UNLOCKING EXASCALE

A FIRST LOOK AT JUPITER

30.04.2025 I **EPICURE WEBINAR SERIES**

J.ZJUPA, J.CHEW (JSC/FZJ)

Mitglied der Helmholtz-Gemeinschaft

EPICURE
Unlocking European-level HPC Support

JÜLICH
Forschungszentrum

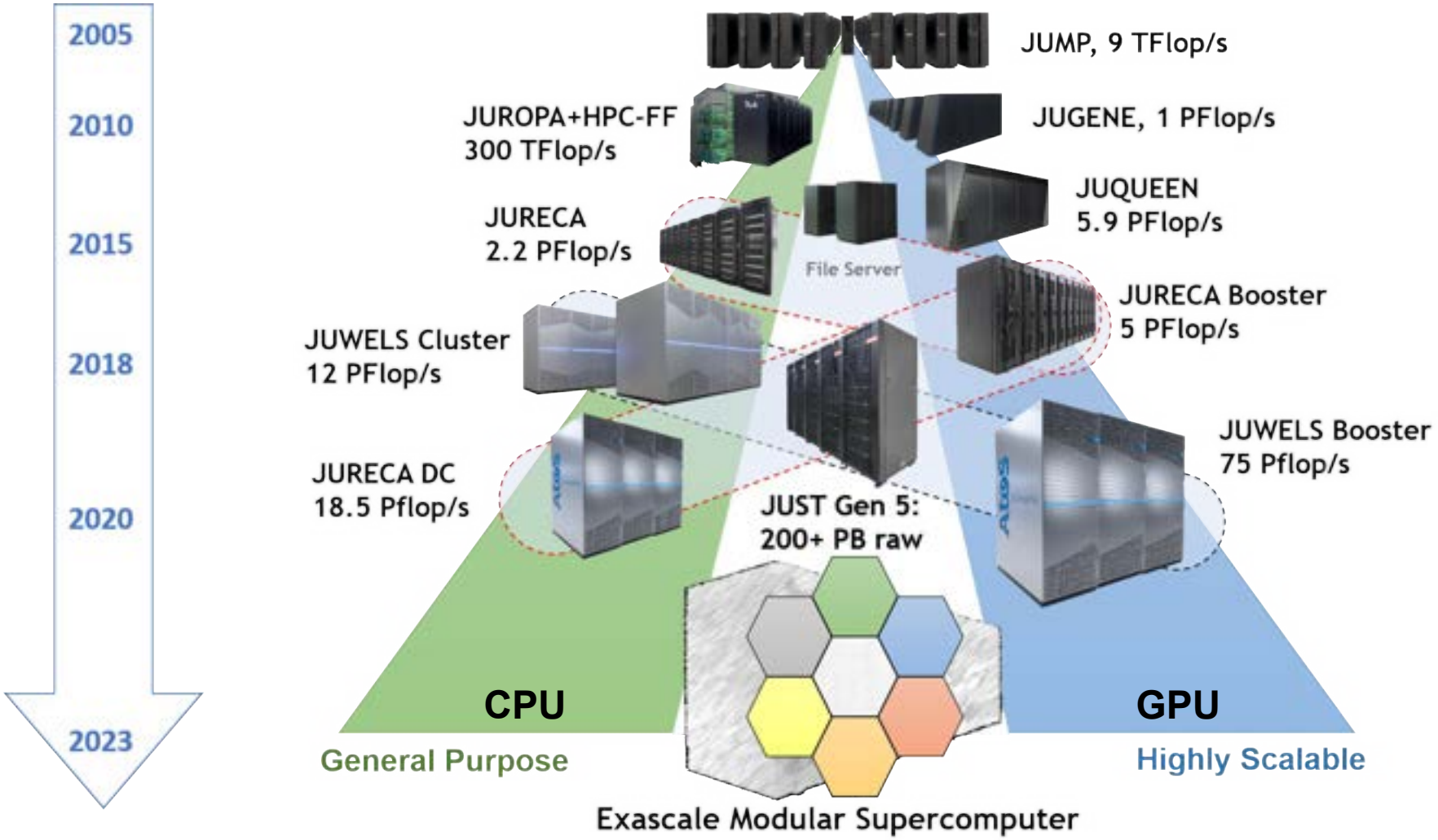
Jülich Supercomputing Centre (JSC)

