound to socket 0[core
2[hwt 0]], socket 0[core 3[hwt 0]]:
[./B/B/./././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././.
/./.]
[…
```



Code_Saturne

AWS Graviton 3 – Arm Neoverse V1



Conclusion

"This hackathon was the opportunity for us to test an architecture we never explored before. But before all, it challenged our skills, and forced us to reconsider our weaknesses. While most members of the team are experienced programmers, we had to use a lot of different tools to speedup our analysis process. Tools we overlooked before, or did not take the time to learn. This is especially true for Code Saturne. **We really appreciate the effort put in by the organizing committee, for the quality of the infrastructure provided, and especially for the difficulty and variety of the problems we had to solve.**

We strongly believe this challenge was of value, and hope that our efforts presented here will be appreciated."

And most importantly

We found the source of all our problems …

"Either the intrinsic calls we did were not the best ones or **the compiler did not understand what we wanted to do.**"

Next

Teratec

https://teratec.eu/activites/Hackathon.html

More to come for aarch64 adoption

- Euromaster4HPC Summer School (07/23)
- EPITA HPC new major (CY24)
- University of Luxemburg Summer School (06/23)
- …

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

AHUG WORKSHOP @ ISC'23

AWS Graviton 3E

Updates from the field

Brendan Bouffler ('boof')

HPC Engineering @ AWS

Amazon EC2 | The compute platform for every workload

Workload types


Machine Learning


High-Performance Computing


Media Rendering


Containers


Web-based Apps


Batch Processing


Big Data

Instance types for HPC workloads

HPC Optimized

Hpc7g | Hpc6a | Hpc6id

Coming soon

Compute, Memory, and Networking

C7gn | C7g | C6i | M6i

R6i | X2iezn | C5n | M5zn

Accelerators

P4d | P5 | G5 | Trn1

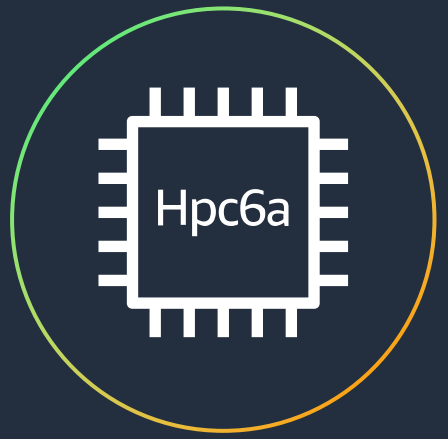
DL1 | F1 | Inf1 | VT1

Scale tightly and loosely-coupled HPC applications

- Choice of processor (e.g., Graviton, Intel, AMD)
- Scale tightly-coupled HPC and ML workloads
- Up to 400 Gbps network bandwidth
- < 15 micro-seconds network latencies
- Accelerators use hardware to perform functions more efficiently than is possible in software running in CPUs

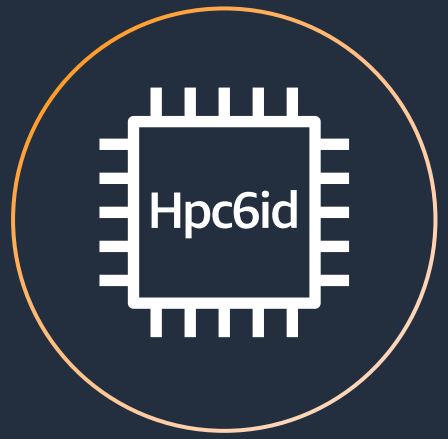
HPC instance choice for diverse workloads

X86 - AMD



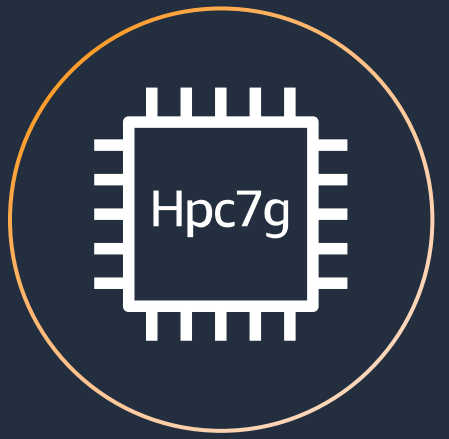
- AMD Milan
- 96 physical cores / instance
- Up to 3.6 GHz
- 384 GB RAM
- 100 Gbps EFA

X86 - Intel



- Intel Ice Lake
- 64 physical cores / instance
- Up to 3.5 GHz
- 1 TB RAM
- 15.2 TB NVMe
- 200 Gbps EFA

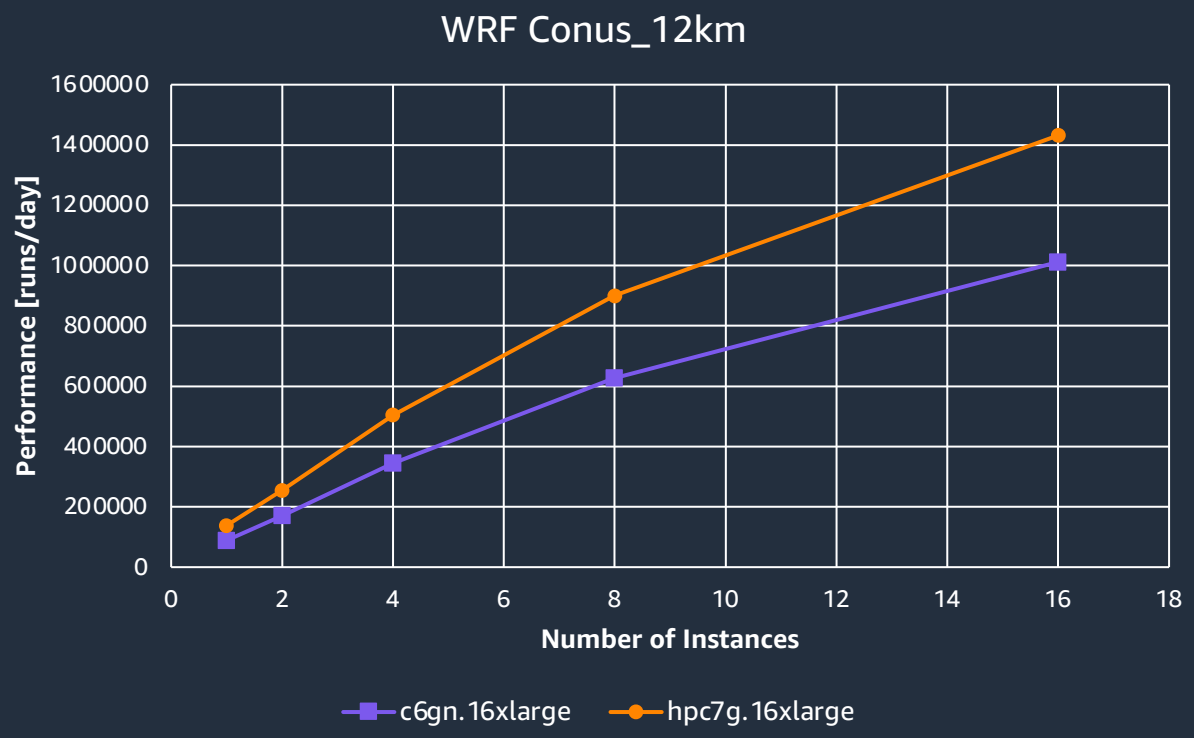
ARM – AWS (coming soon)



- AWS Graviton3E
- 64 physical cores / instance
- Up to 2.6 GHz
- 128 GB RAM
- 200 Gbps EFA

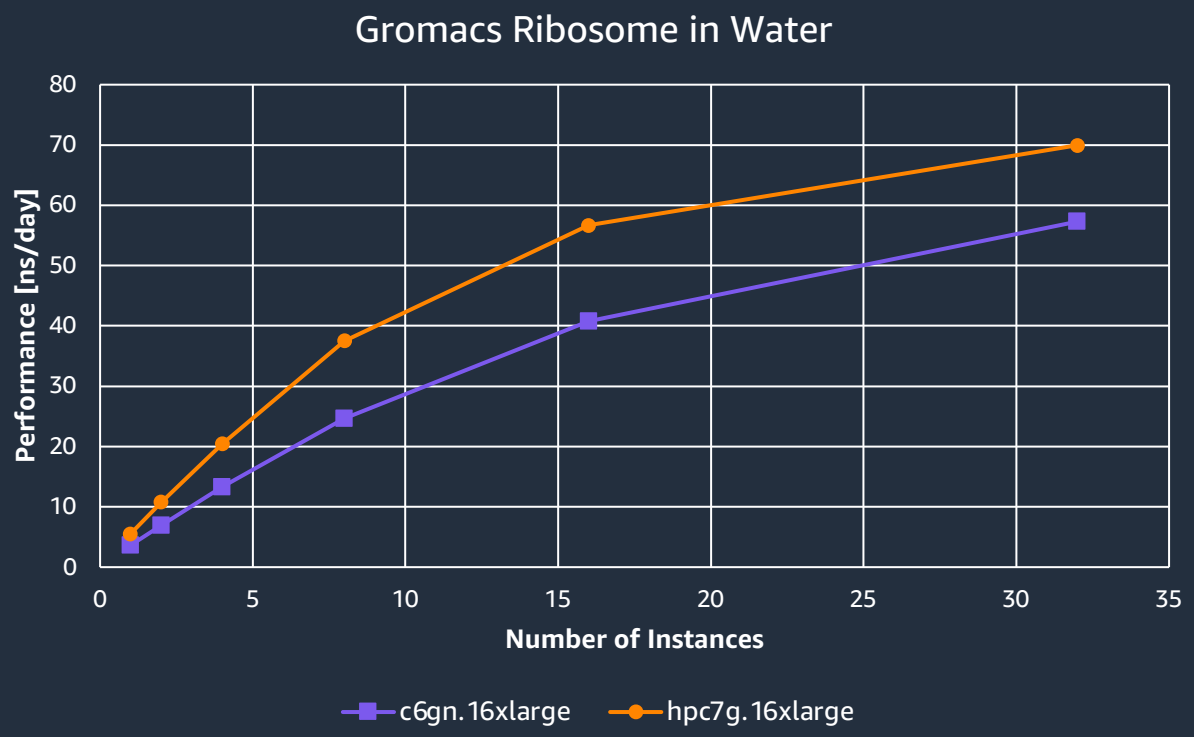
Example Hpc7g Performance: WRF

- Continental United States 12x12km (127,500 grid points)
- **56%** speed-up on single instance
- **41%** speed-up at 16 instances



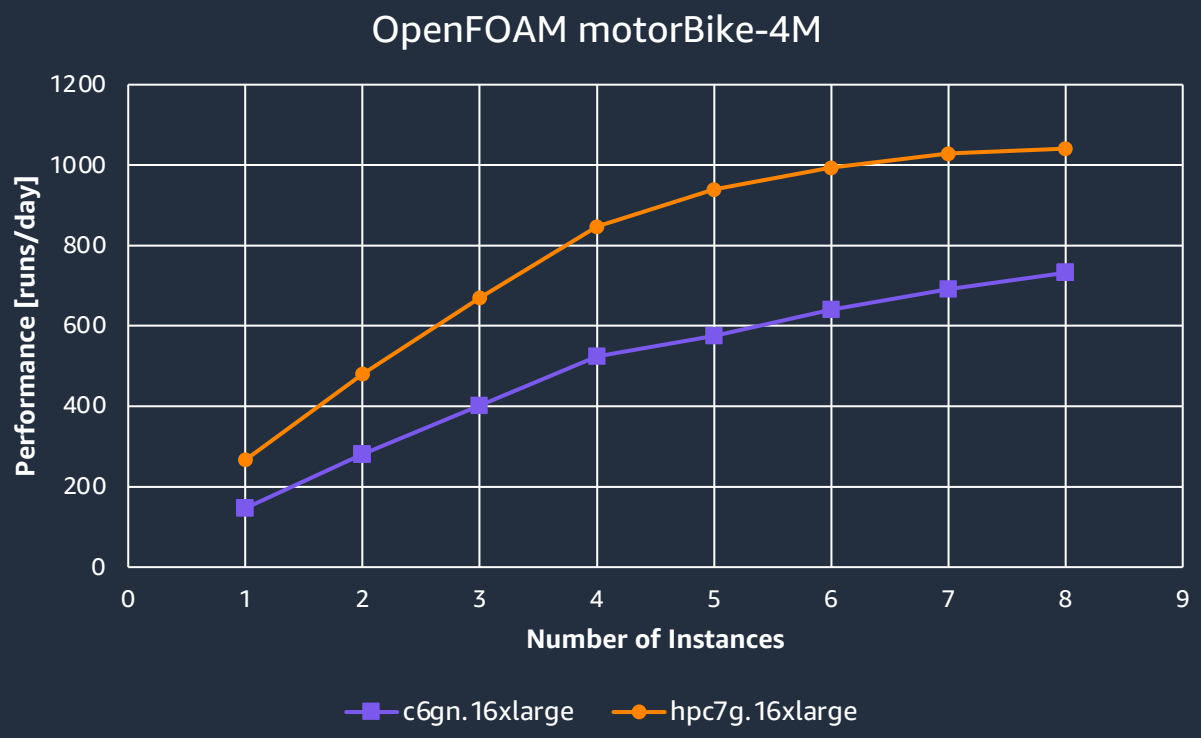
Example Hpc7g Performance: Gromacs

- 2M atoms Ribosome in Water
- **53%** speed-up on single instance
- **22%** speed-up at 32 instances



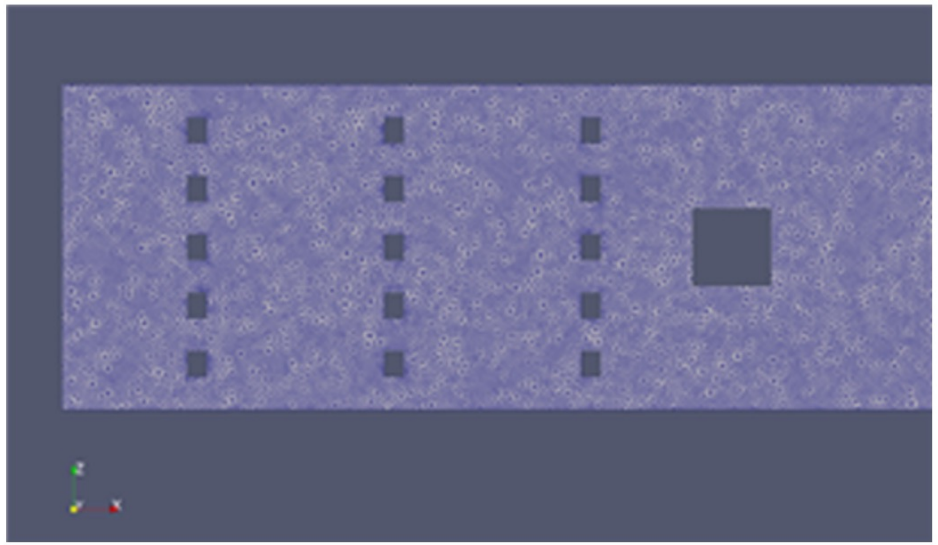
Example Hpc7g Performance: OpenFOAM

- Small motorBike test case (4M cells) to show scaling bottlenecks
- **82%** speed up on single instance
- **42%** speed-up on 8 instances

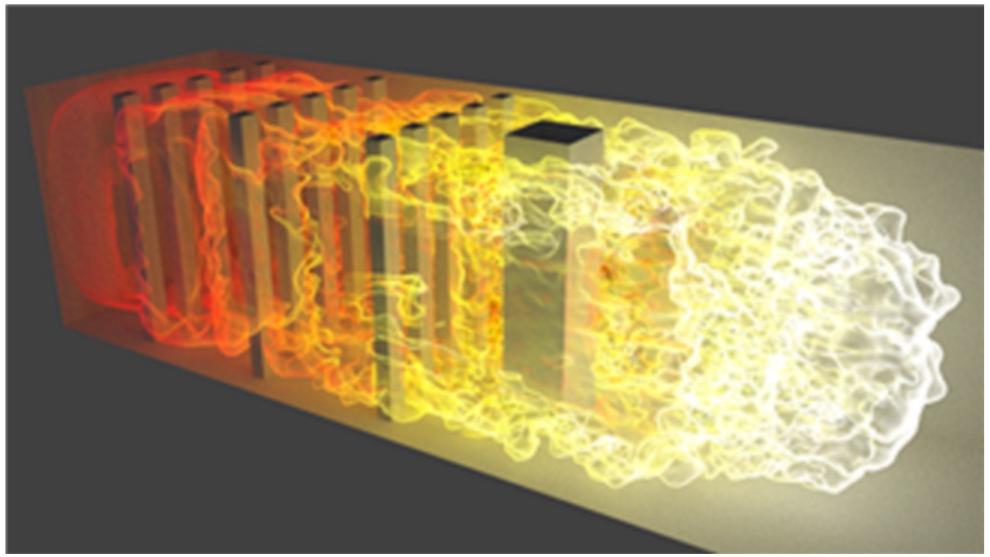


CERFACS: Hpc7g use case

- It is a state-of-the-art simulation of an explosion. Modelling and algorithms match industrial applications used by CERFACS partners (AIRBUS, SAFRAN, Total).
- From the previous collaboration benchmark we updated the AVBP version to 7.9. The grid has 60M Tetra.

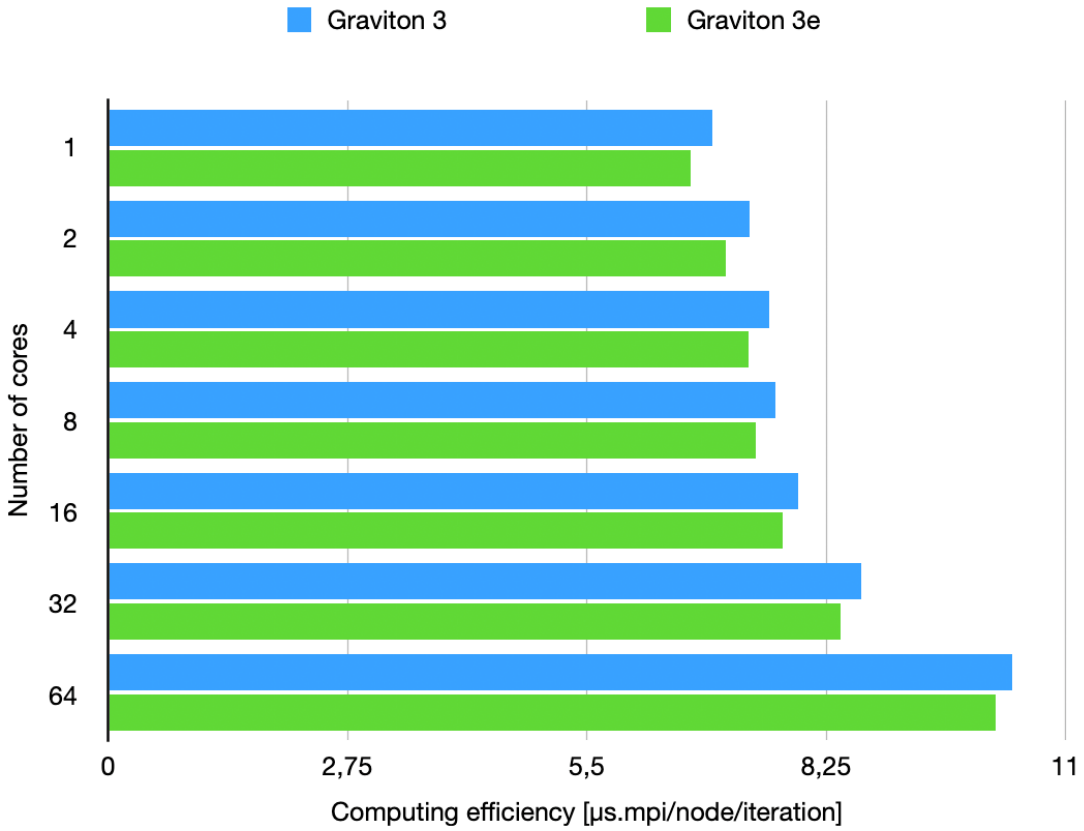


Top view of a mid-cut of the grid.