source: google maps


source: Herwig Zilken / FZJ

source: Herwig Zilken / FZJ

JSC Supercomputing systems



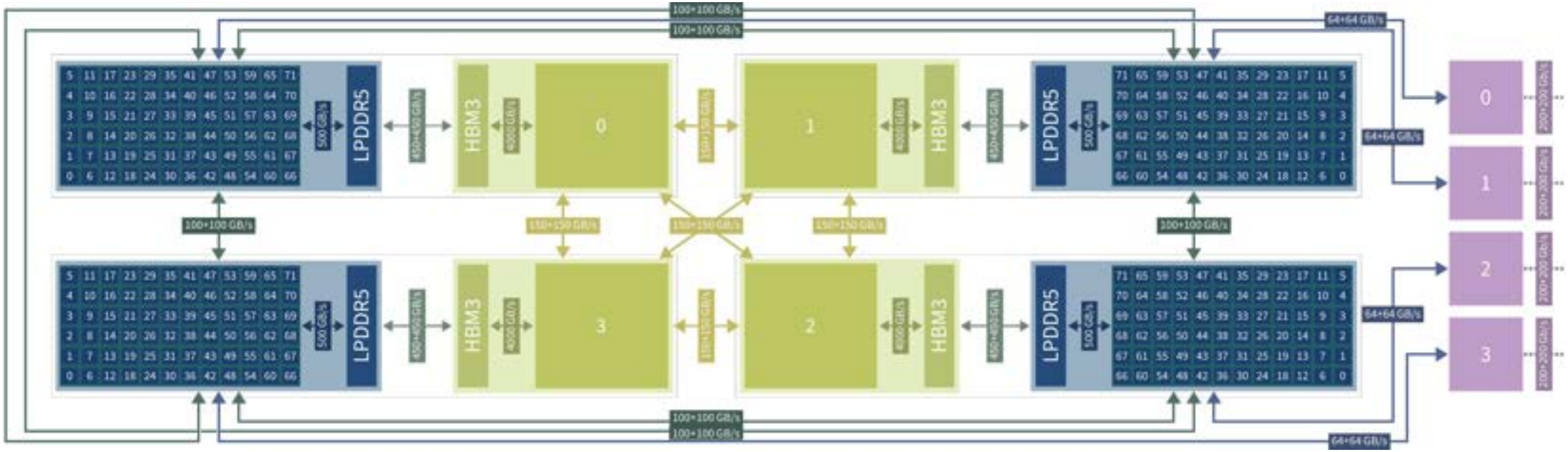
source: Bernd Mohr / JSC

JUPITER Booster

- 1 ExaFLOP/s (FP64, HPL) performance
- ~ 6000 compute nodes

➢ node: 4 NVIDIA *Grace Hopper* GH200 Superchips

NVIDIA *Hopper* H100 GPU: 96 GB memory, NVIDIA *Grace* CPU: 72 ARM cores, 120 GB memory (RAM)



https://www.fz-juelich.de/en/ias/jsc/jupiter/tech

JUPITER HW installation progress

Completed:

- IT Rooms (double container units): 8 out of 8
- 2.5 MW Power substations: 15 out of 15
- Adiabatic towers: 14 out of 14
- Racks: 125 out of 125
- IB cabling: 293 km out of 293 km
- ExaSTORE storage cluster and ExaTAPE

In progress:

- ExaFLASH storage cluster

JUPITER SW installation progress

Completed:

- Management networks completed
- Login node preparation done
- Integration into Slurm routines done
- Integration into other site routines and filesystems done

In progress & next steps:

- Stabilising (onlining) of JUPITER racks and compute nodes
- Testing & early access (JUREAP) expected in the next week(s)

JSC HPC Tools

- **JuDoor** - user portal and project management

https://judoor.fz-juelich.de

- **KontView** - resource utilisation timeline (accessible through JuDoor project page)
- **LLview** - job reporting ----------→

https://llview.fz-juelich.de

- **JUBE** - benchmarking environment

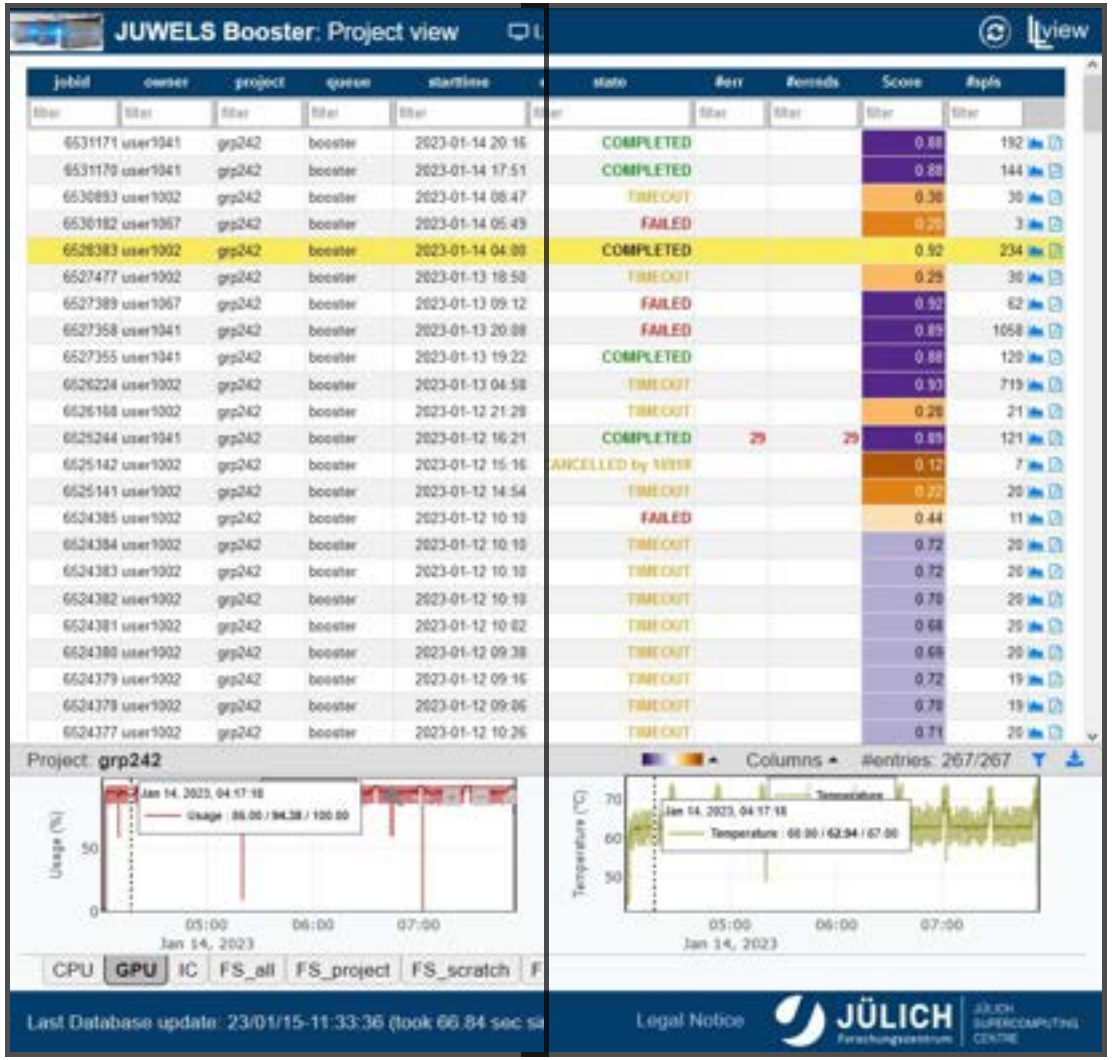
https://www.fz-juelich.de/en/ias/jsc/services/user-support/software-tools/jube

- **Score-P** - profiling & tracing measurement system

https://www.score-p.org

- **Scalasca** - trace analysis toolset

https://www.scalasca.org/



JSC Support Infrastructure

- HPC support: *sc@fz-juelich.de*
- **ATML**s - Algorithms, Tools and Methods Labs

https://www.fz-juelich.de/de/ias/jsc/ueber-uns/struktur/algorithms-tools-and-methods-labs-atmls

- **SDL**s - Simulation and Data Labs

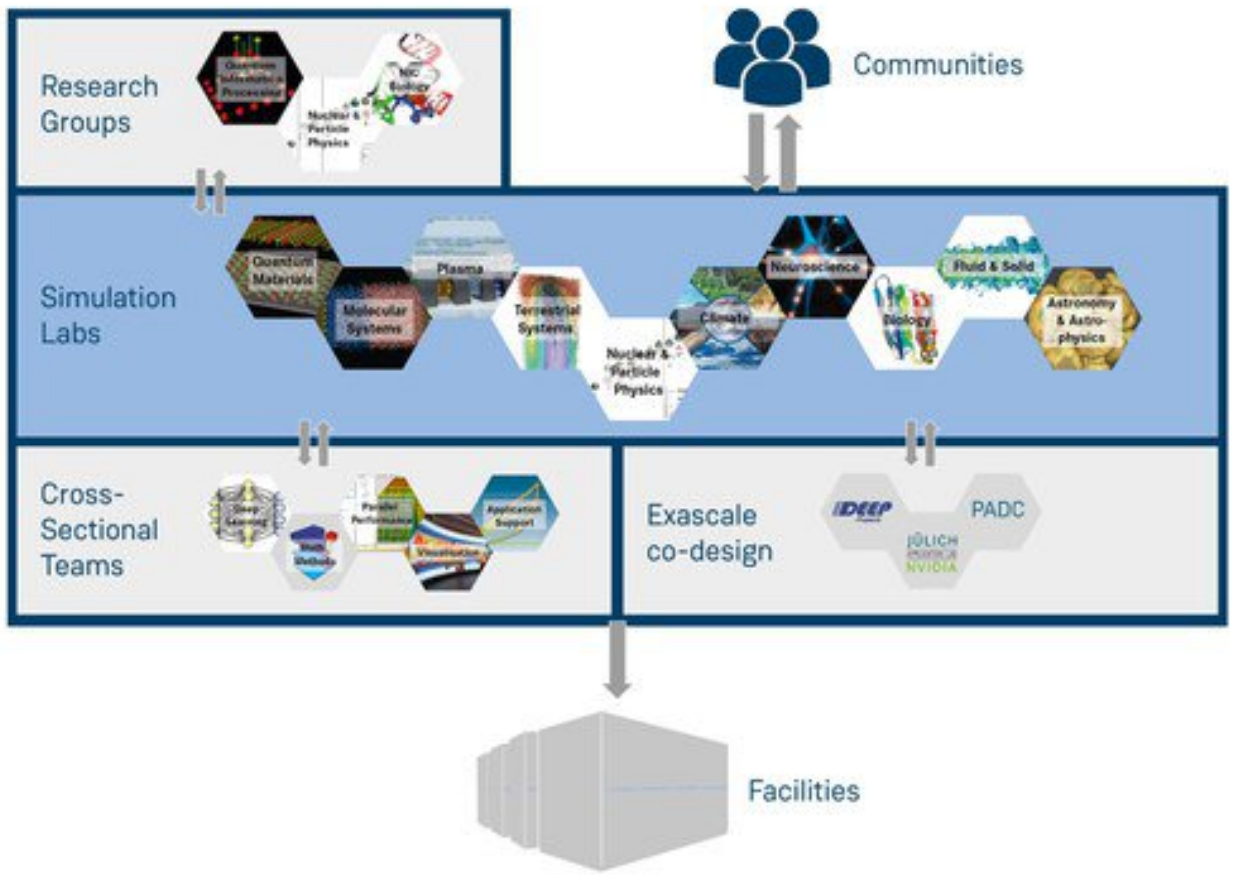
https://www.fz-juelich.de/de/ias/jsc/ueber-uns/struktur/simulation-and-data-labs

- **Project Mentor**

https://www.fz-juelich.de/en/ias/jsc/services/user-support/project-mentoring

- **Training**

https://www.fz-juelich.de/en/ias/jsc/news/events/training-courses/2025



Getting access to JUPITER resources - *national*

50% national (GCS) share - **Regular** & **Large Scale Projects**
next call: **opens 07 July 2025 - closes 11 August 2025, 5pm CEST**
GCS Call 2025-2: *https://www.gauss-centre.eu/news*
GCS Call 2025-1: *https://www.gauss-centre.eu/news/gcs-call-2025-1-for-large-scale-projects-now-open-call-33*

GCS call information:
https://www.gauss-centre.eu/for-users/hpc-access
Important Notice on How to Apply for Computing Time on JUPITER/JUWELS (with fact sheet)
https://www.gauss-centre.eu/for-users/hpc-access/instructions-for-juwels

JSC call information (with fact sheet)
https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/apply-for-computing-time/gcs-nic

Fact sheet for JUPITER (with application link)
https://jards.gauss-centre.eu/PublicFiles/FactSheets/FactSheet-GCS-NIC-JUPITER.pdf

Application link
https://jards.gauss-centre.eu/gcshome/application/

Getting access to JUPITER resources - *European*

50% European (EuroHPC) share

calls: https://eurohpc-ju.europa.eu/supercomputers/supercomputers-access-calls_en

Regular*
- deadline: 5 Sept 2025, 10am CEST
- allocation period: 15 Feb 2026 – 14 Feb 2027