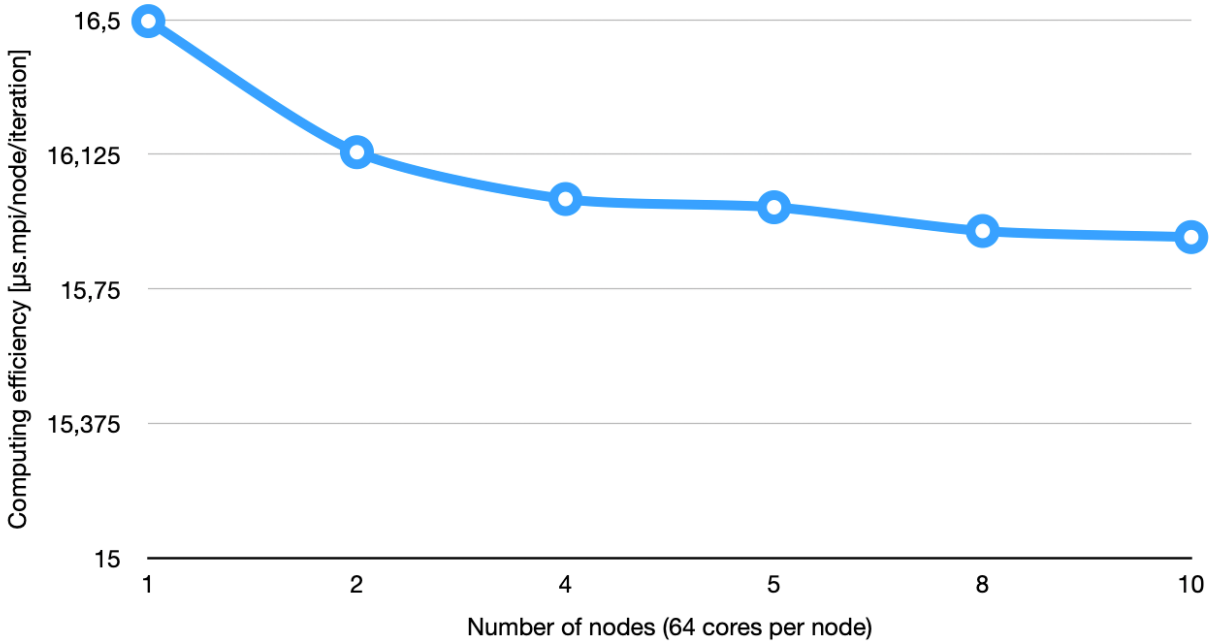
Isosurface of temperature rendering (Gullaud et al)

Single node performance



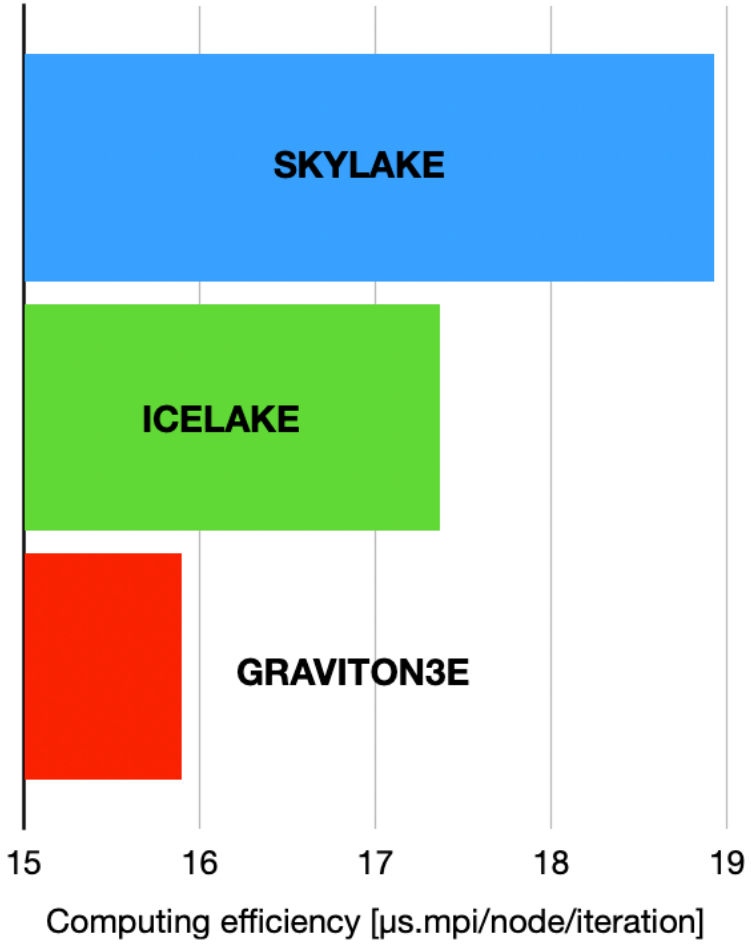
- This graph shows the comparative performance of graviton3 and 3e.
- Performance is measure using 10 iteration without I/O
- Overall graviton3e is 4% more efficient than graviton3
- Strong scaling from 1 core to full node (64 cores) is excellent. For further tests full nodes will be used.
- Graviton3e is 186% faster than graviton2

Multi node Performance



- This graph shows the strong scaling of graviton3e from 1 to 10 nodes. Performance measured in production mode with 1000 iterations.
- Perfect scaling would be a straight line.
- Here we can see that performance is near perfect, with a 3.7% loss from 1 to 10 nodes.

Multi node Performance

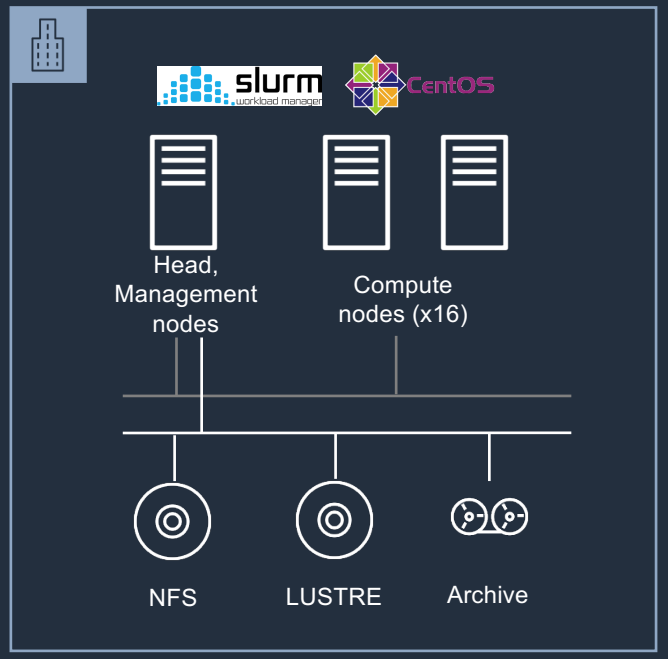


- This graph shows the efficiency of graviton3 in a production mode measured using 6000 iterations.
- 15 skylake nodes (540 cores) versus 10 Icelake nodes (760 cores) versus 10 graviton3 (640 cores) are compared here in terms of computing efficiency per mpi per node per iteration.
- Graviton3 is 16% and 8.4% faster than comparable Skylake and Icelake nodes.

Simcenter STAR-CCM+

Tooling

ON-PREMISES



PARALLELCLUSTER RECIPE (YAML)

```
Image:
  Os: centos7

HeadNode:
  InstanceType: c5.4xlarge
  Networking:
    SubnetId: subnet-b46032ec
  Ssh: KeyName: My_PC3_KeyPair
  AllowedIps: 0.0.0.0/0

Scheduling:
  Scheduler: slurm
  SlurmSettings:
    ScaledownIdletime: 10
    Dns:
      DisableManagedDns: true
  SlurmQueues:
    - Name: q1_ondemand
      ComputeSettings:
        LocalStorage:
          RootVolume:
            Size: 100
        CapacityType: ONDEMAND
        ComputeResources:
          - Name: compute-resource-1
            InstanceType: c5.n18xlarge
            Efa:
              Enabled: true
            MinCount: 0
            MaxCount: 64
            Networking:
              SubnetIds:
              - subnet-a12321bc
              PlacementGroup:
                Enabled:true

SharedStorage:
  - MountDir: /shared
    Name: myebs
    StorageType: Ebs
    EbsSettings:
      VolumeType: gp3
      Size: 100
  - MountDir: /lustre
    Name: myfsx
    StorageType: FsxLustre
    FsxLustreSettings:
      StorageCapacity: 1200
      DeploymentType: SCRATCH_2
      ImportPath: s3://myhpcbucket
```

OS

Head node

Scheduler Settings

Compute nodes

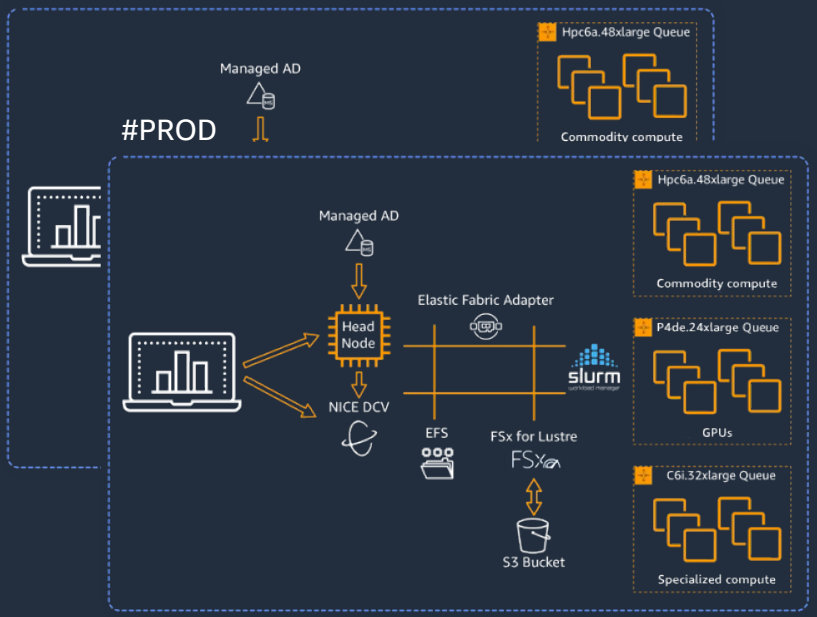
Network settings

NFS Storage

Lustre storage

#DEV

#PROD



Recipe becomes the manageable asset.

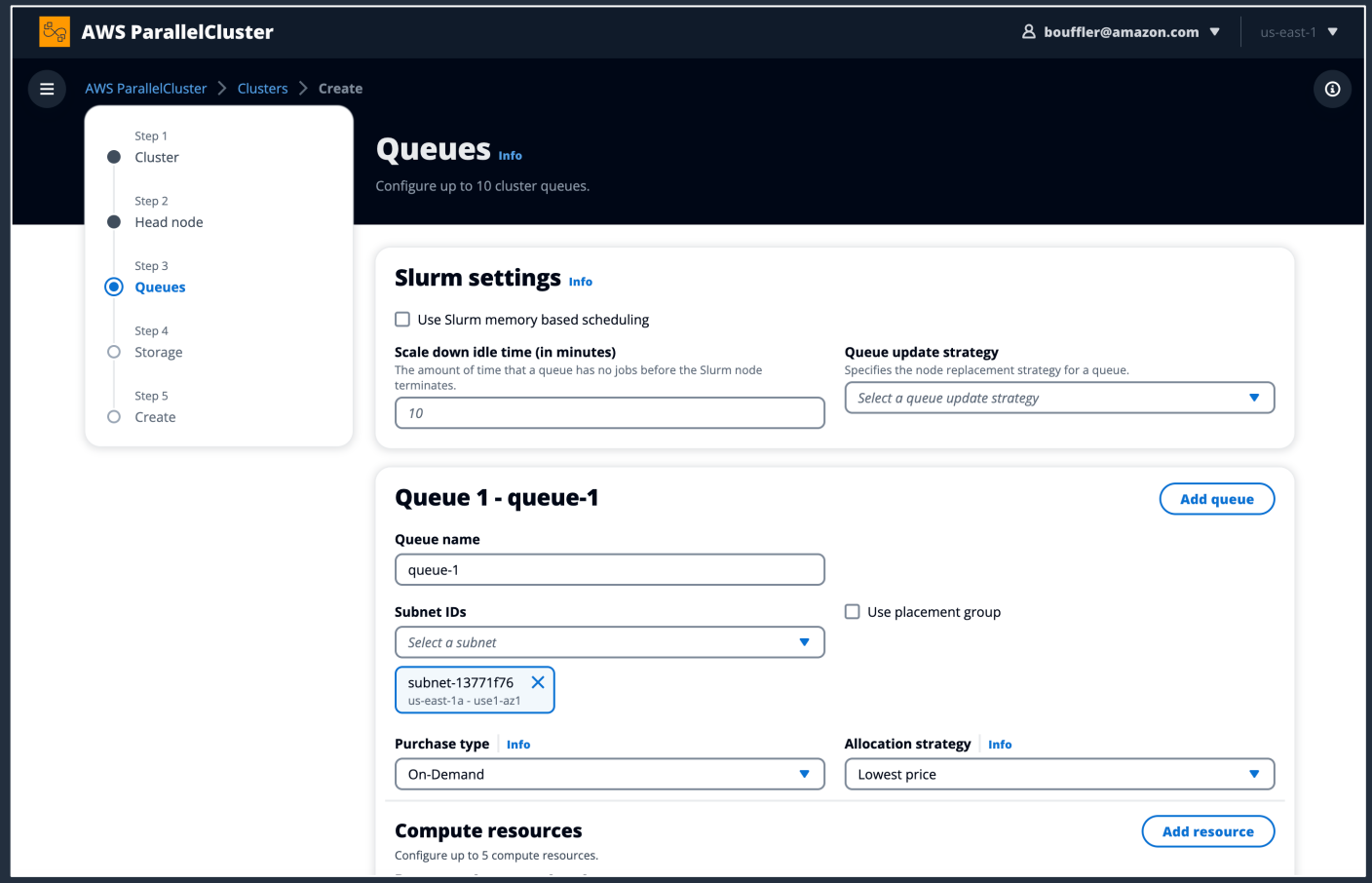
AWS ParallelCluster UI

Guided interface for designing and deploying ParallelCluster

Design and deploy multiple clusters through a guided process

Manage and debug all your clusters (multi-region) through a single dashboard interface.

Connect to clusters securely through remote desktop or command line interface



Spack Configs for AWS ParallelCluster

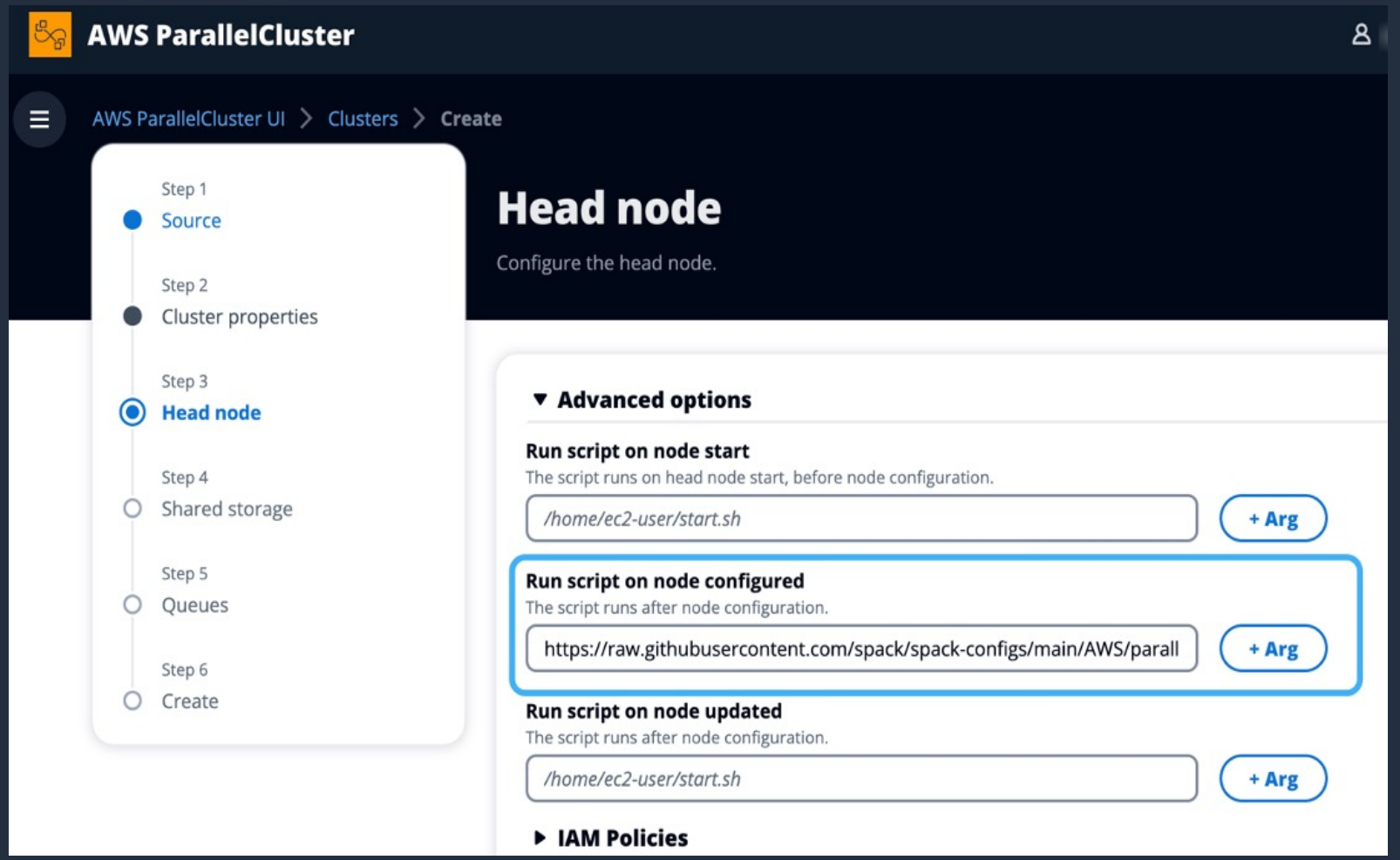
- Validated best-practices from AWS HPC Performance Engineering
- Fixes and general optimizations for any application
- Tuned configurations for common HPC workloads
- Supports Intel, AMD, Graviton
- Easy to use

Application	Speed-up
OpenFOAM	N/A
WRF	**11.9**
MPAS	N/A
GROMACS	**1.16**
Quantum Espresso	**1.53**
Devito	N/A
LAMMPS	**16.6**

Speedup relative to original Spack configs. N/A indicates no original Spack build was available.

hpc.news/down158

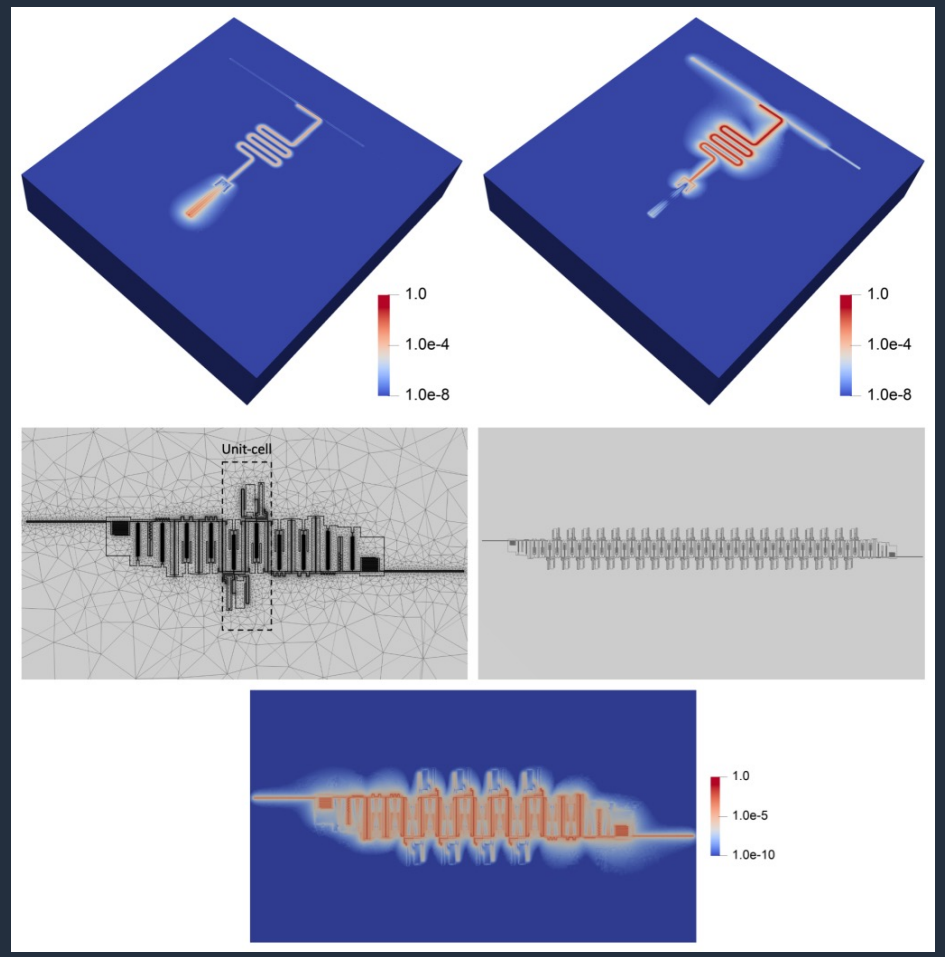
Using Spack Configs for AWS ParallelCluster



- Configure cluster to run script on Head Node during boot-up
- Script runs asynchronously
- Parameterizable – specify what packages to install

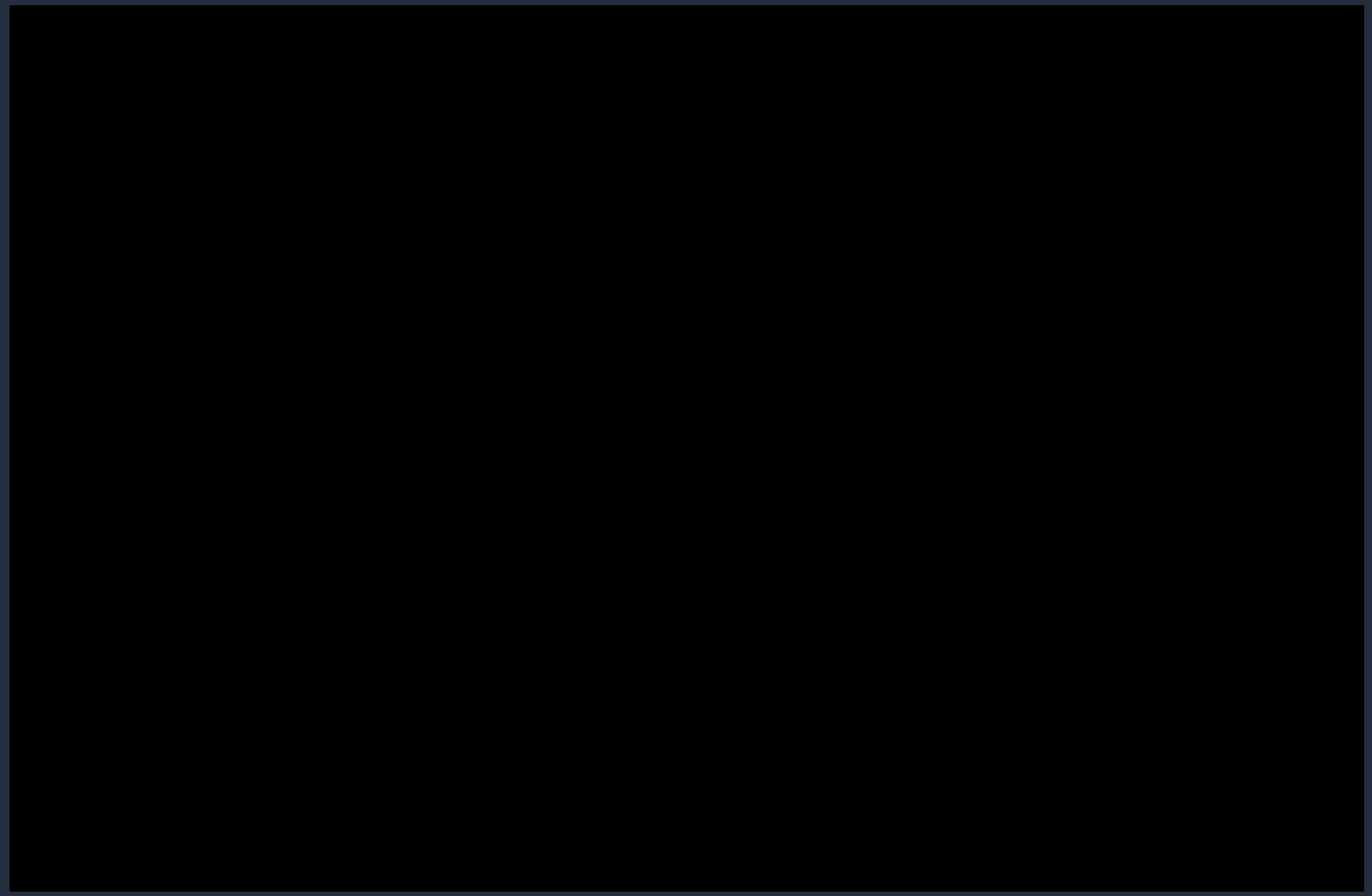
Palace: 3D Finite Element Solver for Computational Electromagnetics

- Enable large-scale 3D simulations of complex electromagnetics models & enable the design of quantum computing hardware
- Optimized for EFA and EC2 HPC instances
- Leverages MFEM from LLNL
- Deep dependency stack. Great for Spack.