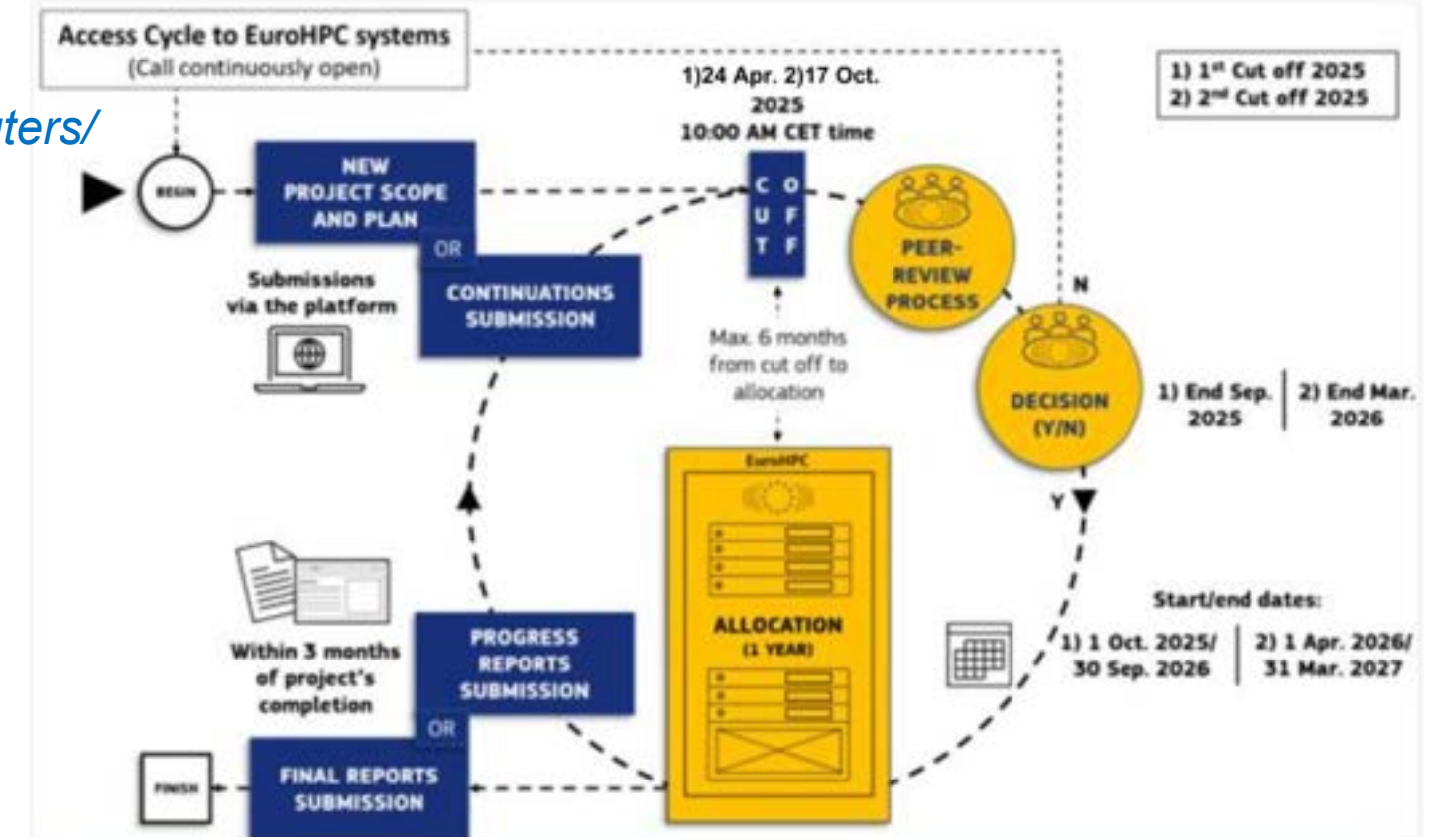
Extreme Scale
- deadline: 17 Oct 2025, 10am CEST
- allocation period: 01 April 2026 – 31 March 2027

Development* & **Benchmarking***
- deadline: 1st of *each month* 2025, 10am CET

AI* (upcoming)
- deadline: every two months

** JUPITER not listed (yet)*

Researchers from academia, research institutes, public authorities, and industry established or located in an EU Member State or in a country associated with Horizon 2020 can apply



https://eurohpc-ju.europa.eu/eurohpc-ju-call-proposals-extreme-scale-access-mode_en

Getting access to JUPITER resources - *complementary*

Test projects*

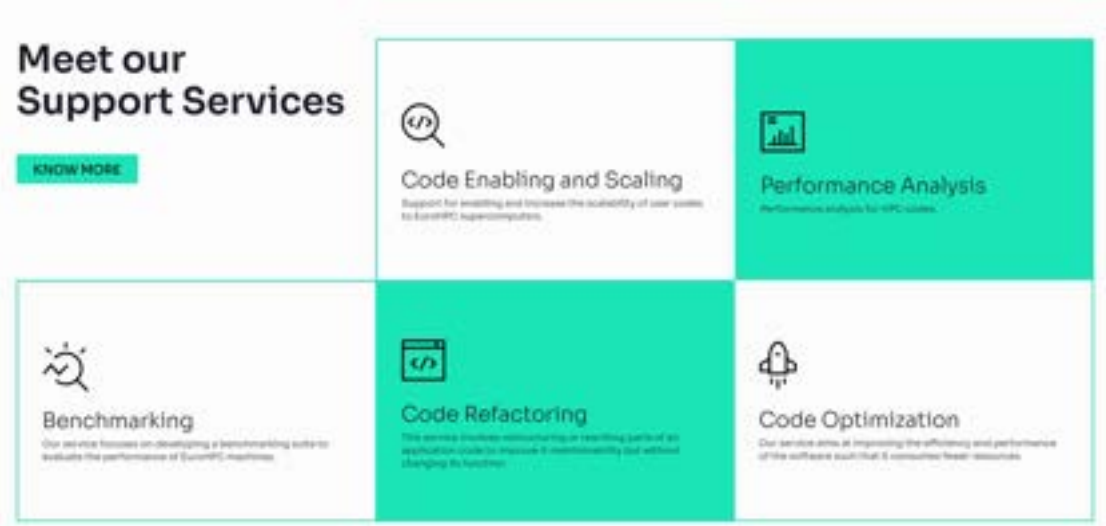
https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/call-for-applications-for-test-projects-with-jsc-supercomputing-and-support-resources

- application: rolling call
- duration: 6 months

Data projects

https://www.fz-juelich.de/en/ias/jsc/services/data-services/data-projects

- application: rolling call
- duration: 1 year
- extension possible



Epicure support available through EuroHPC applications

https://epicure-hpc.eu/support-services

project oriented (intensive), free-of-charge support

a.o. porting, performance analysis, optimisation, debugging, profiling, etc.

** In prep. for JUPITER proposals, best apply for a test project on JUWELS Booster & apply a factor of **2** speed up for GPU-only applications & consider the different number of cores per node*

HARDWARE CONSIDERATIONS AND APPLICATION

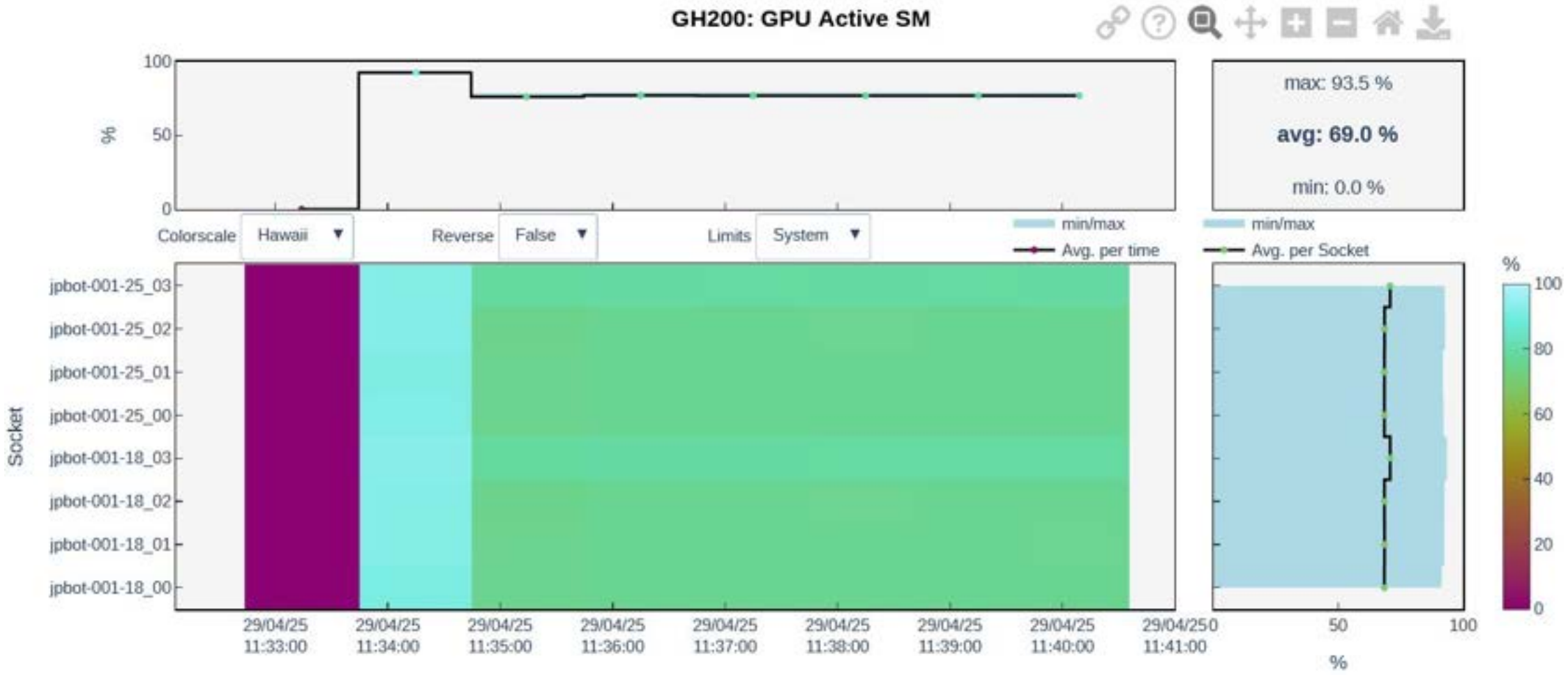
THINGS TO LOOK OUT FOR + APPLICATION PERFORMANCE

Mitglied der Helmholtz-Gemeinschaft

EPICURE
Unlocking European-level HPC Support

JÜLICH
Forschungszentrum

LLview job monitoring on JEDI



LLview job monitoring on JEDI

JobID	Scheduler		Core Usage		Load/Memory		GPU			Power/Energy					Statu	Reports	
	Runtime	#Nodes	Usage	#Cores	Load	MaxMem	GPUActiveSM	GPUMaxMem	GPUMaxClkr	NodePwr	SuperchipPwr	CPUPwr	GPUPower	Energy (est)	S		
	4m	1	0.52	1.00	0.34	51.50		0.28	1	873.75	172.29	53.19	100.27	56.54	COI		
	3m	1	0.95	0.67	0.87	55.50		1.48	1	1,018.33	231.45	40.75	125.59	53.18	I		
	4m	1	0.54	1.00	0.55	65.68		0.02	1	878.25	189.60	38.38	100.02	56.76	COI		
	8m	2	46.80	134.50	24.66	181.82	69.02	16.41	1	1,760.25	398.95	78.85	280.41	462.56	COI		
	32m	1	0.22	0.47	4.14	59.17	0.18	2.14	1	845.72	164.11	50.62	106.57	449.03	I		
	3m	1	1.43	4.00	1.42	62.95	14.91	9.56	1	1,487.33	359.45	49.86	246.48	73.12	COI		
	4m	1	0.61	1.25	100.65	305.28	0.07	4.44	1	890.75	191.99	47.60	120.23	59.40	COI		

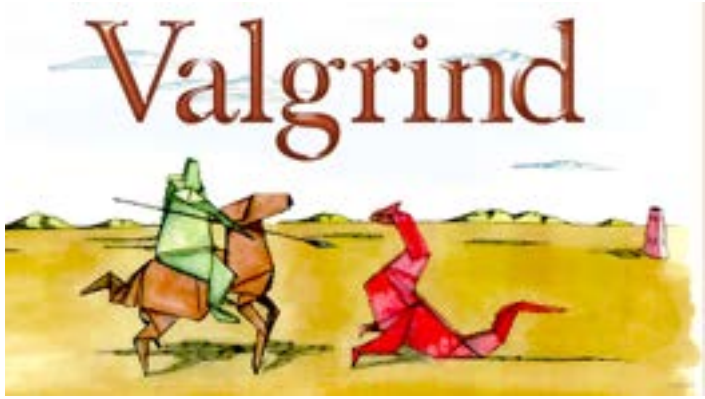
Columns ▲ Showing 4942/4942 entries

Disclaimer: The power measurement feature is in active development.