https://awslabs.github.io/palace/stable/

From zero to PALACE in 4 minutes…



- Start with naive c7gn head node
- Re-use postinstall.sh script to install Spack
- Install GCC 12
- Optimize dependencies for neoverse V1
- Everything is in build cache except for **armpl-gcc**
- Python, Open MPI, & Petsc installed
- 84 packages in 4 minutes. Plus architecture optimizations!

Outcomes and next steps

Installation *sans* build cache is ~1 hour on a full G7 instance

Added ARMPL support for Palace (should boost performance)

Implemented using Spack CICD pipelines

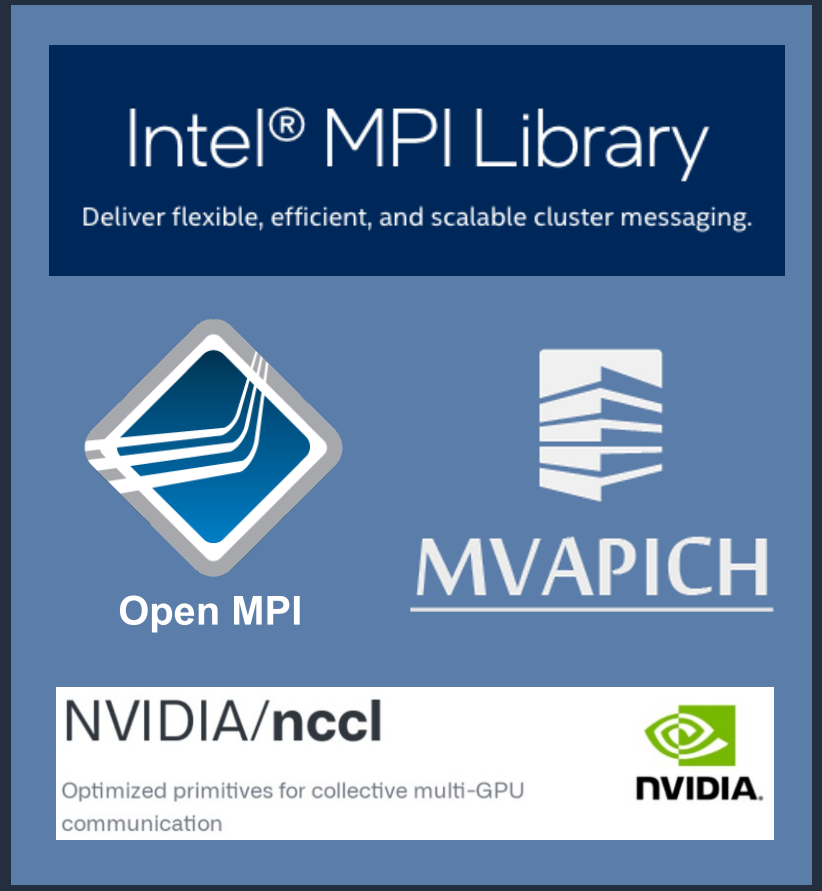
PALACE, OpenFOAM, GROMACS fully available in build cache

- OPENFOAM install: 16 seconds
- GROMACS install: 1 second

Lingering issues with Icelake, Skylake, AMD fixed soon

MPIs and NCCL

- EFA is presented as a libfabric provider
- Known MPIs:
  - Intel MPI supports EFA
  - MVAPICH supports EFA
  - AWS provides support for Open MPI on EFA
- NVIDIA's Collective Communications Library (NCCL) supports EFA.



What else?

Dear Boof,

We look after an open-source thing called foo which is used by the bar community to resolve fubars in labs all around the world.

We have this slightly crazy idea that if we were able to build and maintain an Arm64 variant, we could resolve 2x more fubars than most labs can currently afford.

Sincerely, AHUG people

Dear Boof,

We look after a tool that nearly every scientific developer could use to eliminate the daily crap they need to do in order to get a science code developed quickly on Arm64.

Can we talk?

Sincerely

Dear Boof,

I have a pain in all the diodes down my left-hand side.

Sincerely, Marvin

Thank you!

Boof

Twitter: **@boofla**

Email: bouffler@amazon.com

SIPEARL

Evaluation and performance projections for ARM chips
AHUG workshop / ISC 2023

Etienne Renault

SiPearl, the company born from a European will

September 2018

EuroHPC Joint Undertaking

Launch of the EuroHPC Joint Undertaking
backed by a €8bn budget
to deploy in Europe a world class exascale
supercomputing infrastructure

December 2018

European Processor Initiative

Launch of a call for proposals in 2017 for developing
a new generation of high-end European microprocessors

- Budget: €150m
- Target: high-performance and energy-efficiency

Coordinated by Bull (Atos Group), the European Processor Initiative (EPI) consortium won this call for proposals. It has currently 28 members:

- Scientists: research institutes, universities and supercomputing centres
- Industry: European leaders, IT, electronics and automotive specialists

June 2019

SiPearl is the private company created within the EPI
to launch a strategic industry for Europe.

SiPearl in a nutshell

Building the world first energy-efficient HPC-dedicated microprocessor designed to work with any third-party accelerator (GPU, artificial intelligence, quantum).

Incorporated in June 2019

Key personnel from
Atos, MediaTek, ST, Marvell, NXP, Infineon, intel, NOKIA

Funded by the European Union

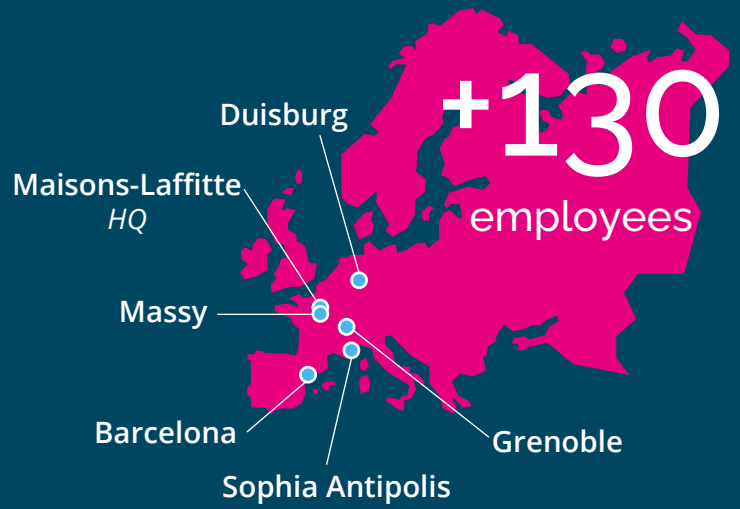
Financing
Initial closing of the Series A: €90m