Other profiling tools on JEDI

Score-P

perforce TotalView



Software stack on JEDI (GH200), GCC toolchain example

Derived from configured ‘Easybuild-framework’:

- Current gcc compiler toolchain in Easybuild already includes ARM-based architecture config.
- https://github.com/easybuilders/easybuild-framework/blob/develop/easybuild/toolchains/compiler/gcc.py

```
# used when 'optarch' toolchain option is enabled (and --optarch is not specified)
    COMPILER_OPTIMAL_ARCHITECTURE_OPTION = {
        (systemtools.AARCH32, systemtools.ARM): '-mcpu=native',
        (systemtools.AARCH64, systemtools.ARM): '-mcpu=native',
        # implies -march=native and -mtune=native
        ...
    }
```

GCC’s aarch64 commit in Sep 2022 made life easier!

- “Rewrite -march=native to -mcpu if no other -mcpu or -mtune is given”

Automatic translation of “-march=native” (x86 option) in Makefiles into “-mcpu=native”.

See: https://gcc.gnu.org/onlinedocs/gcc/AArch64-Options.html

It Just Works***disclaimer time...

GH200 “unified memory”: System vs Managed memory?

Allocation of memory accessible by both CPU/GPU comes in two flavours in unified memory:
- system-allocated memory with “malloc()”
- CUDA managed memory with “cudaMallocManaged()”

Original code

```
t0
cudaMalloc()
t1
cpu_init_data
t2
cudaMemcpy(H2D)
gpu_kernel<<<>>>()
cudaMemcpy(D2H)
cudaDeviceSynchronize();
t3
```

Unified Memory

```
t0
malloc()/cudaMallocManaged()
t1
cpu_init_data
t2

gpu_kernel<<<>>>()

cudaDeviceSynchronize();
t3
```

Pseudo-code example porting of explicit copy CUDA code to unified memory. [1]

Doesn’t need explicit data movements in unified memory!

[1] Schieffer, G., Wahlgren, J., Ren, J., Faj, J. and Peng, I., 2024, August.
Harnessing Integrated CPU-GPU System Memory for HPC: a first look into Grace Hopper.
In Proceedings of the 53rd International Conference on Parallel Processing.

GH200 “unified memory”: System vs Managed memory?

Things to note:

1. First-touch Page Placement
    - First-touch policy: Memory page is not allocated during malloc calls, but on first access (i.e. initialisation)
    - System-allocated memory places page in system page tables **always**. Regardless CPU/GPU first-touch.
    - Managed memory places page in GPU or system page tables, depend on CPU/GPU which accesses first.
2. Access granularity
    - System-allocated memory access is at the granularity of **cache-line**.
    - Managed memory access is at the granularity of **page size**.

What does this mean for GPU only computation????

- In system-allocated memory, if first-touch is done by GPU, slow as CPU has to handle page fault and populate system page table.
- In Managed memory, if first-touch is done by GPU, allocated page is placed in GPU page table. Good performance since data is already on GPU.
- In system-allocated memory, if first-touch is done by CPU, faster initialisation and the cache-line sized CPU-to-GPU transfers **spreads out the the cost of access**.
- In Managed memory, if first-touch is done by CPU, **upfront high cost of access (page-sized)** but lower compute time afterwards since data is then read directly from GPU memory.

GH200 “unified memory”: System vs Managed memory?

IT’S TOO COMPLICATED!!!!

Well... Existing CUDA code with explicit copies still work fine...
Just a heads up for folks who uses cudaMallocManaged()...

Recommended read!
https://arxiv.org/abs/2407.07850

EPICURE
Unlocking European-level HPC Support

ARM vs x86: Memory Ordering Model?

Sequential Consistency... on the software level?

- Compiler can ensure sequential consistency of operations of a single thread.
- Compiler does not know which variable is shared between threads!
- Best to tell compiler ourselves, how to synchronise the accesses between threads.

Software Memory Models says, if program is data-race-free (DRF), Sequential Consistency (SC) is guaranteed. For memory critical operations, usage of synchronisation operations like mutex or atomic constructs are common strategies.

But... how does this related to hardware's memory model???

- "load acquire - store release" concept from mutexes and atomic constructs generate special instructions that hardware tries to adhere to.

Architecture	Load (acquire)		Store (release)	
	Ordinary	SC atomic	Ordinary	SC atomic
x86	mov	mov	mov	xchg
Armv8-A (onwards)	LDR	LDAR	STR	STLR

Nvidia's Grace CPU implements the Armv9.0-A architecture.