arm, European Innovation Council Fund, Banque européenne d'investissement, EVIDEN, France 2030

ARM architecture
Energy-efficiency, quick time to market, proven ecosystem

Fabless
Manufactured by TSMC, mandatory at this technological level

Identified customers
Server manufacturers based on user specifications (governments, supercomputing centres, academics, industries, etc.)

6 locations in Europe



Motivation

Have insights from

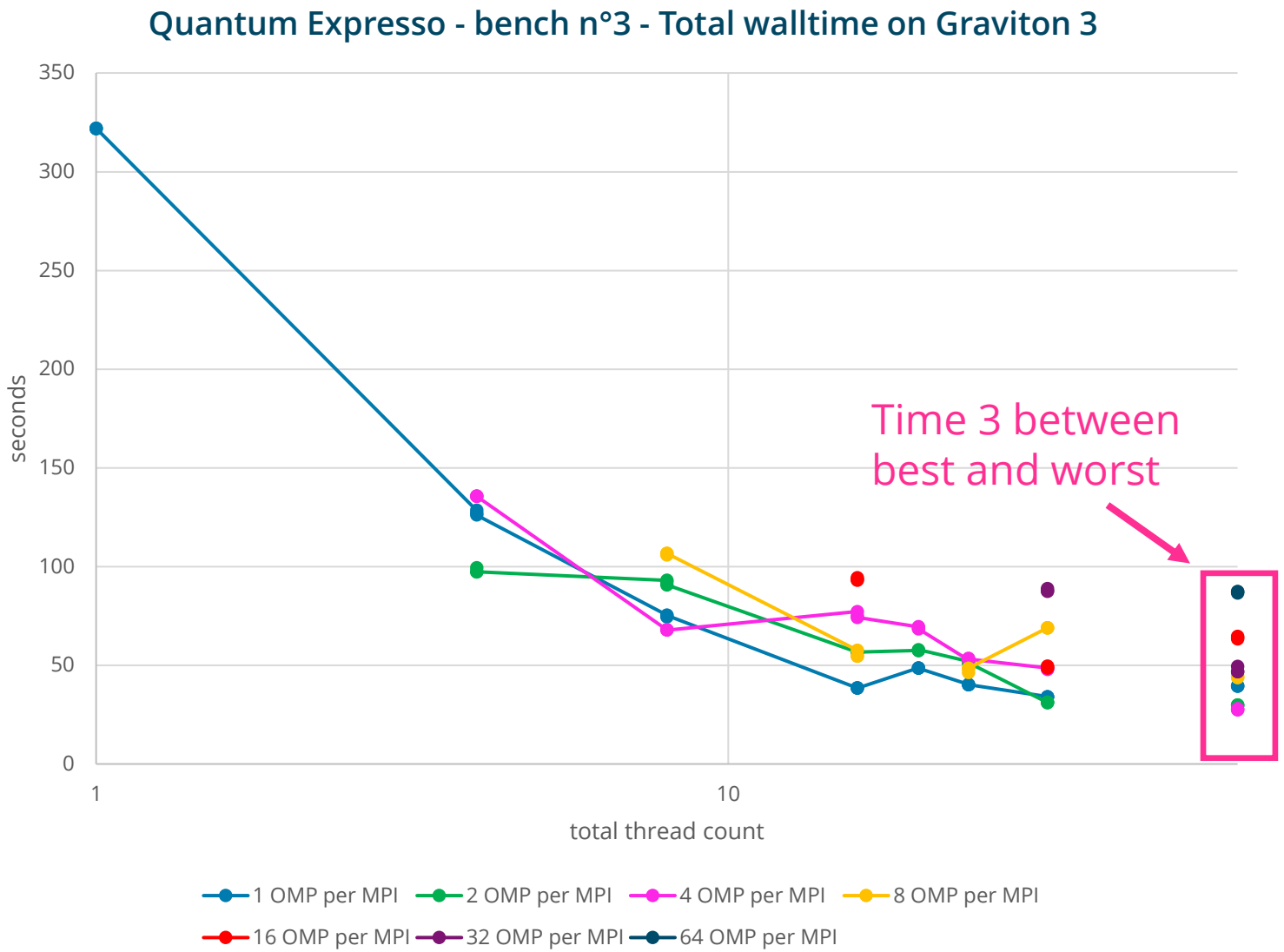
- What can be achieved from a given benchmark ?
- …On a given chip

Avoid brute force by anticipating the best combination to use

- Machines
- MPI processes
- OMP threads
- Numa nodes
- …

Build new chips

- SiPearl et al. job



Projection considered 1/2

All presented metrics are output agnostic

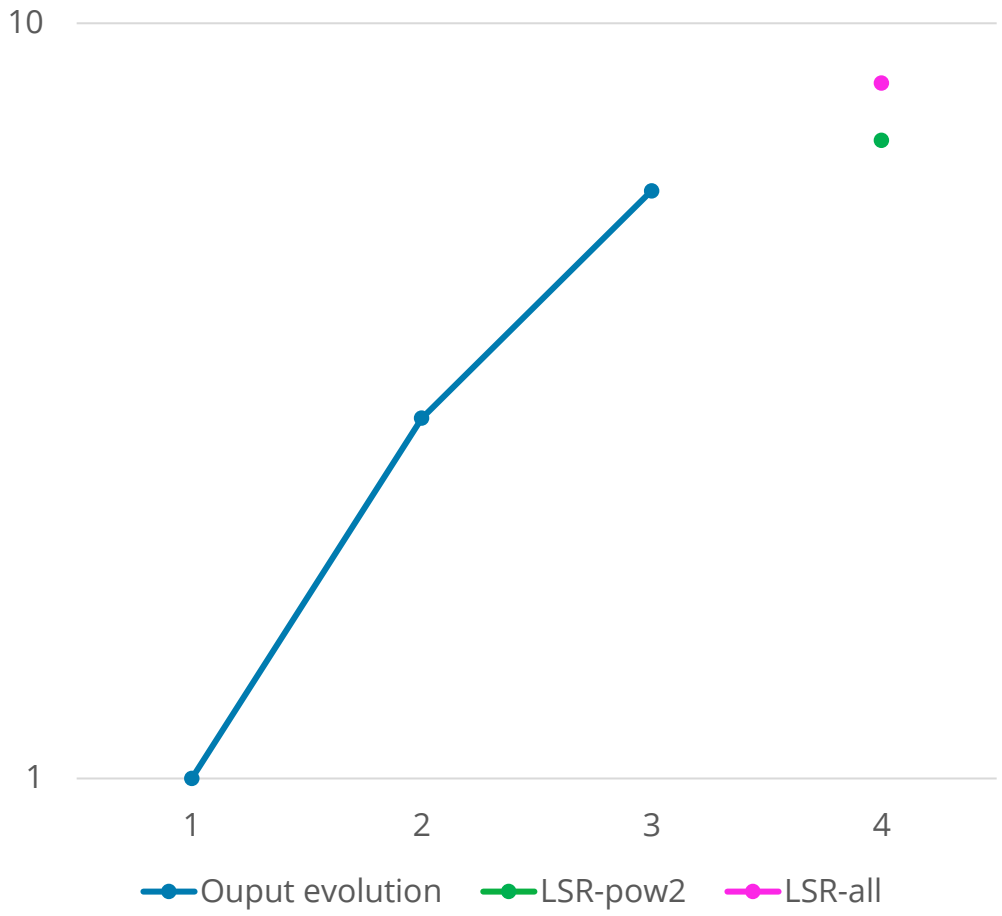
Least Square Root (LSR)

Compute next value (y) using $y = a + b.x$ using

$$a = \bar{y} - b.\bar{x} \qquad b = \frac{\sum(x - \bar{x}).(y - \bar{y})}{\sum(x - \bar{x})^2}$$

Two variations

- **pow2**
  - Baseline : the output for the 2 last power of two computations
- **all**
  - Baseline : the output for all the previous computations



LSR-all(4) = 8.3

LRS-pow2(4) = 7

Projection considered 2/2

Thread Impact (TI)

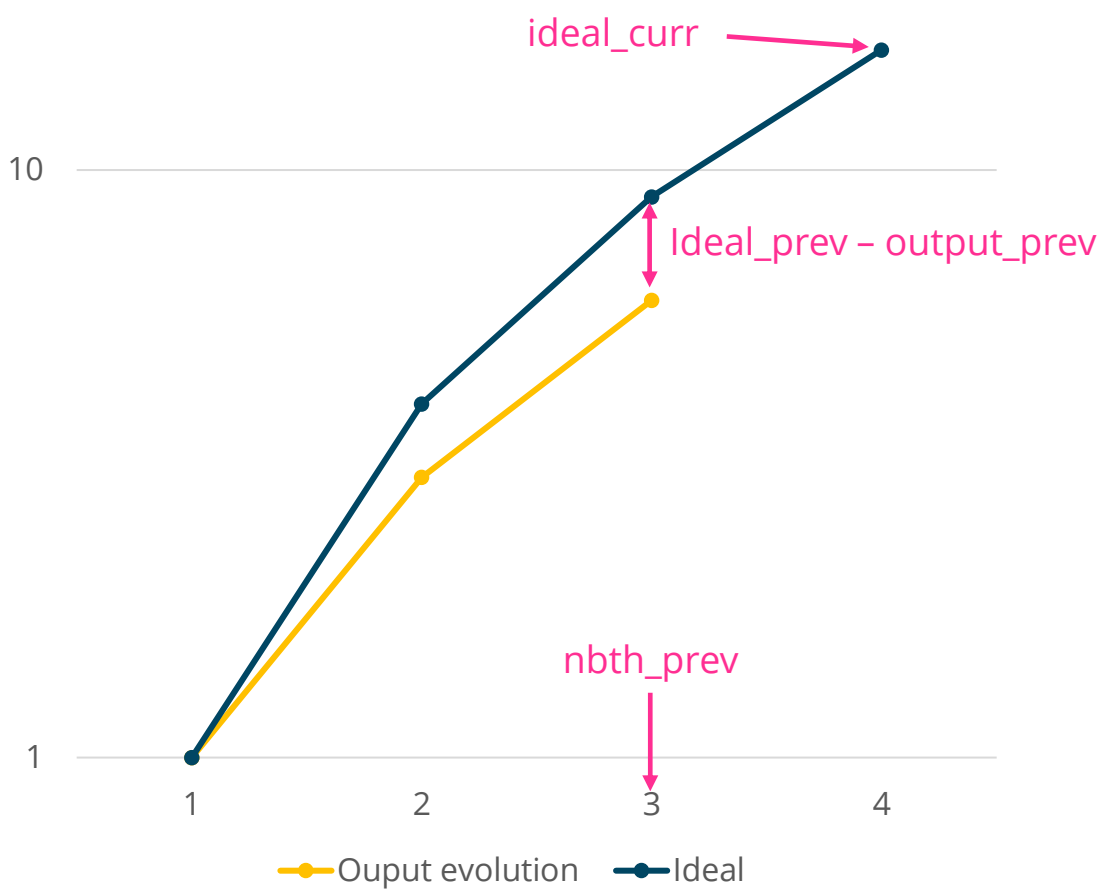
- Compute the « cost » of a thread at the previous computation
- Project it on the current computation

$$ideal_{curr} - \left(\frac{ideal_{prev} - output_{prev}}{nbth_{prev}}\right) * nbth_{curr}$$

Performance Drop (PD)

- Compute the output with the assumption that performance follows the one of another machine (here Ampere)

$$ideal_{curr} - ideal_{curr} * (1 - ratio_{ampere})$$



First case study : SPEC CPU

Benchmark description :

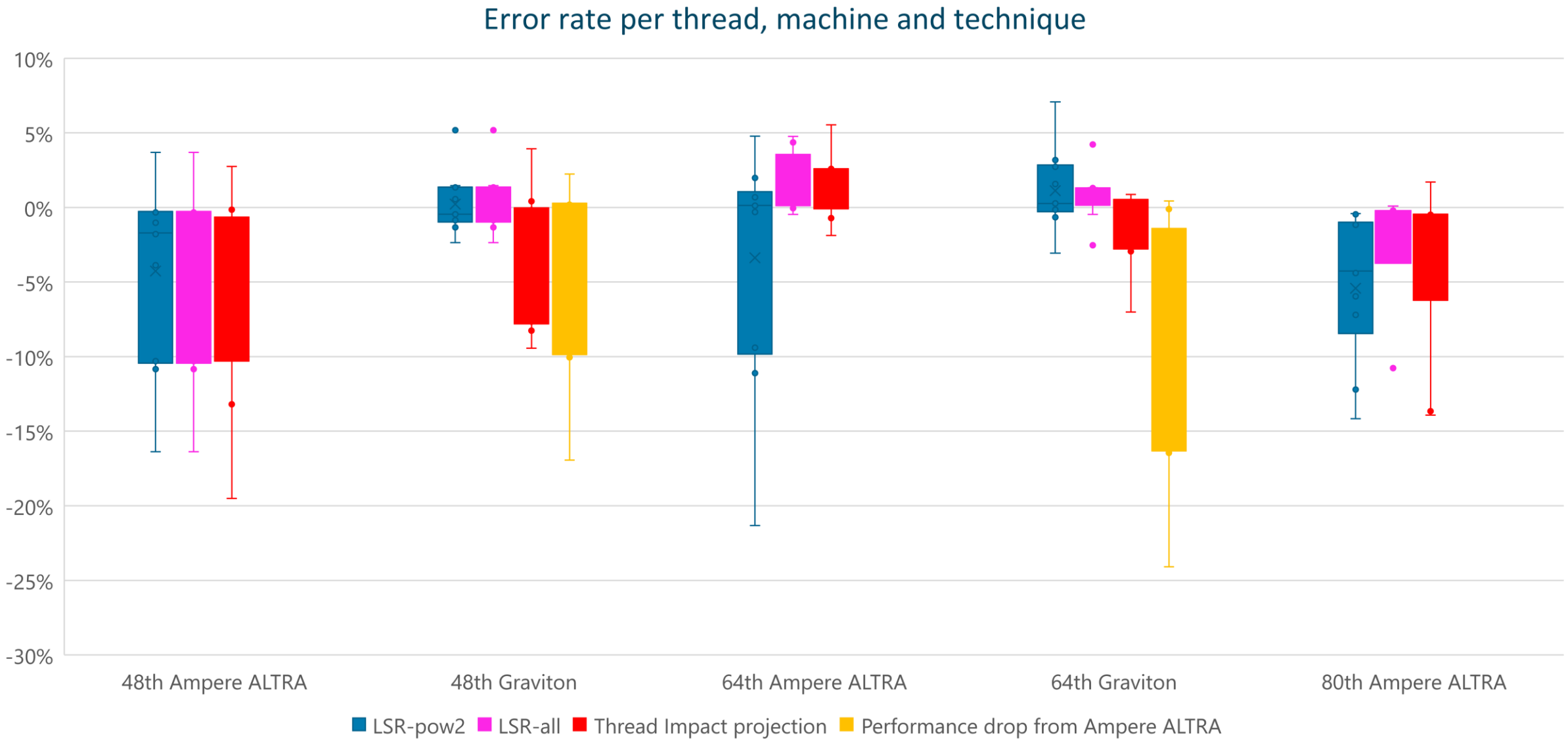
- Designed to provide performance measurements that can be used to compare compute-intensive workloads on different computer systems
- Run n copies of the underlying benchmark
- Expect no performance drop

Provide INT and FLOAT versions

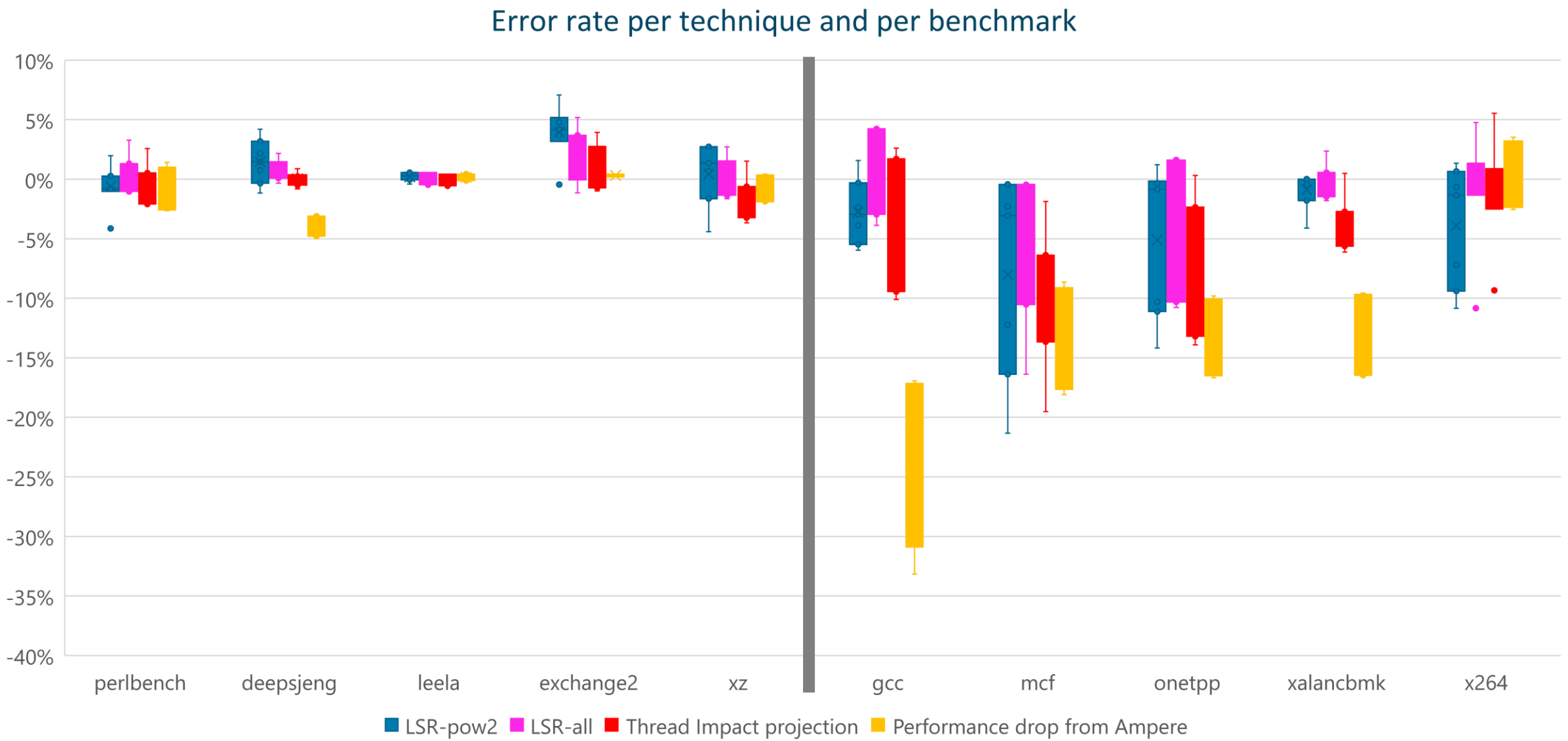
- Here only present the INT version without loss of generality
- Conclusions on the FLOAT versions are the same

Ideal output: stable whatever the number of considered threads

SPEC/CPU – Error Rate Summary 1/2



SPEC/CPU – Error Rate Summary 2/2



Partial conclusion

Walltime projections tends to be …

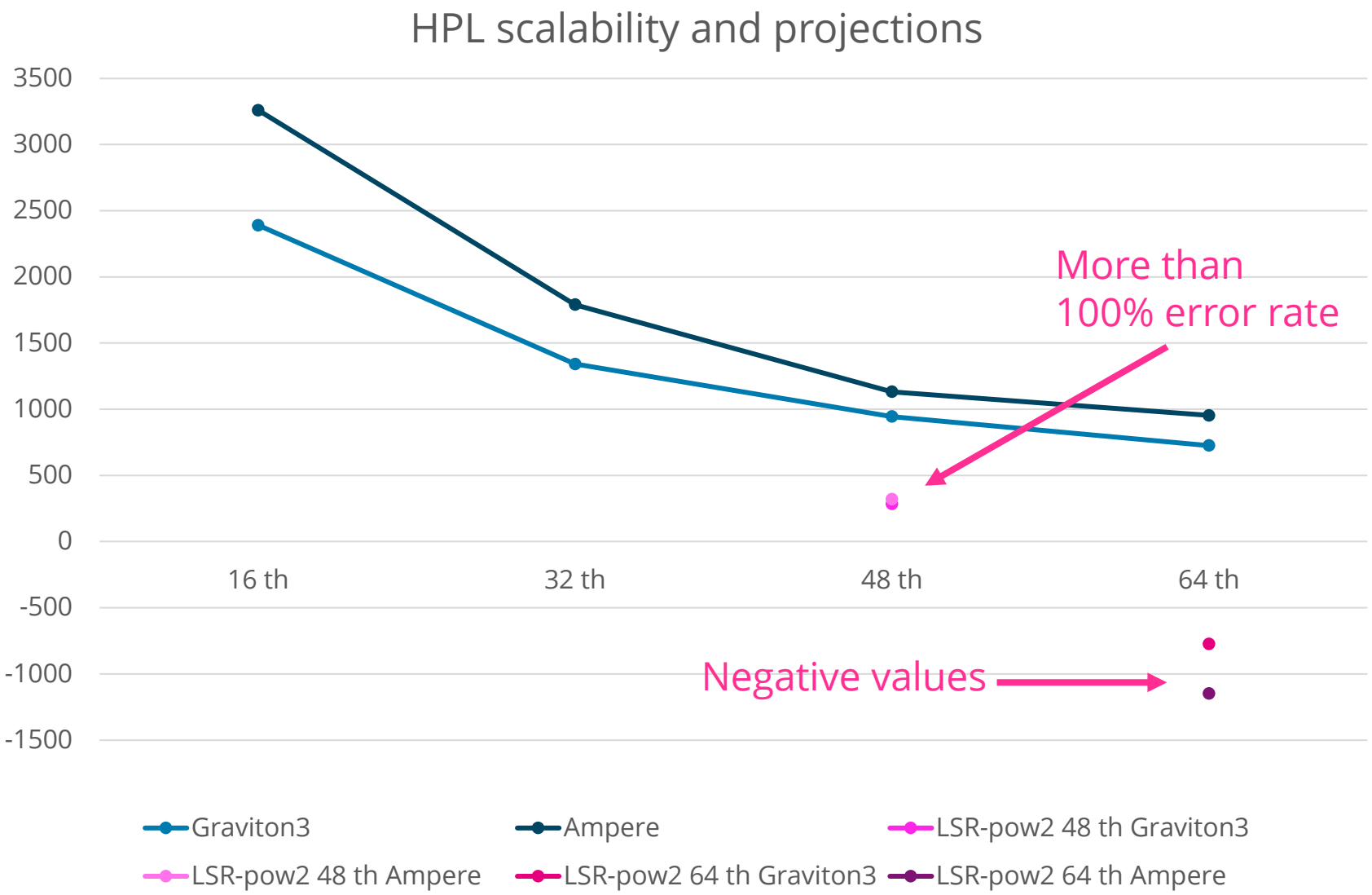
- … always better than reality

Projection type	Mean Error Rate
LSR-pow2	-2%
LSR-all	-1%
Thread Impact	-3%
Performance drop	-7%

- … in average better as the number of threads increase
- … sensitive in micro variation : projections must be done on averages
- … sensitive to bandwidth ⟹ need to anticipate correctly these projections

Non linear growth / decrease must be handled

The case of HPL – LSR-pow2 Projections



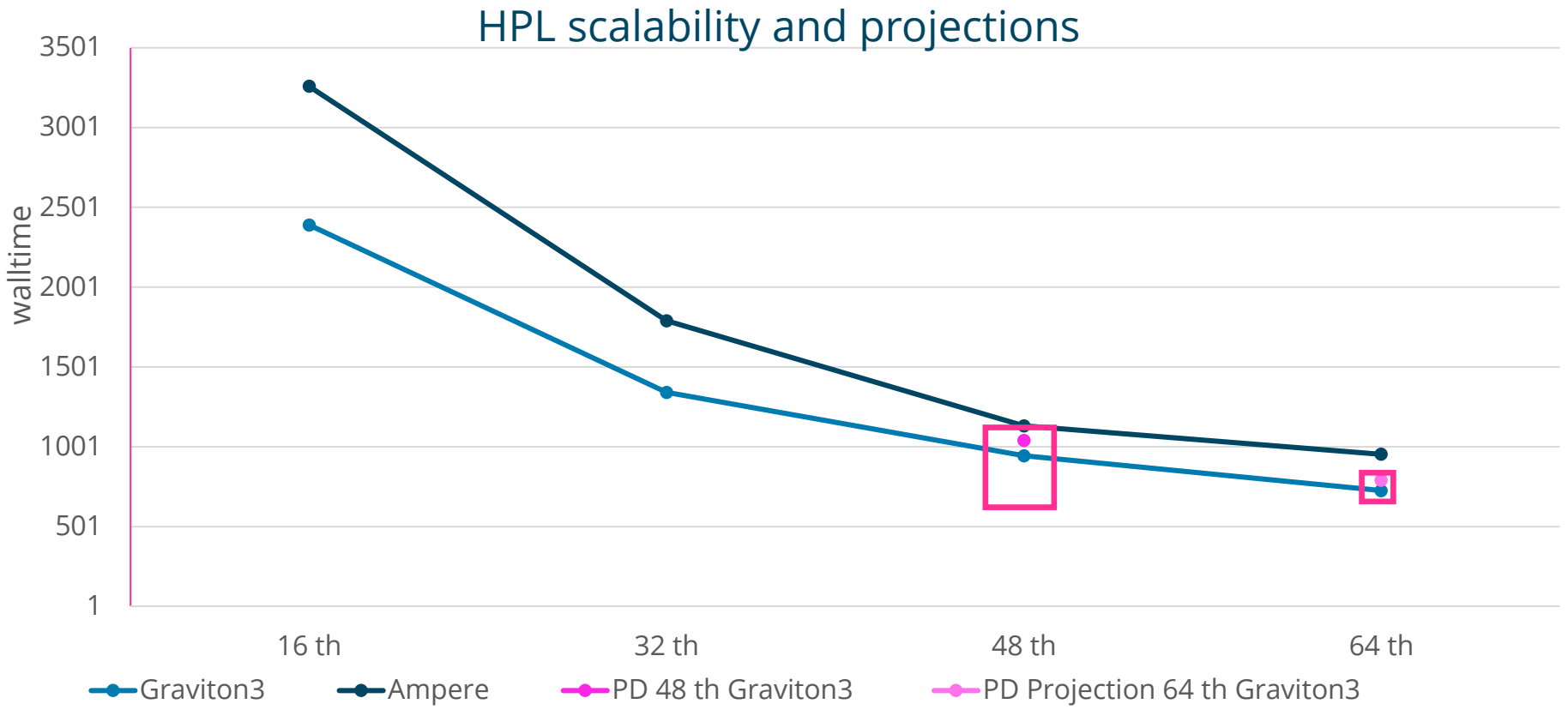
Similar results for LSR-all projections walltime is to sensitive

→

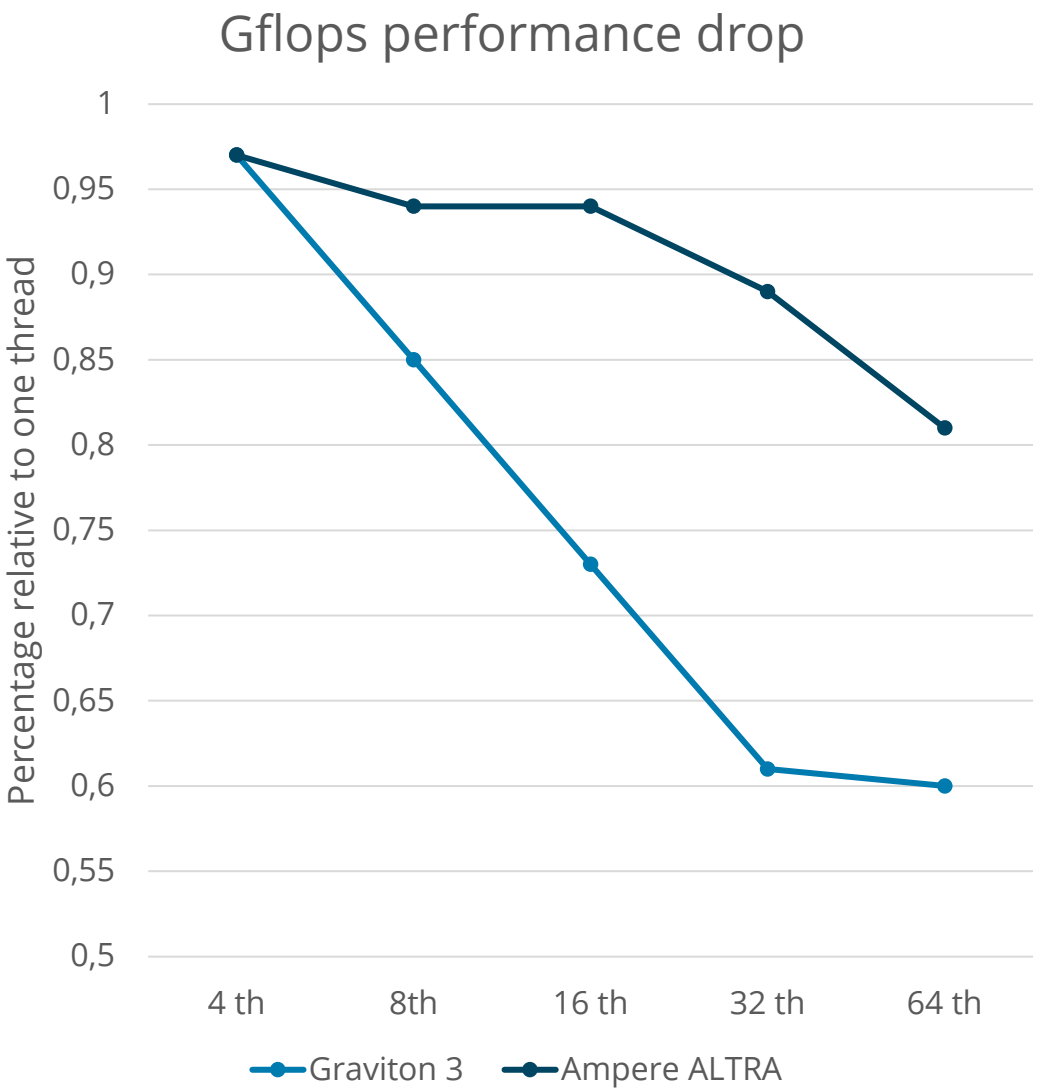
Need to pass inflexion point

The case of HPL – TI / PD

- For SPEC the **ideal** was Easy to compute, but her we don't have such metric.
- Will consider **ideal** as perfect scalability for walltime.



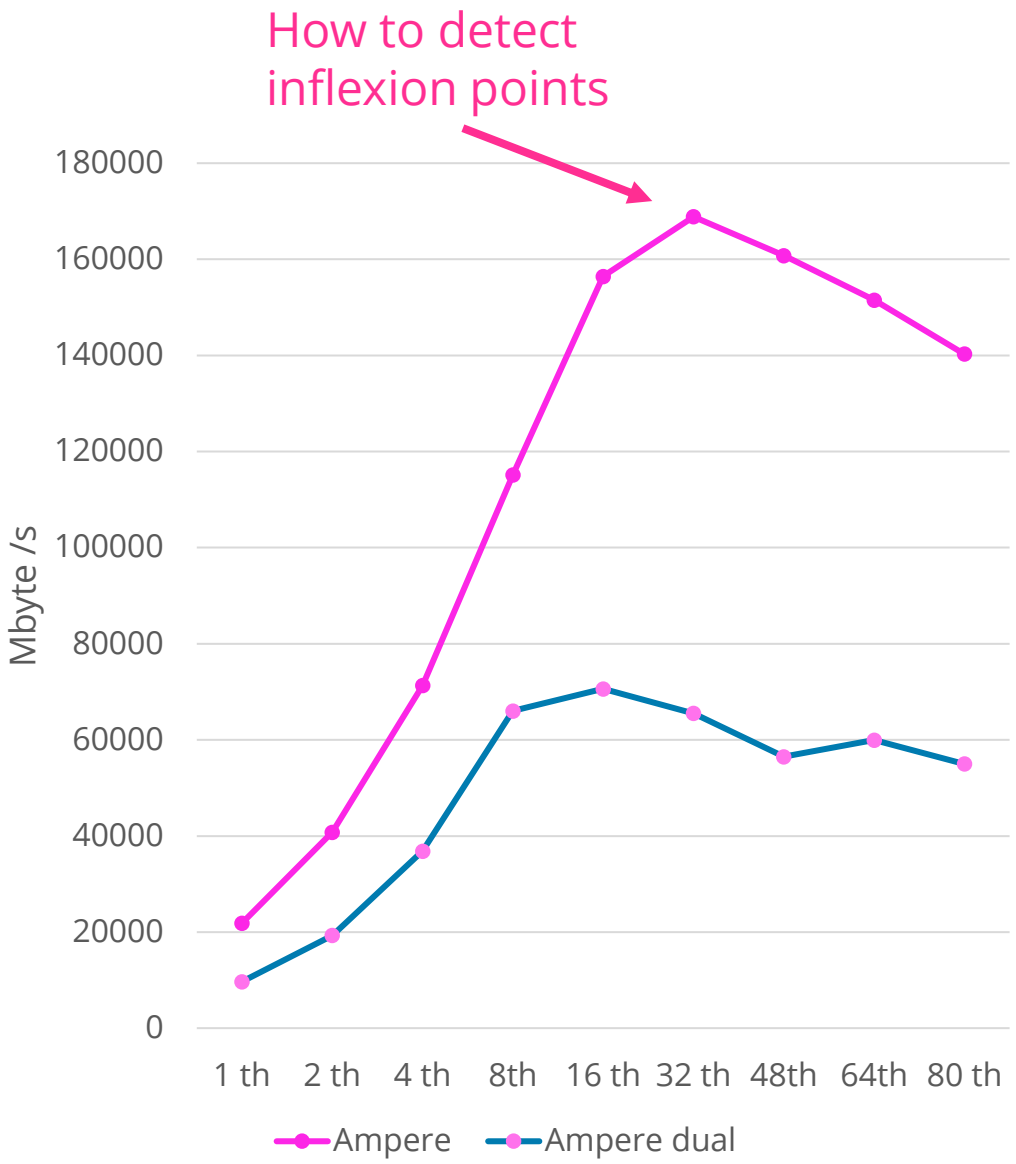
HLP – GFlops drop metric



More abstract metrics (like Gflops or ratio to peak) also work with projections

Projection type	Error Rate 48th On Graviton3	Error Rate 64th on Graviton3
LSR-pow2	2%	2%
LSR-all	5%	3%
Thread Impact	5%	3%
Performance drop	14%	-20%

STREAM - anticipate bad projections



	Metric	Error Rate 48th	Error Rate 64th	Error Rate 80th
Ampere single numa	Gflops	11%	22%	2%
	Walltime	-14%	-33%	-4%
Ampere dual numa	Gflops	7%	-6%	-4%
	Walltime	7%	-8%	4%

Since Gflops and time are strongly correlated, it is easy to detect incorrect projections

Case 48 th compared to 32 thread ;

- 16% faster **BUT** with 14% performance drop
- **INCONSITENCY**

→ Proj Gflops 0.08 rectification error rate = -6%
→ Proj walltime 0.14 rectification error rate = -3%

Conclusion

On the importance of combining multiple metrics

- To leverage erros and bypass local extremum
- To obtain one valid projection

About Projections

- LSR seem relatively stable, regarless the basline
  - But work better on other metrics than walltime
- TI/PD suitable for benchmark with inflexion points
- PD not adapted due to NOC differences
- Projections are working even better for Graviton3e

Future Work

- Tests projections with AFX64
- Run projections on HPC cluster, not single chips

About... SiPearl

SiPearl is building the world first energy-efficient HPC-dedicated microprocessor designed to work with any third-party accelerator (GPU, artificial intelligence, quantum). This new generation of microprocessors will first target EuroHPC Joint Undertaking ecosystem, which is deploying world-class supercomputing infrastructures in Europe for solving major challenges in medical research, artificial intelligence, security, energy management and climate while reducing its environmental footprint.

SiPearl is working in close collaboration with its 27 partners from the European Processor Initiative (EPI) consortium - leading names from the scientific community, supercomputing centres and industry - which are its stakeholders, future clients and end-users.

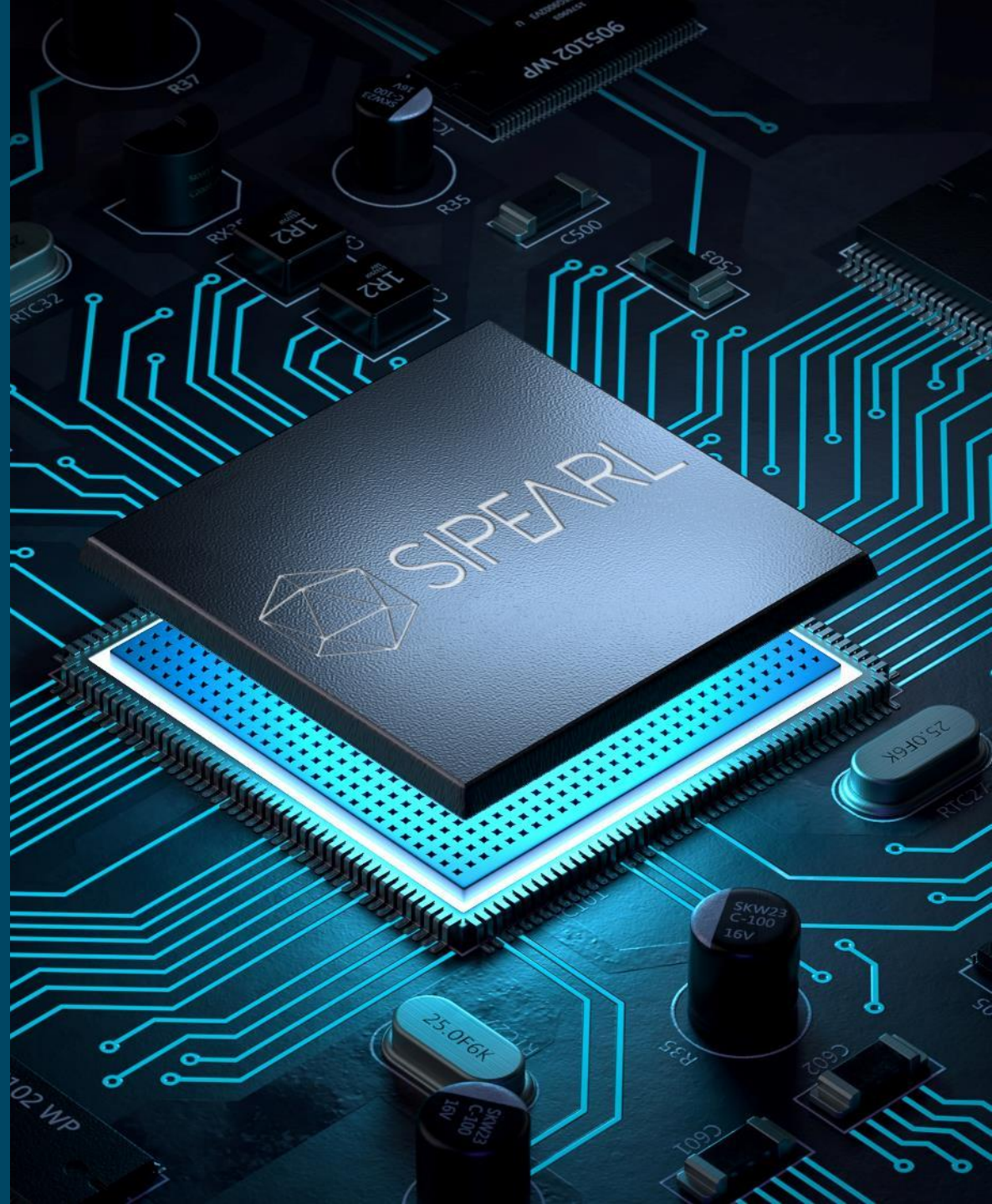
SiPearl employs 130 people in France (Maisons-Laffitte, Grenoble, Massy, Sophia Antipolis), Germany (Duisburg) and Spain (Barcelona).

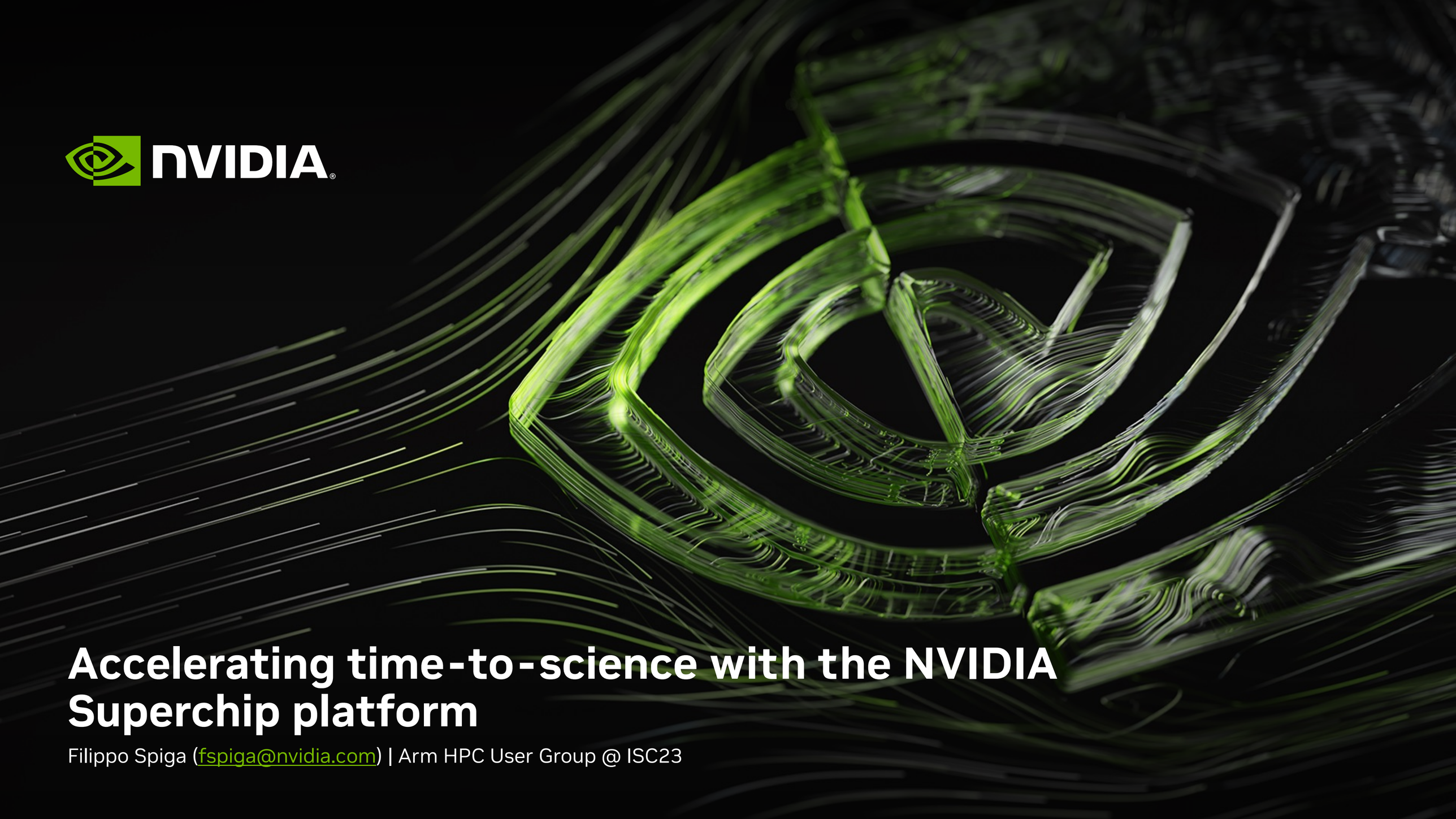
Media contact:

Marie-Anne Garigue / Grégory Bosson
+ 33 6 09 05 87 80 / + 33 6 60 75 71 61
marie-anne.garigue@sipearl.com
gregory.bosson@sipearl.com

Funded by the European Union

European Processor Initiative





Accelerating time-to-science with the NVIDIA Superchip platform

Filippo Spiga (fspiga@nvidia.com) | Arm HPC User Group @ ISC23

NVIDIA Grace for Cloud, AI and HPC Infrastructure

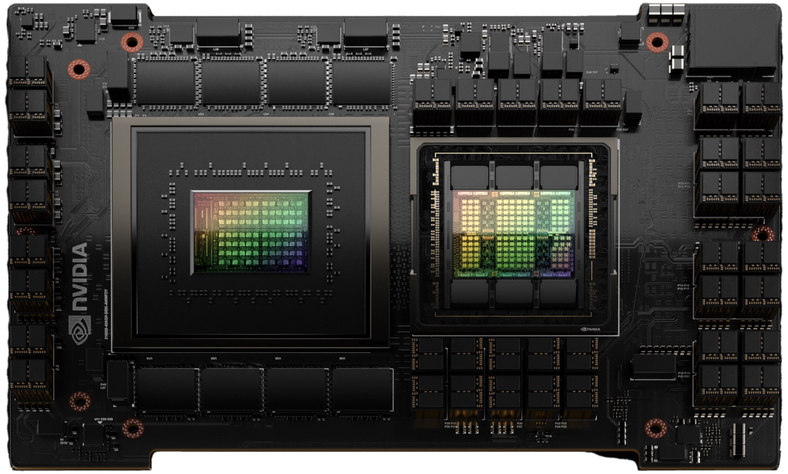
Grace CPU Superchip

CPU Computing

CPU-based applications where absolute performance, energy efficiency, and data center density matter, such as scientific computing, data analytics, enterprise and hyperscale computing applications

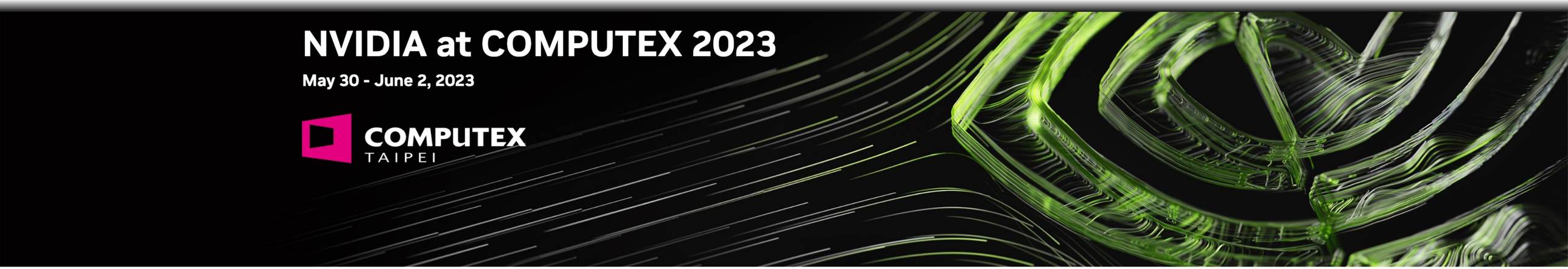
Grace Hopper Superchip

Large Scale AI & HPC



Accelerated applications where CPU performance and system memory bandwidth are critical; extreme and highly atomic collaboration between CPU & GPU contexts for flagship AI & HPC

NEW ANNOUNCEMENTS NEXT WEEK



NVIDIA Keynote at COMPUTEX 2023

Monday, May 29, 2023 at 11:00 a.m. Taipei Time

Sunday, May 28, 2023 at 8:00 p.m. Pacific Time

Join NVIDIA founder and CEO Jensen Huang at COMPUTEX 2023 for a special keynote address streaming online.