x86: Strong memory ordering model

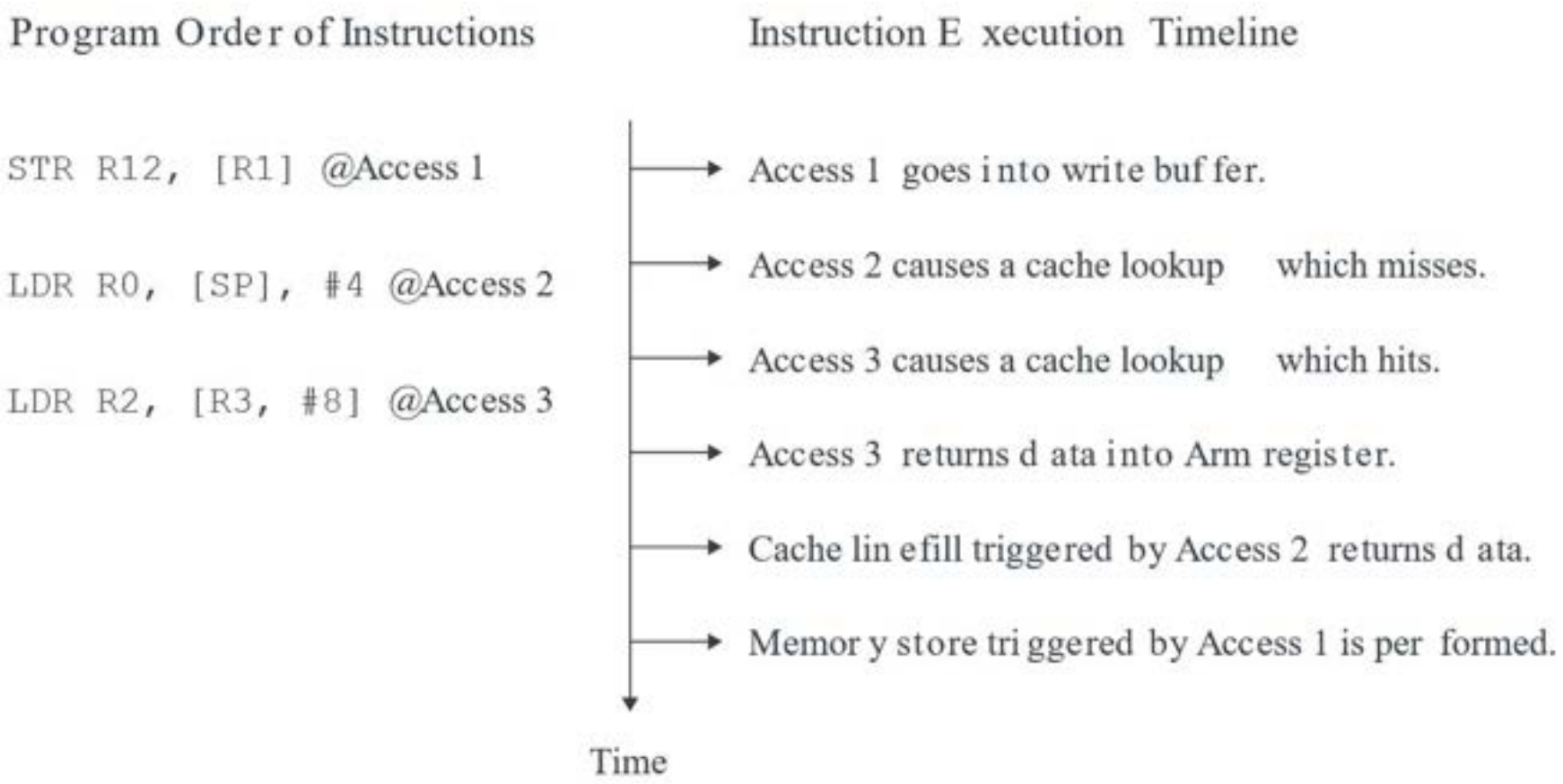
Intel 64 memory ordering obeys the following principles:

1. Loads are not reordered with other loads.
2. Stores are not reordered with other stores.
3. Stores are not reordered with older loads.
4. Loads may be reordered with older stores to different locations but not with older stores to the same location.
5. In a multiprocessor system, memory ordering obeys causality (memory ordering respects transitive visibility).
6. In a multiprocessor system, stores to the same location have a total order.
7. In a multiprocessor system, locked instructions have a total order.
8. Loads and stores are not reordered with locked instructions.

refer to: https://www.cs.cmu.edu/~410-f10/doc/Intel_Reordering_318147.pdf

ARM: Weak memory ordering model

Reordering of consecutive loads allowed on ARM architecture!



refer to: https://developer.arm.com/documentation/102336/0100/Memory-ordering

ARM vs x86: Relaxed atomics

Normal atomics in software level SC-DRF enforces load (acquire) - store (release) ordering!
– generated assembly instructions follow* the order of program (synchronised across threads)

In C/C++, relaxed atomics allows reordering of mutex/atomic operations between threads.
– in such cases, the hardware memory ordering will be the safeguard for correct concurrent memory access.

see: https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html

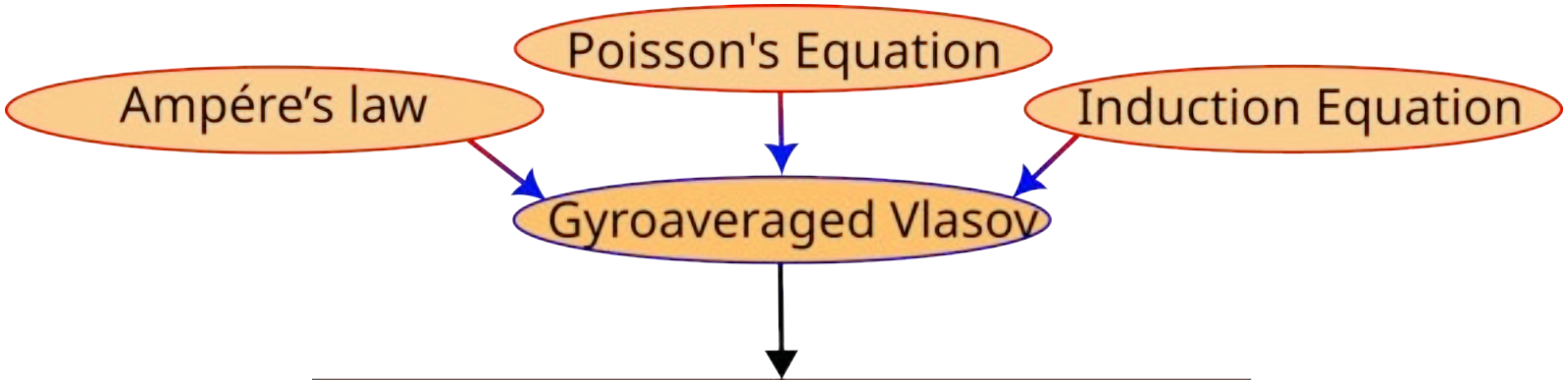
COMPILER  ARCHITECTURE

Take-away message:
- Folks developing compilers works with assembly instruction sets provided by architectures
- Rarely, compiler devs might mistakenly use improper instruction sets.
- Best Practice:
  - compare benchmark results between architectures, especially for highly optimised code with concurrent memory accesses