Save the Date

SCAN ME

https://www.nvidia.com/en-us/events/computex/

NVIDIA

NVIDIA Grace CPU

Building Block of the Superchip

High Performance Power Efficient Cores

72 flagship Arm Neoverse V2 Cores with
SVE2 4x128b SIMD per core

Fast On-Chip Fabric

3.2 TB/s of bisection bandwidth connects
CPU cores, NVLink-C2C, memory, and system IO

High-Bandwidth Low-Power Memory

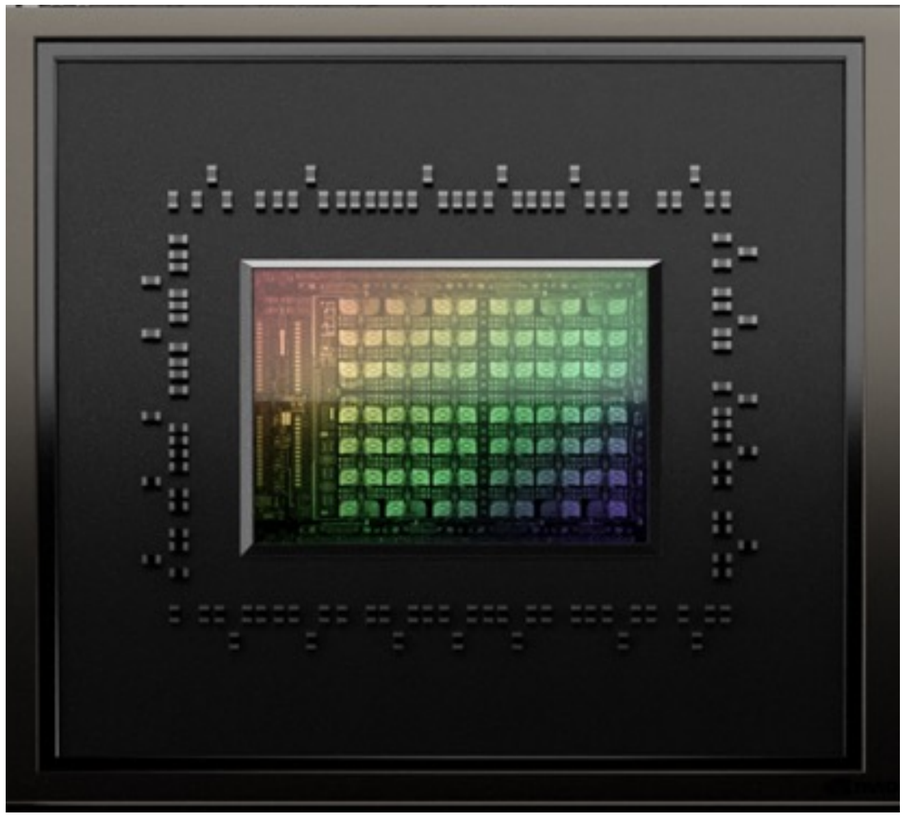
Up to 480 GB of data center enhanced LPDDR5X Memory that
delivers up to 500 GB/s of memory bandwidth

Coherent Chip-to-Chip Connections

NVLink-C2C with 900 GB/s bandwidth for coherent
connection to CPU or GPU

Industry Leading Performance Per Watt

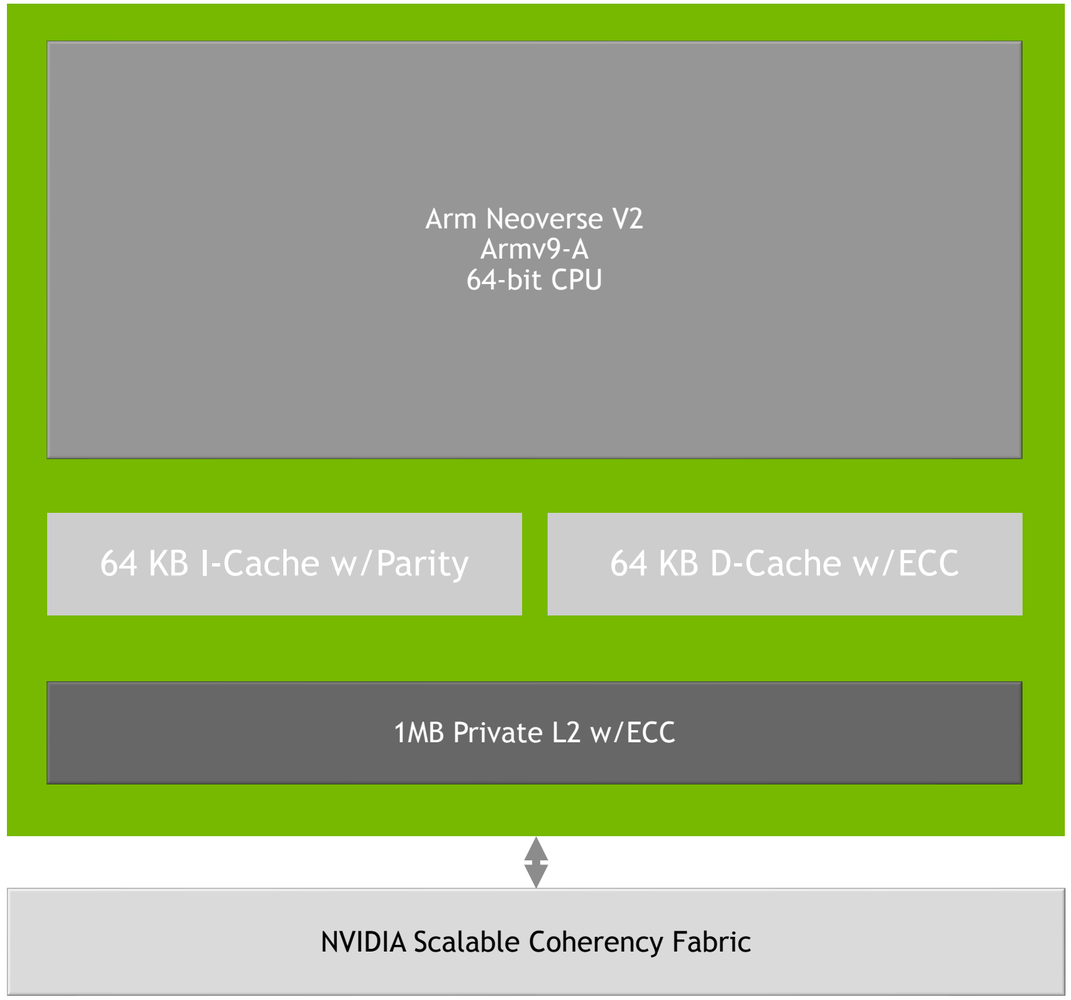
Up to 2X perf / W over today's leading servers



NVIDIA Grace core

Neoverse V2

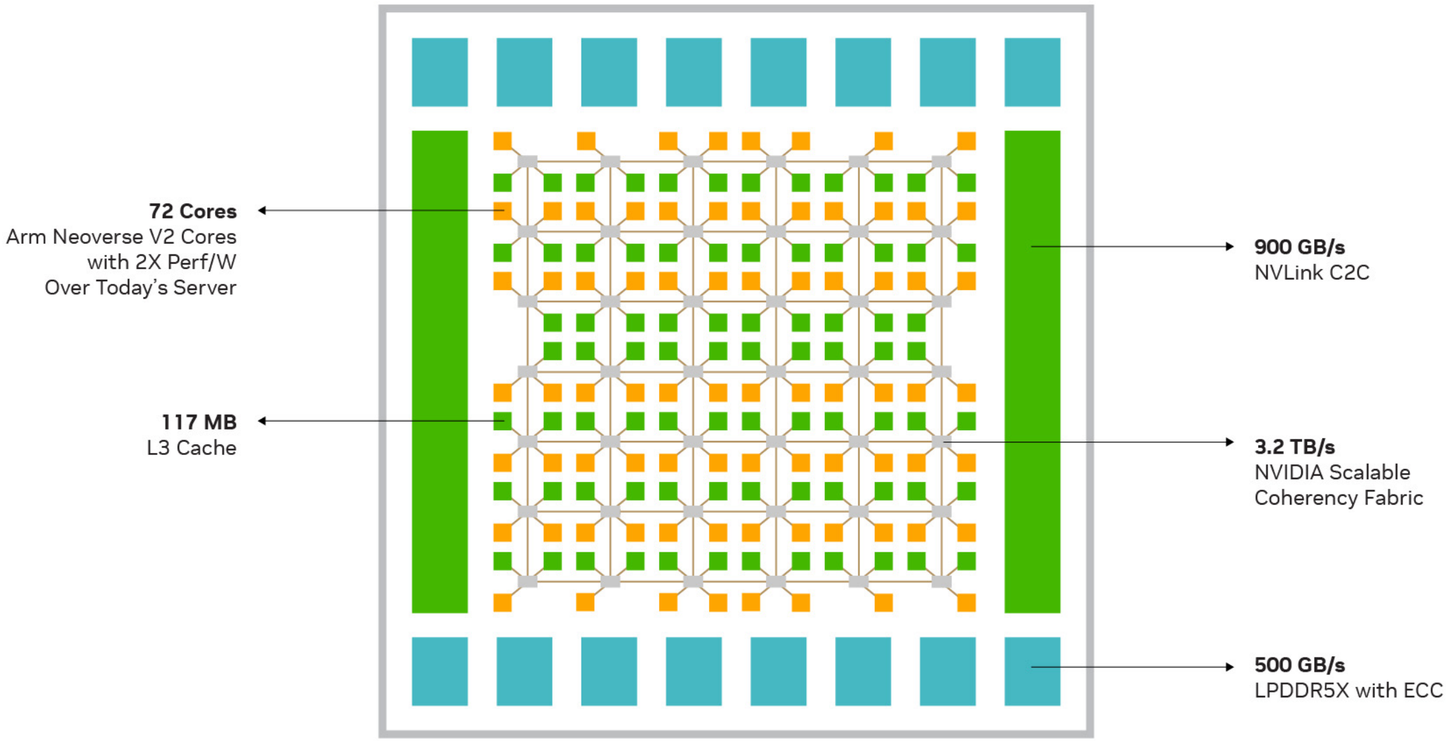
- **Arm Neoverse V2 core - Arm v9.0**
- AARCH64 at all ELs
- v9.0 scalable vector extensions
  - **Scalable Vector Extension 2 (SVE2) - 4 x 128b**
  - Scalable Vector AES (SVE_AES)
  - Scalable Vector PMULL (SVE_PMULL)
  - Scalable Vector SHA3 (SVE_SHA3)
  - Scalable Vector Pit Permutes (SVE_BitPerm)
- V9.0 debug
  - Embedded Trace Extension (ETE)
  - Trace Buffer Extension (TBE)



NVIDIA Grace is a Compute and Data Movement Architecture

NVIDIA Scalable Coherency Fabric and Distributed Cache Design

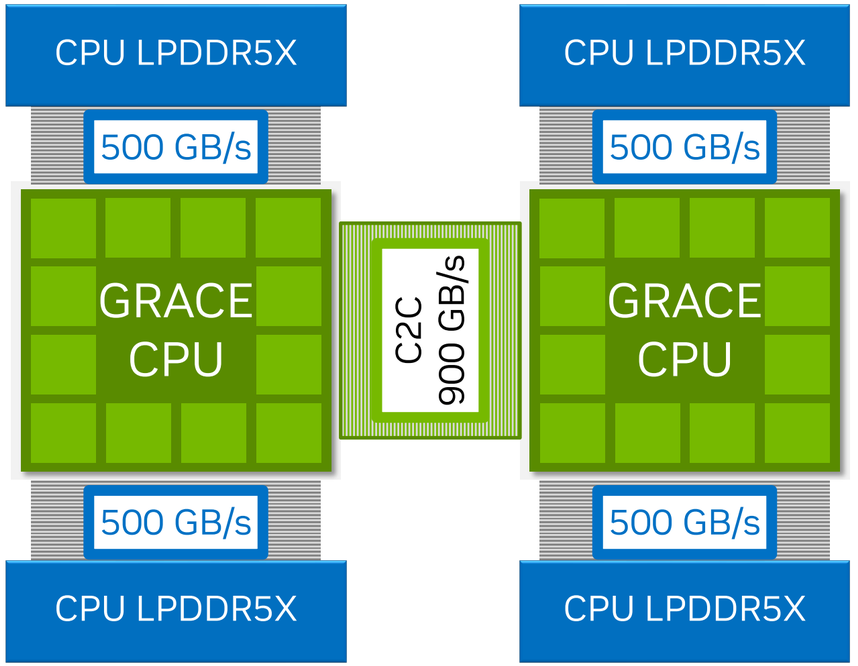
- 72 high performance Arm Neoverse V2 cores with 4x128b SVE2
- 3.2 TB/s bisection bandwidth
- 117MB of L3 cache
- Local caching of remote die memory
- Background data movement via Cache Switch Network



Low-Power High-Bandwidth Memory Subsystem

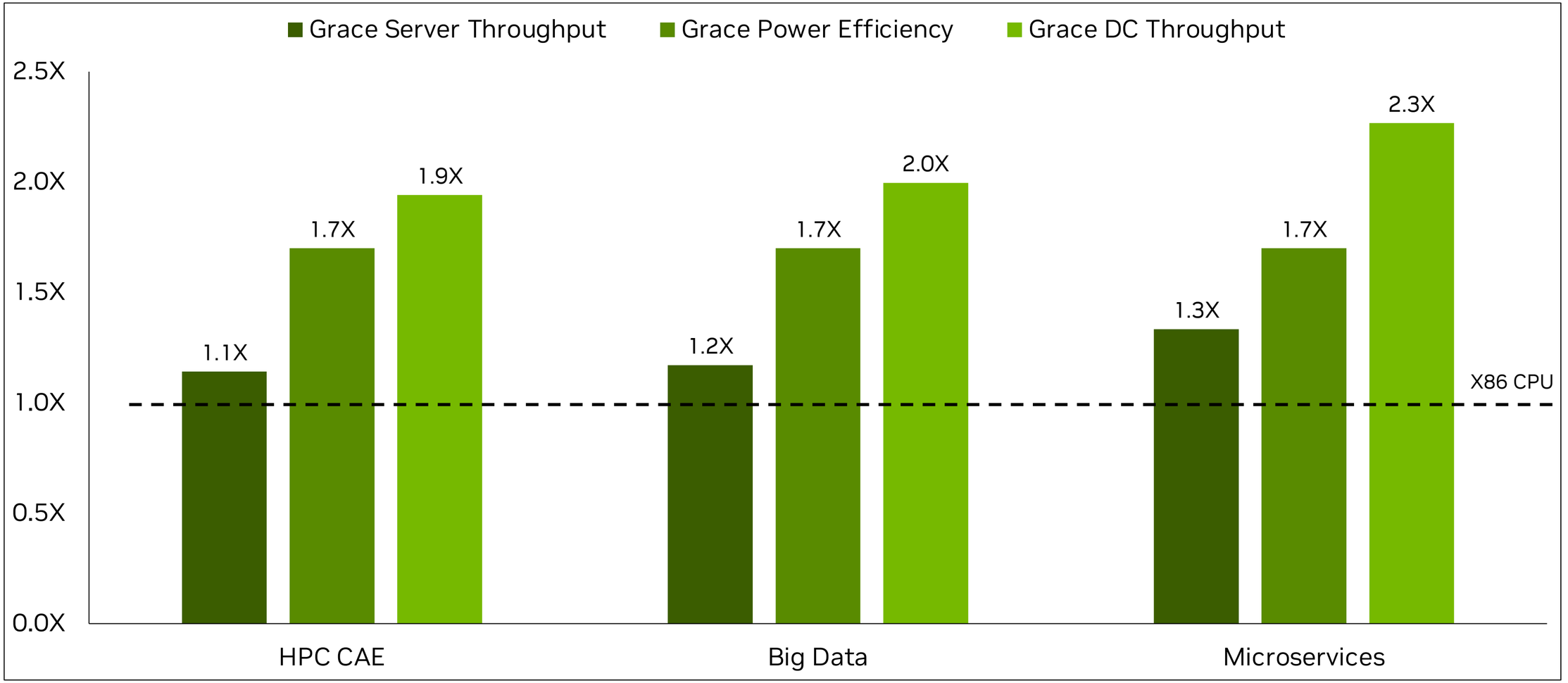
LPDDR5X Data Center Enhanced Memory

- Optimal balance between bandwidth, energy efficiency and capacity
- Up to 1TB/s of raw bidirectional BW
- 1/8th power per GB/s vs conventional DDR memory
- Similar cost / bit to conventional DDR memory
- Data Center class memory with error code correction (ECC)



Nvidia Grace CPU Delivers 2X Data Center Throughput at the Same Power

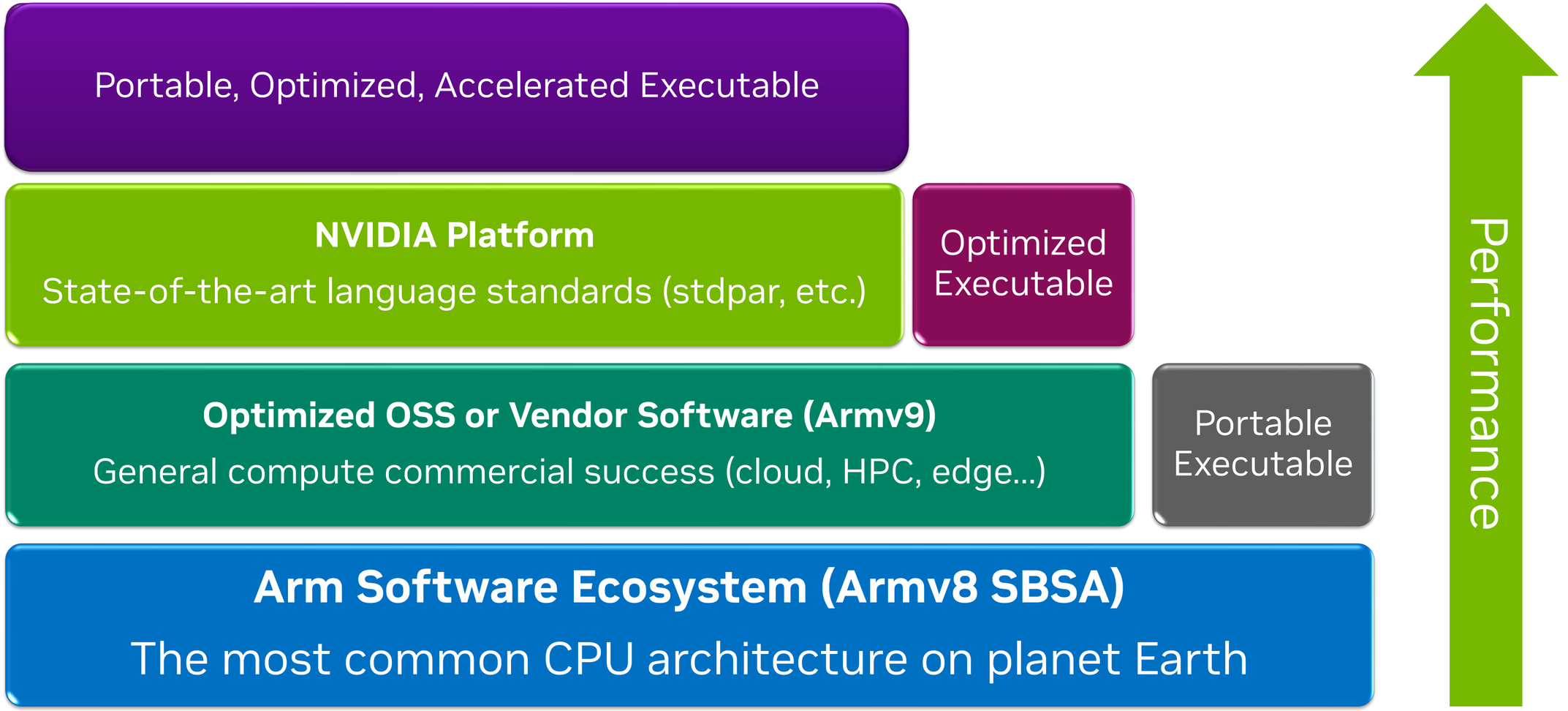
Breakthrough Performance and Efficiency



Data Center level projection of NVIDIA Grace Superchip vs x86 flagship 2-socket data center systems (112 and 192 core systems). HPC CAE: OpenFOAM (Motorbike | Small) Big Data: HiBench+K-means Spark (HiBench 7.1.1, Hadoop 3.3.3, Spark 3.3.0) and Microservices: Google Protobufs (Commit 7cd0b6fbf1643943560d8a9fe553fd206190b27f | N instances in parallel).
NVIDIA Grace Superchip performance based on engineering measurements. Results subject to change.

Grace Software Ecosystem is Built on Standards

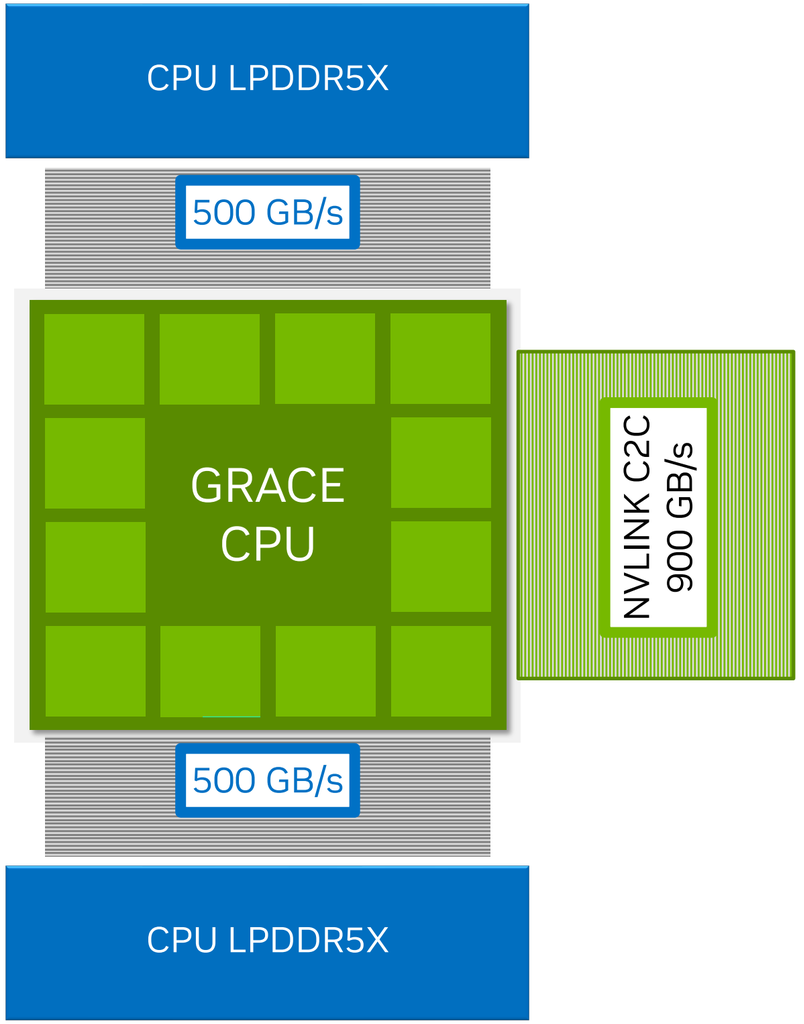
Grace brings the full NVIDIA software stack to Arm.



NVLINK-C2C

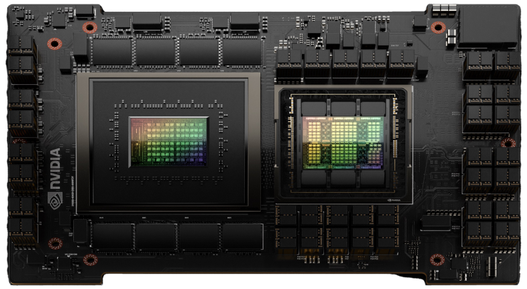
High Speed Chip to Chip Interconnect

- Creates Grace Hopper and Grace Superchips
- Removes the typical cross-socket bottlenecks
- **Up to 900GB/s of raw bidirectional BW**
  - Same BW as GPU to GPU NVLINK on Hopper
- **Low power interface - 1.3 pJ/bit**
  - More than 5x more power efficient than PCIe
- **Enables coherency** for both Grace and Grace Hopper superchips



GRACE HOPPER SUPERCHIP

The breakthrough accelerated CPU for Large-Scale AI and HPC applications



Grace CPU + H100 GPU

72 Arm Neoverse V2 Cores with SVE2 4x128b
Transformer Engine and ~4PFLOPS of FP8

Fast NVLink-C2C Connection

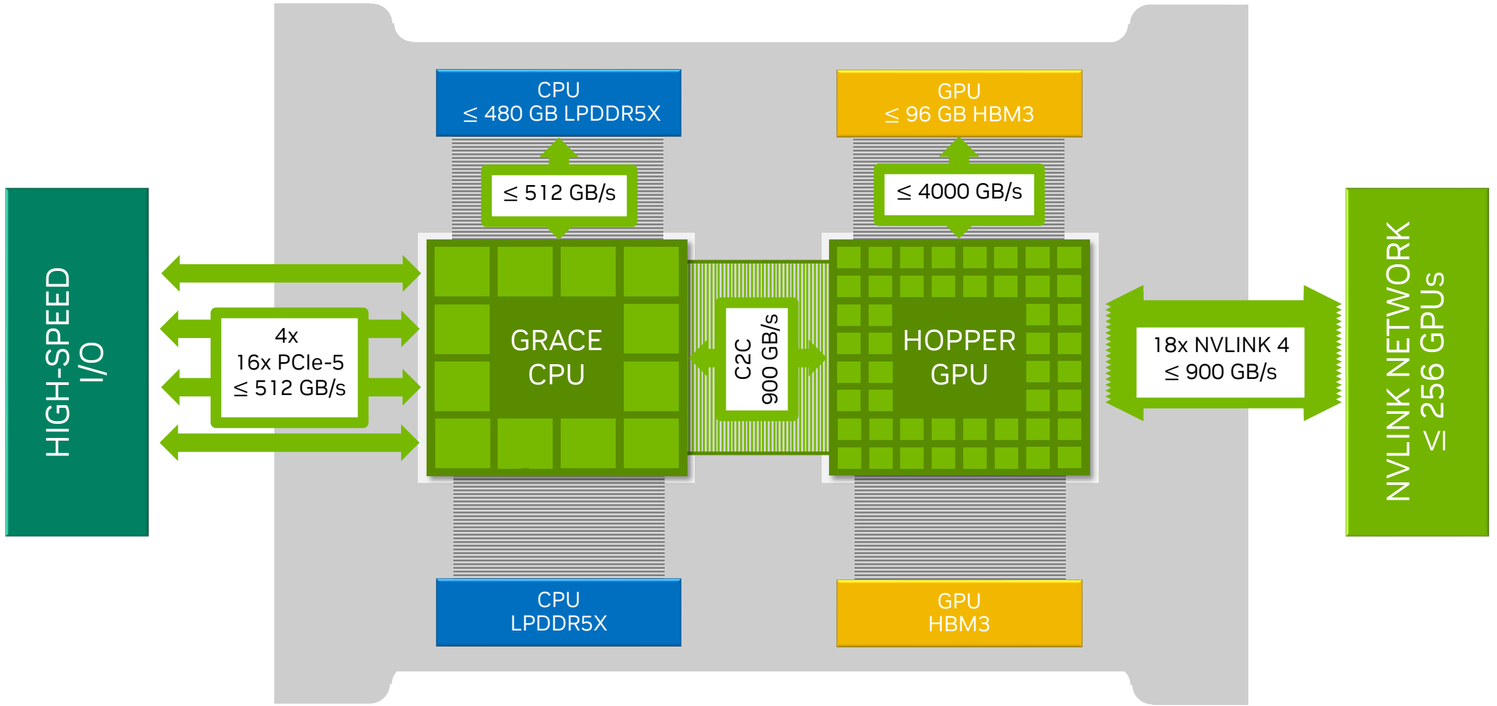
900GB/s bi-directional bandwidth CPU to GPU
7X faster than PCIe Gen 5

~600GB of Fast Access Memory

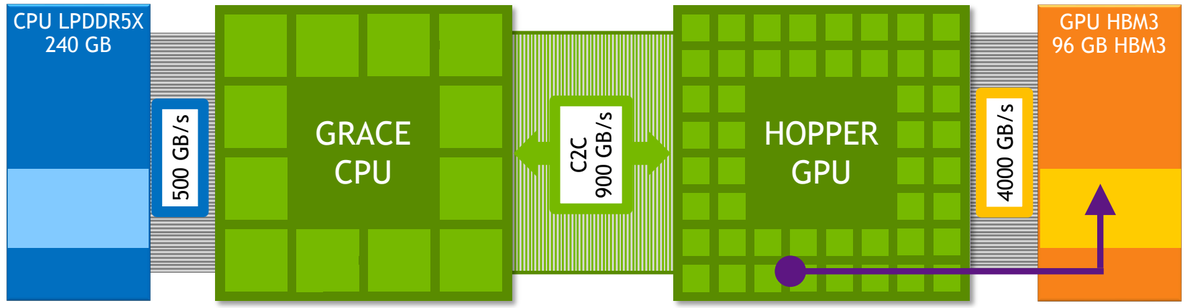
Up to 96GB HBM3, 4TB/s bandwidth
Up to 480GB LPDDR5X, 512GB/s bandwidth

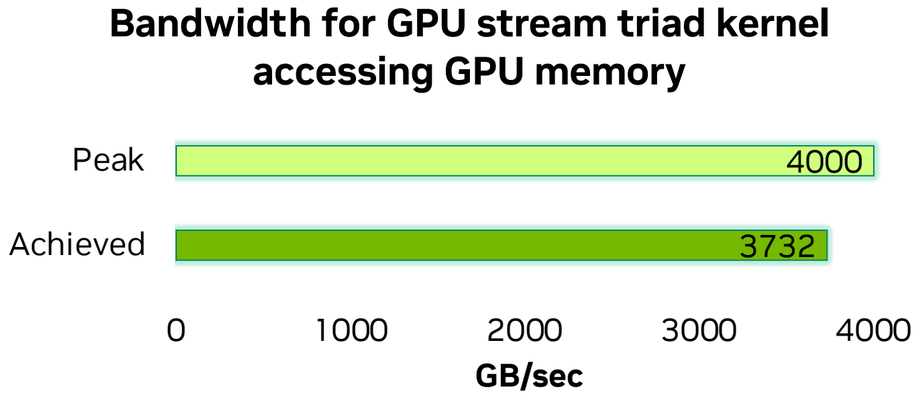
Full NVIDIA Compute Stack

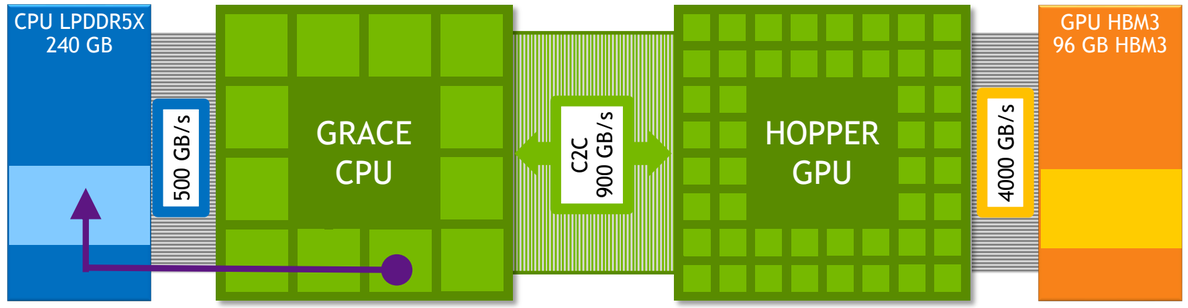
HPC, AI, Omniverse

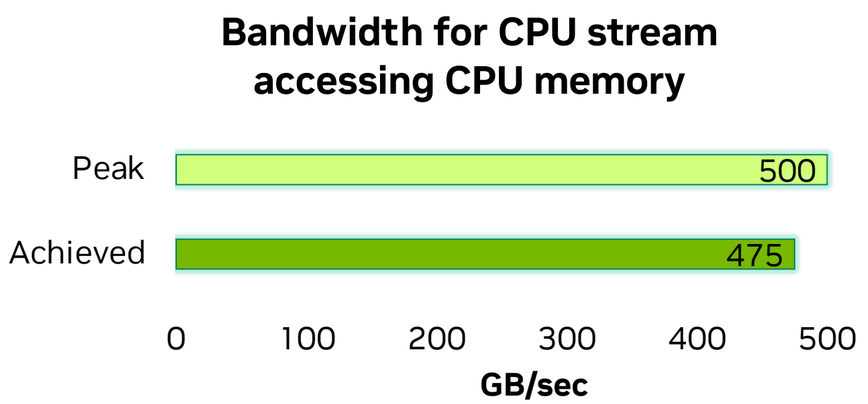


High Bandwidth Memory Access & Automatic Data Migration

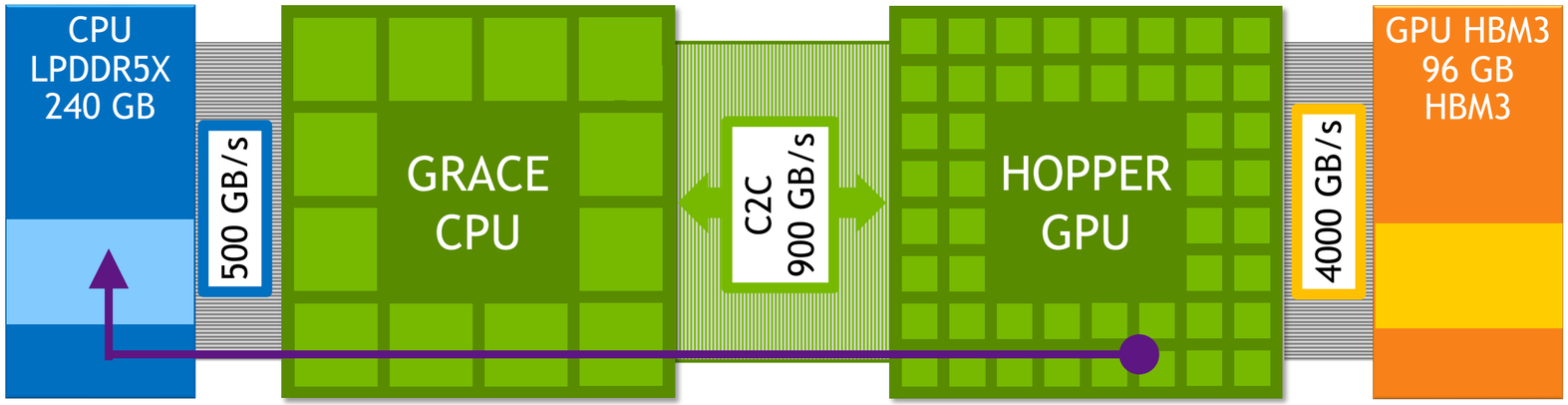


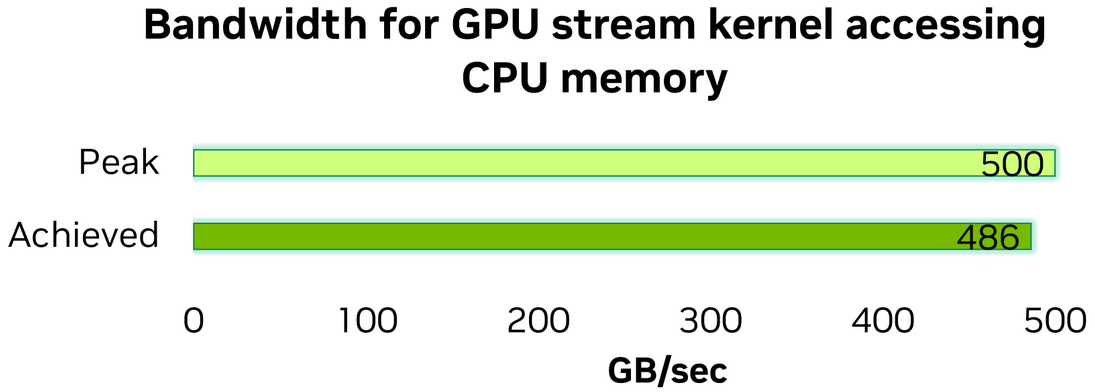






High Bandwidth Memory Access & Automatic Data Migration





These models work best with a hardware supported shared address space

GRAMMING THE NVIDIA PLATFORM

Unmatched Developer Flexibility

On PCs

On Prem

At the Edge

In the Public Cloud

Parallelism in Standard Languages

python

Directives For Existing Apps

OpenACC

OpenMP

Peak Performance

NVIDIA CUDA

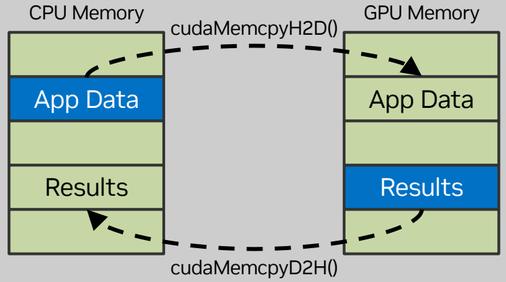
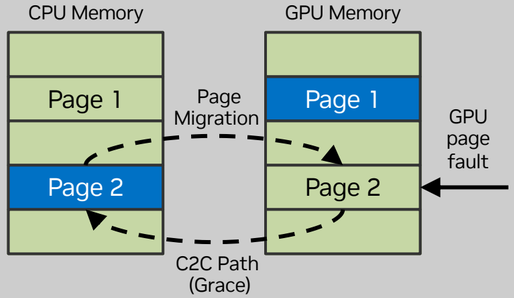
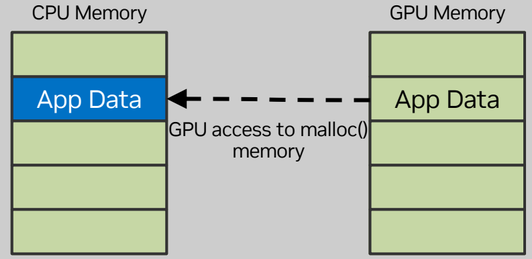
C++ | Fortran | Python

Acceleration Libraries

(AI, Data Analytics, Algebra, Quantum, Communication)

ADVANTAGES OF THE GRACE HOPPER MEMORY MODEL

Full CUDA support with additional Grace memory extensions

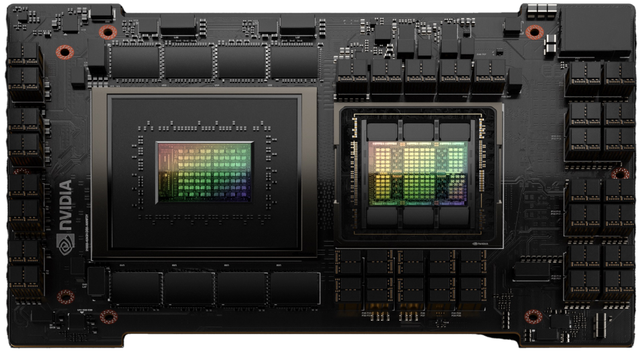
	Explicit Copy	Managed Memory	System Allocated
	Application explicitly moves data between CPU & GPU as needed 	CPU and GPU can access memory on-demand and data migrated locally for higher BW access 	GPU can access memory allocated from `malloc()`, `mmap()`, etc. 
HGX	~60 GB/s PCIe Gen5 transfers (H2D/D2H)	Requires migration to GPU	Access possible with explicit call to `cudaHostRegister()` at PCIe speeds Requires HMM patch in Linux Kernel
G + H	7x faster transfers, up to 450 GB/s (NVLink C2C)	Migrations not required and faster migrations when they happen at NVLink C2C speed	`cudaHostRegister()` not needed; access at NVLink C2C speeds

Grace Hopper HPC Platform

Unified Memory and Cache Coherence for next gen HPC performance

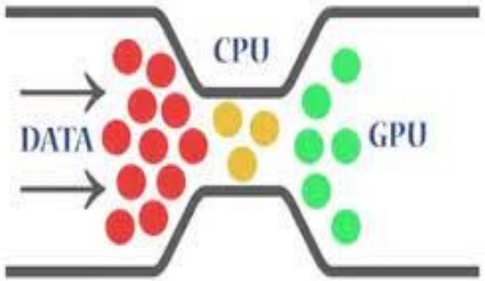
Partially GPU Accelerated Apps

Big performance gains with no code changes



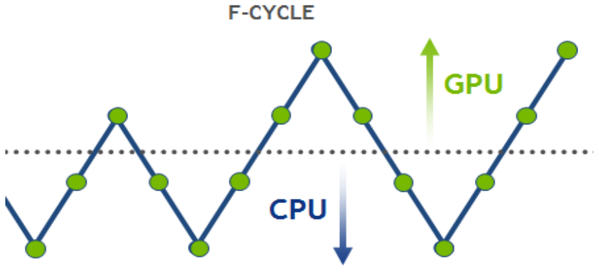
No More PCIe Bottleneck

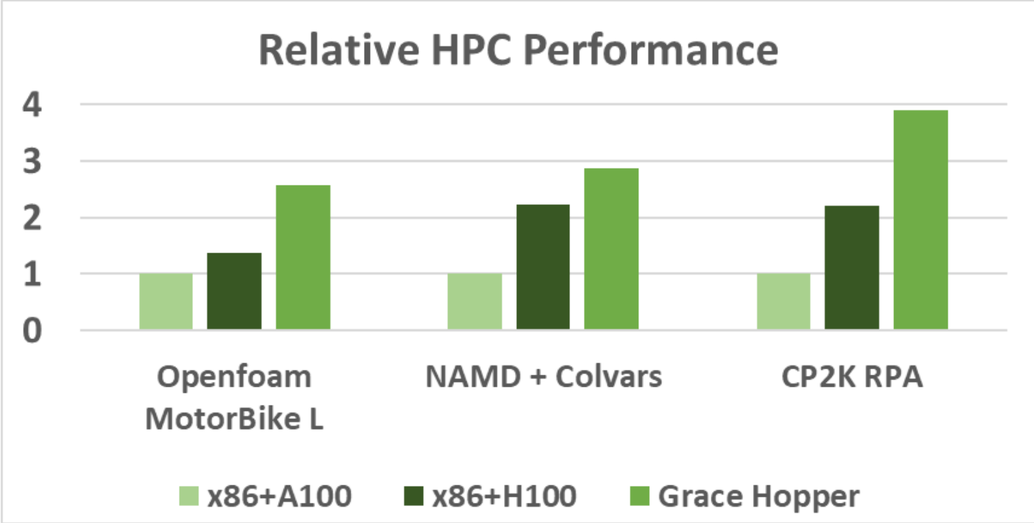
NVLink-C2C is 7X PCIe BW



CPU & GPU Cache Coherence

Incremental code changes yield big gains





Fast Access Memory

600GB

Memory Bandwidth

4TB/s

HPC: Preliminary results comparing DGX A100, DGX H100 and Grace Hopper systems.
AI: Simulation results comparing DGX H100 and Grace Hopper systems. GNN: GraphSAGE model for an augmented ogbn- products dataset of 626M nodes and 31B edges, DRLM: Deep Learning Recommendation Model (DLRM) 27TB model, NLP: fine-tuning a GPT-175B model

Application on Accelerated Systems

Partially GPU Accelerated

As GPUs become faster applications become increasingly limited by non-GPU factors

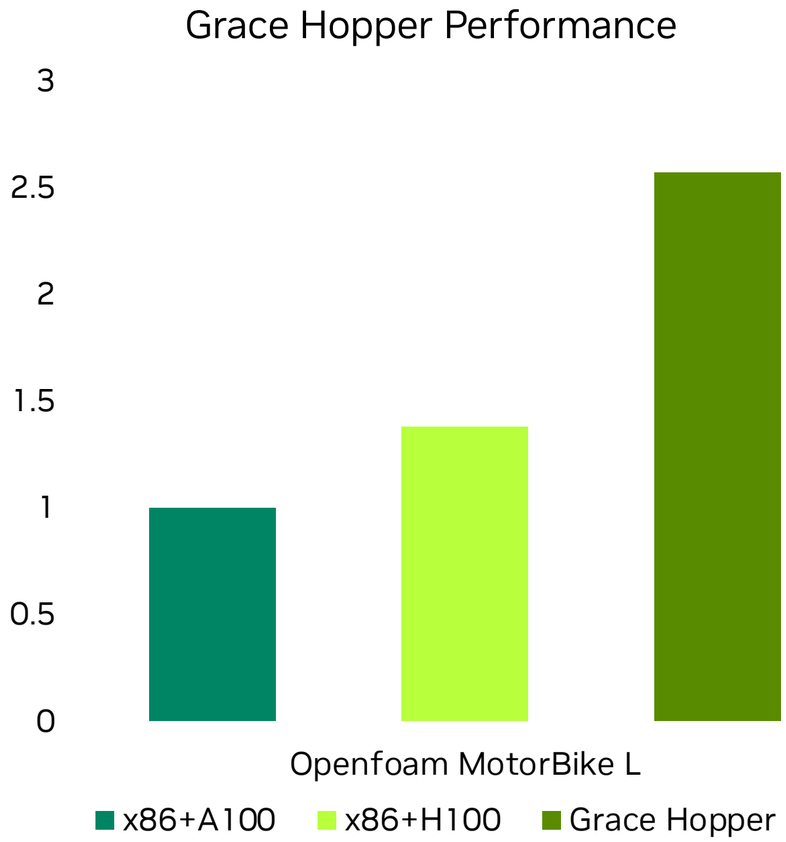
e.g. mostly data transfer (PCIe) limited

- mostly CPU limited

OpenFoam

Partially GPU Accelerated – mostly CPU limited

- Computational fluid dynamics (CFD) toolbox developed by OpenCFD
  - Popular in automotive and other engineering sectors
  - Highly configurable fluid flow solvers with turbulence / heat transfer / etc.