Gyrokinetic Electromagnetic Numerical Exp. (GENE)

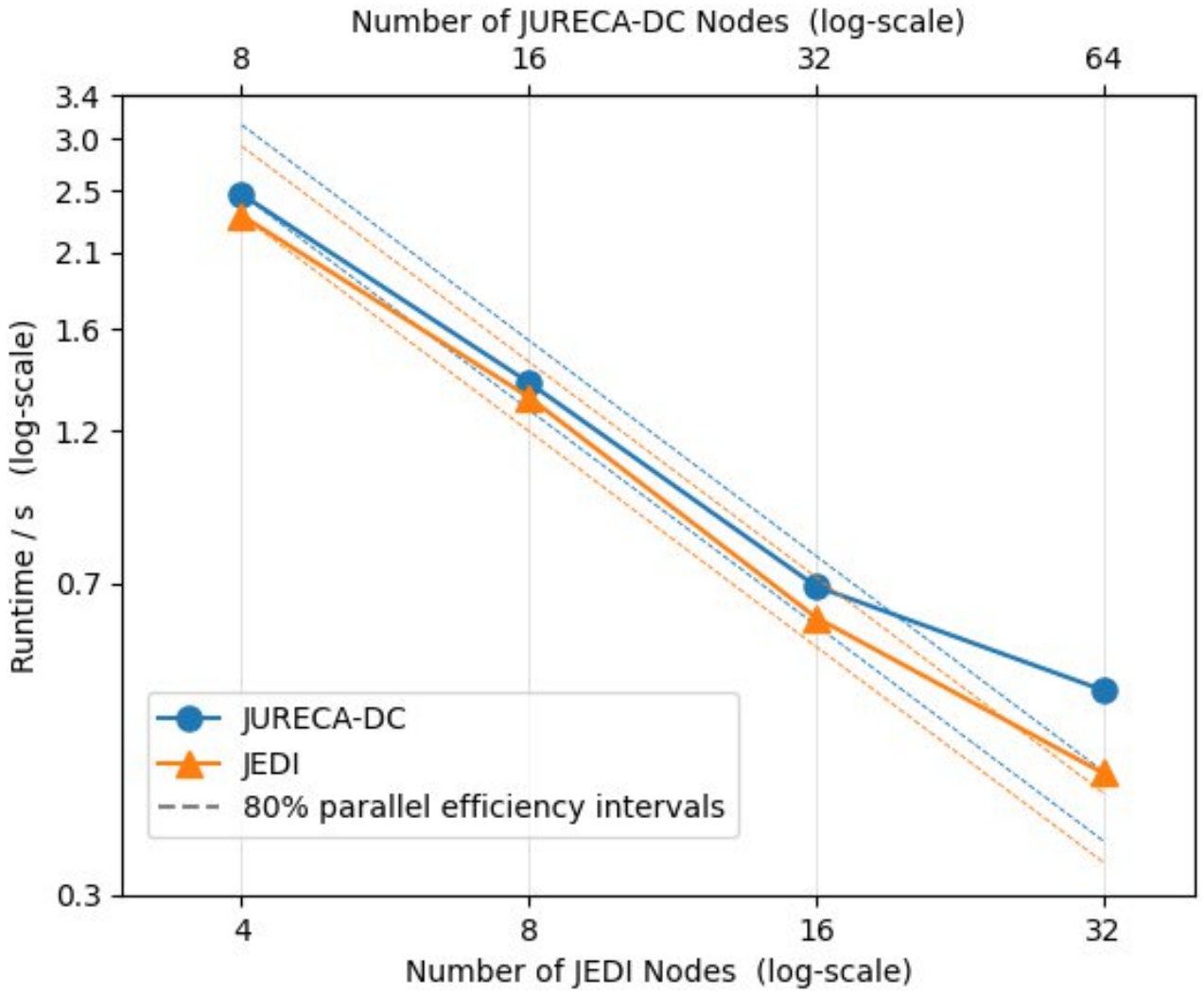
- In development for > 20 years
- Fortran language with C++ interface for CUDA offloading
- Structured grids
- Sparse matrix solver
- Spectral + FVM numerical method
- Uses PETSc for eigenvalue computation
- Fully ported to GPU computation with CUDA since 2021
- Open source with active development from research groups
- See: https://genecode.org/

Gyrokinetic Electromagnetic Numerical Exp. (GENE)



$$\frac{\partial f}{\partial t} + \dot{\mathbf{X}} \cdot \nabla f + \dot{\mu}\frac{\partial f}{\partial \mu} + \dot{v}_\parallel \frac{\partial f}{\partial v_\parallel} = 0$$

Performance comparison: A100 VS H100



- Simulation setup description:
  - 200 × 128 × 96 × 70 × 24 grid points
  - Simulates the narrow transport barrier at the edge of an H-mode fusion plasma
  - understand the suppression mechanism of plasma microturbulences at pedestal
- JURECA-DC_GPU carries 4 A100s each node
- JEDI carries 4 GH200 superchips each node
- Strong scaling on twice the number of A100 GPUs
- Observed >2x speedup on Hopper

Take away messages!

- Easybuild toolchain configurations on Github already has aarch64 default values
- Eases the build of software modules on top of toolchains

- GCC compiler has in-built adaptation of `-march=native` flag to corresponding Aarch64 option
- If user application Makefile uses architecture specific flags, refer to compiler specific documentation!
- Other compilers (NVHPC, CLANG, LLVM) may/may not require special attention on aarch64 vs x86

- GH200 comes with Unified Memory, which is different now a different concept of cudaMallocManaged()
- Different memory access granularity between system-allocated and managed memory
- Careful with variable initialisation (CPU/GPU), choose the best for your application

- Be aware that aarch64 is a weak memory ordering model
- Compilers should translate program code to proper assembly instructions based on architecture for SC
- Potential edge cases where SC breaks down, benchmark correctness of output with x86-like MM hardware

THANK YOU FOR YOUR TIME!