  - Leverage GPU accelerated AMGX linear solvers
- HPC motorbike problem (Large)
  - Around 30% of CPU-only execution is spent in linear solves
- Performance on Grace Hopper
  - High CPU and GPU memory bandwidth improve compute performance
  - C2C bandwidth minimises the cost of migrating CPU matrix data



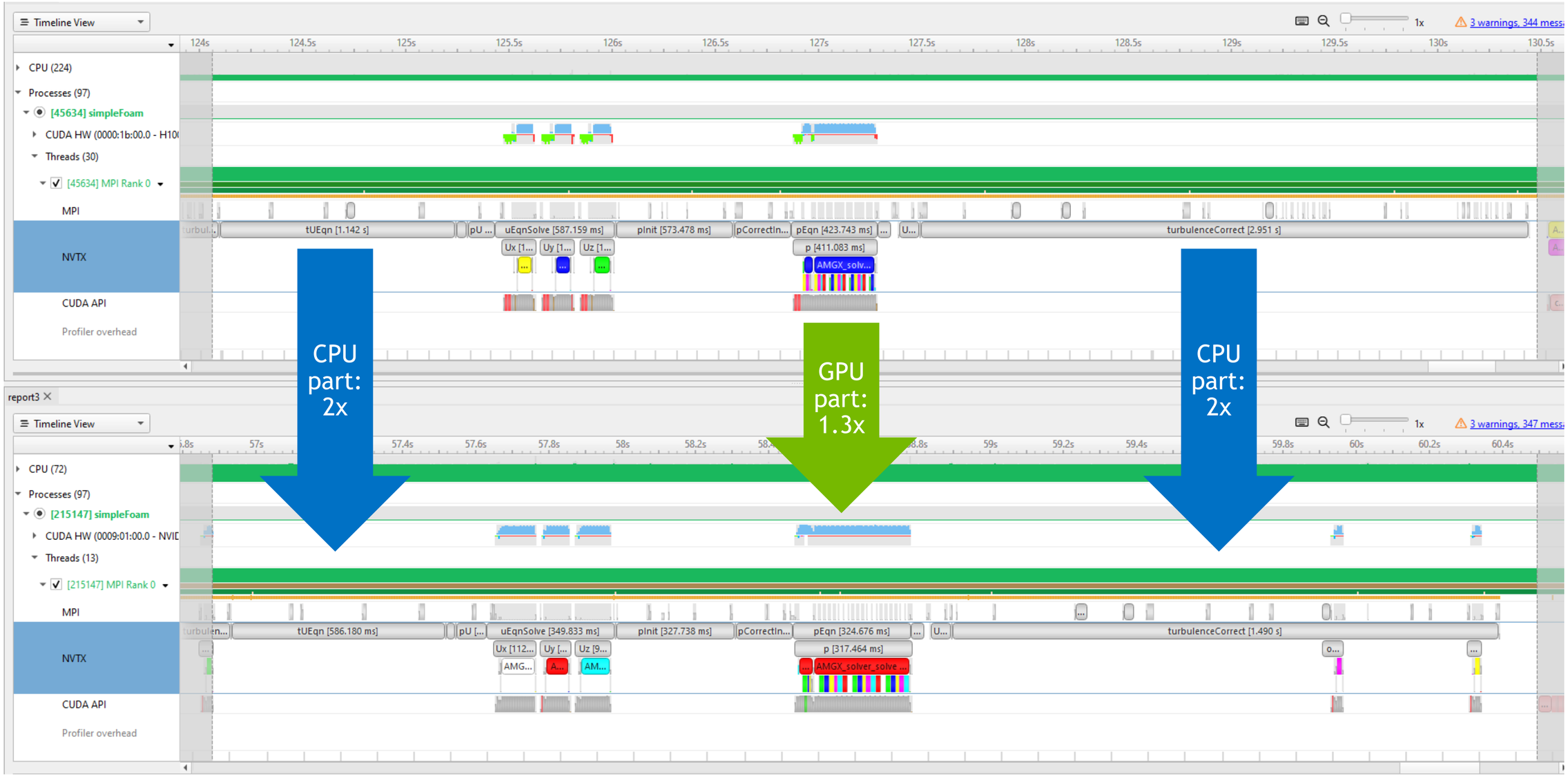
~35M cells benchmark designed by OpenFOAM HPC technical committee

OpenFoam

Nsight Systems Profile

x86 + H100 (H100 80GB HBM3)

Grace Hopper (H100 96GB HBM3)



NVIDIA

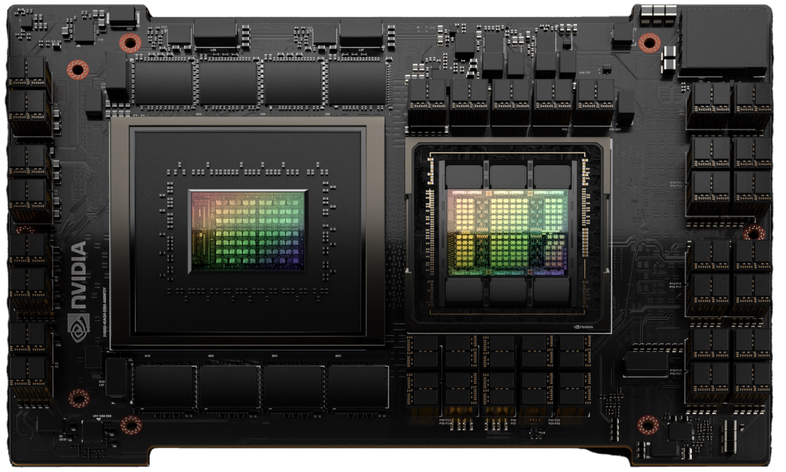
Further Resources for Grace CPU and Grace Hopper

Grace CPU Superchip

- Grace CPU Superchip Architecture Whitepaper
- Grace CPU Architecture In-Depth Blog
- Grace CPU Superchip Data Sheet
- Grace CPU Energy Efficiency Blog
- A Demonstration of AI and HPC Applications for NVIDIA Grace CPU [S51880]

Grace Hopper Superchip

- Grace Hopper Superchip Architecture Whitepaper
- Grace Hopper Architecture In-Depth Blog
- Grace Hopper Superchip Architecture Data Sheet
- Grace Hopper Recommender System Blog
- Programming Model and Applications for the Grace Hopper Superchip [S51120]




NVIDIA

An Analysis of Arm Graviton Systems Using Linaro Performance Reports

Beau.Paisley@linaro.org

Linaro Forge

An interoperable toolkit for debugging and profiling

The de-facto standard for HPC development

- Most widely-used debugging and profiling suite in HPC
- Fully supported by Linaro on Intel, AMD, Arm, IBM Power, Nvidia, AMD GPUs, etc.

State-of-the art debugging and profiling capabilities

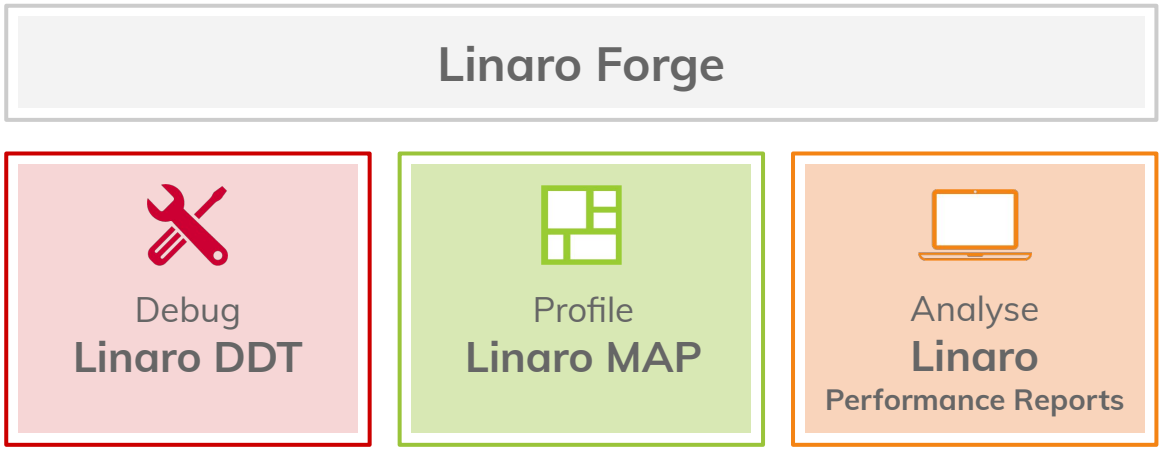
- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflopic applications)

Easy to use by everyone

- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

HPC Development Solutions from Linaro

Best in class commercially supported tools for Linux and high-performance computing (HPC)



Performance Engineering for any architecture, at any scale

9 Step Guide

Optimizing high performance applications

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster.

This pragmatic, 9 Step best practice guide, will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.

Bugs

- Correct application

Analyze before you optimize

- Measure all performance aspects.
  You can't fix what you can't see.
- Prefer real workloads over artificial tests.

Cores

- Discover synchronization overhead and core utilization
- Synchronization-heavy code and implicit barriers are revealed

I/O

- Discover lines of code spending a long time in I/O.
- Trace and debug slow access patterns.

Vectorization

- Understand numerical intensity and vectorization level.
- Hot loops, unvectorized code and GPU performance reveleaed

Workloads

- Detect issues with balance.
- Slow communication calls and processes.
  Dive into partitioning code.

Memory

- Reveal lines of code bottlenecked by memory access times.
- Trace allocation and use of hot data structure

Communication

- Track communication performance.
- Discover which communication calls are slow and why.

Verification

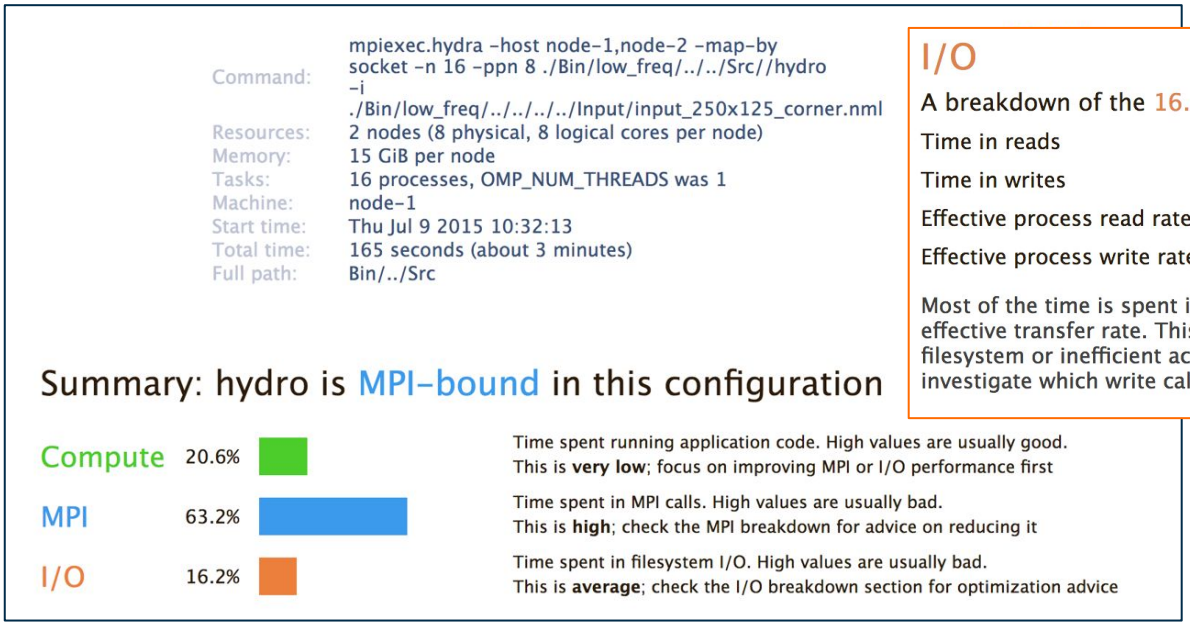
- Validate corrections and optimal performance

Key : Linaro Forge
Linaro Performance Reports

LinaroForge

Linaro Performance Reports

A high-level view of application performance with “plain English” insights



Command:	mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/low_freq/../../Src//hydro -i ./Bin/low_freq/../../../../Input/input_250x125_corner.nml
Resources:	2 nodes (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	16 processes, OMP_NUM_THREADS was 1
Machine:	node-1
Start time:	Thu Jul 9 2015 10:32:13
Total time:	165 seconds (about 3 minutes)
Full path:	Bin/../Src

Summary: hydro is MPI-bound in this configuration

Compute	20.6%	Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first
MPI	63.2%	Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it
I/O	16.2%	Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

I/O

A breakdown of the 16.2% I/O time:

Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	1.38 MB/s

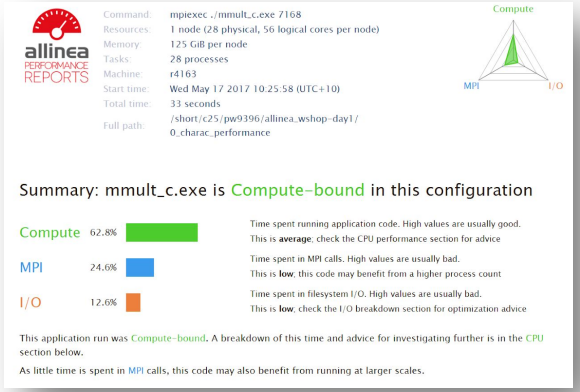
Most of the time is spent in write operations with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

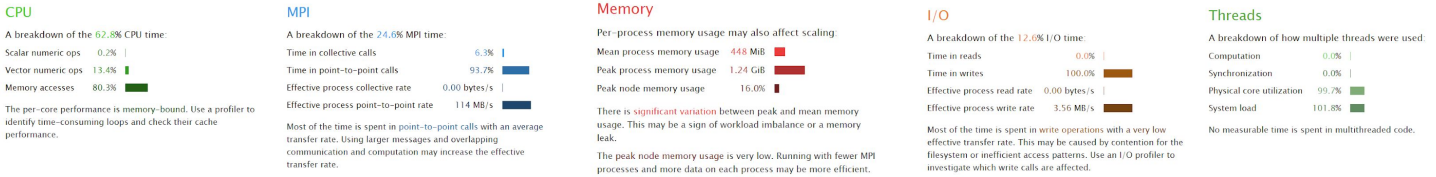
Understand application behaviour now

Set a reference for future work

- Choose a representative test cases with know results
- Analyse performance on existing hardware
- with Linaro Performance Reports
- Test scaling and note compiler flags

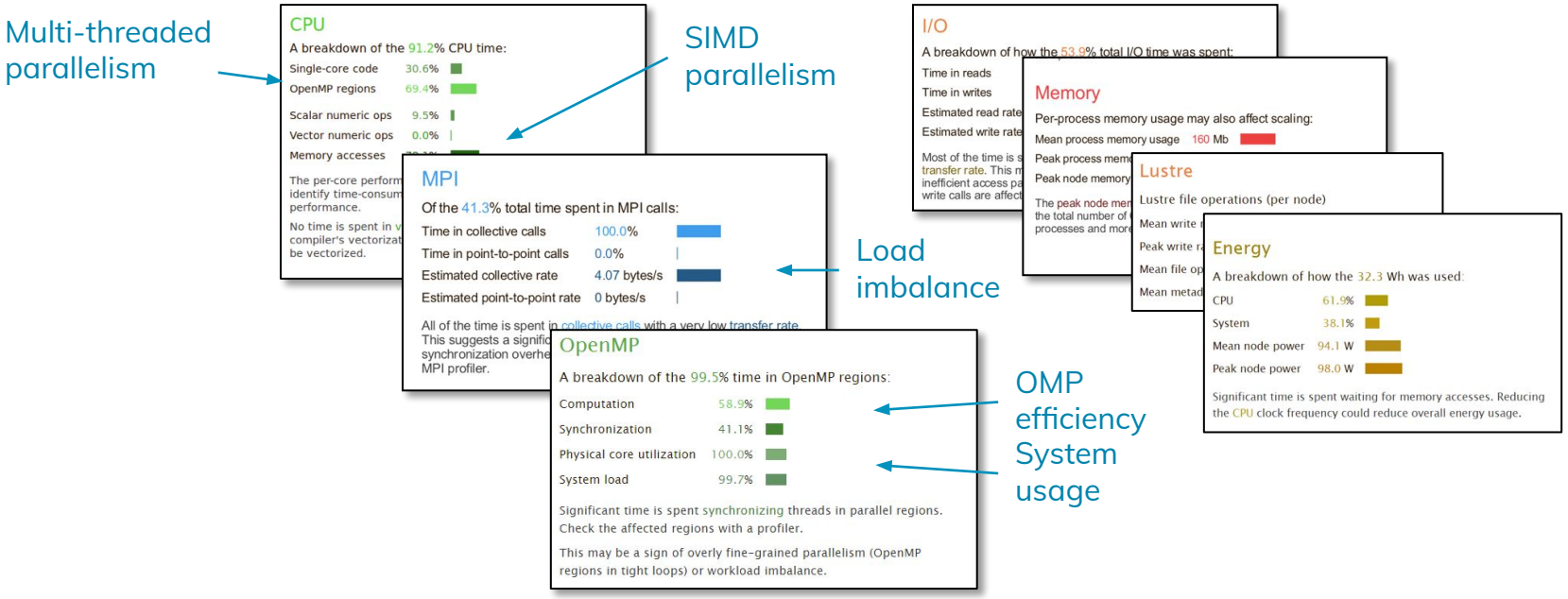
- Example
  $> perf-report mpirun –n 16 mmult.exe





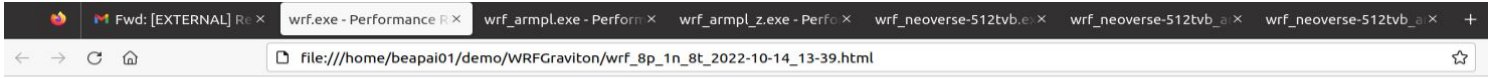
Linaro Performance Reports Metrics

Lowers expertise requirements by explaining everything in detail right in the report



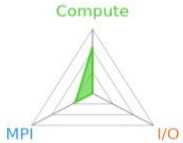
Performance Reports command line options

```
$ perf-report --help
Arm Performance Reports 18.2.1 - Arm Performance Reports

Usage: perf-report [OPTION...] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] MAP_FILE

  --list-metrics                 Display metrics IDs which can be explicitly enabled or disabled.
  --disable-metrics=METRICS              Explicitly disable metrics specified by their metric IDs.
  --enable-metrics=METRICS               Explicitly enable metrics specified by their metric IDs.
  --mpiargs=ARGUMENTS                    command line arguments to pass to mpirun
  --nodes=NUMNODES                       configure the number of nodes for MPI jobs
  -o, --output=FILE              writes the Performance Report to FILE instead of an auto-generated name.
  -n, --np, --processes=NUMPROCS         specify the number of MPI processes
  --procs-per-node=PROCS                 configure the number of processes per node for MPI jobs
  --select-ranks=RANKS                   Select ranks to profile.
```



arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main/wrf.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-82-58.ec2.internal
Start time:	Fri Oct 14 13:39:49 2022
Total time:	897 seconds (about 15 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf.exe is Compute-bound in this configuration

Compute	72.1%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	27.8%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU Metrics

Linux perf event metrics:

Single-core code	4.6%
OpenMP regions	95.4%
Cycles per instruction	0.98
L2D cache miss ratio	2.33
Stalled backend cycles	57.4%
Stalled frontend cycles	1.6%

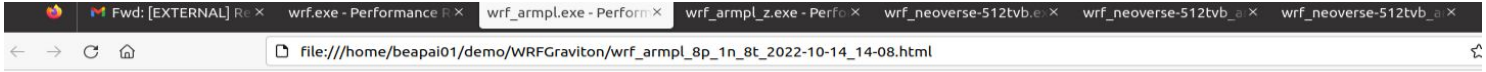
A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 27.8% MPI time:

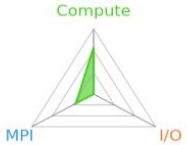
Time in collective calls	65.6%
Time in point-to-point calls	34.4%
Effective process collective rate	37.8 MB/s
Effective process point-to-point rate	669 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.



arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_armpl.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-82-58.ec2.internal
Start time:	Fri Oct 14 14:08:21 2022
Total time:	822 seconds (about 14 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_armpl.exe is Compute-bound in this configuration

Compute	71.5%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	28.4%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU Metrics

Linux perf event metrics:

Single-core code	4.9%
OpenMP regions	95.1%
Cycles per instruction	1.03
L2D cache miss ratio	2.67
Stalled backend cycles	58.3%
Stalled frontend cycles	1.8%

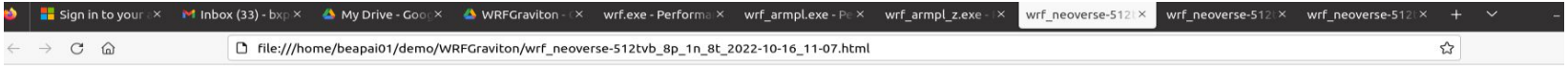
A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 28.4% MPI time:

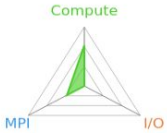
Time in collective calls	70.8%
Time in point-to-point calls	29.2%
Effective process collective rate	38.9 MB/s
Effective process point-to-point rate	838 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.



arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:07:00 2022
Total time:	615 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_neoverse-512tvb.exe is Compute-bound in this configuration

Compute	68.8%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.0%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.2%
OpenMP regions	94.8%
Cycles per instruction	0.66
L2D cache miss ratio	1.13
Stalled backend cycles	65.3%
Stalled frontend cycles	2.2%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.0% MPI time:

Time in collective calls	68.0%
Time in point-to-point calls	32.0%
Effective process collective rate	45.1 MB/s
Effective process point-to-point rate	938 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

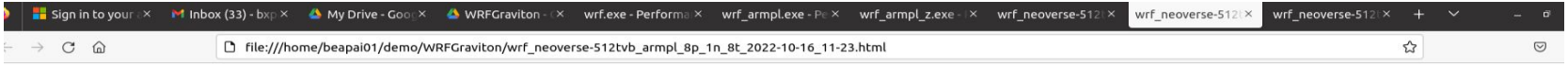
A breakdown of the 0.1% I/O time:

Time in reads	0.0%

OpenMP

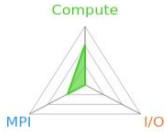
A breakdown of the 94.8% time in OpenMP regions:

Computation	85.3%



arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb_armpl.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:23:15 2022
Total time:	570 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_neoverse-512tvb_armpl.exe is Compute-bound in this configuration

Compute	68.7%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.1%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.3%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.6%
OpenMP regions	94.4%
Cycles per instruction	0.78
L2D cache miss ratio	1.54
Stalled backend cycles	67.1%
Stalled frontend cycles	2.1%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.1% MPI time:

Time in collective calls	74.2%
Time in point-to-point calls	25.8%
Effective process collective rate	39.9 MB/s
Effective process point-to-point rate	1.26 GB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

A breakdown of the 0.3% I/O time:

Time in reads	0.0%

OpenMP

A breakdown of the 94.4% time in OpenMP regions:

Computation	84.9%

Linaro Forge

Announcement

AHUG
ARM HPC USER GROUP

International Workshop on Arm-based HPC: Practice and Experience (IWAHPCE-2024)

in conjunction with HPCAsia2024

Jan 25-27, 2024 | Nagoya, Japan

Topics:

- HPC Applications
- Performance Analysis, Performance Modeling & Measurement
- SVE Vectorization
- Programming Models
- System Software for Arm-based HPC systems
- Networking and accelerators such as GPUs
- Artificial Intelligence and Machine Learning
- Emerging Technologies
- and more …. !

Temporal Important Dates:

Abstract and paper submission due: late Oct. 2023
Paper Notification : late Nov. 2023
Camera ready: Dec. 2023
Workshop Day: 25th Jan. 2024

https://arm-hpc-user-group.github.io/iwahpce-2024/
https://sighpc.ipsj.or.jp/HPCAsia2024/