DISTRIBUTED GPU PROGRAMMING FOR EXASCALE

ISC25 TUTORIAL *SESSION 1*

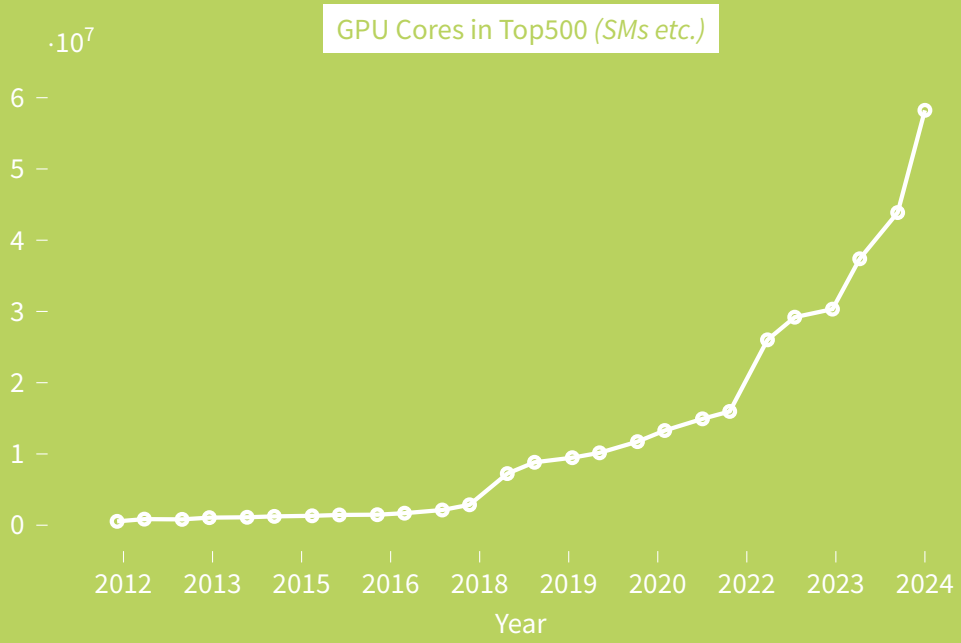
13 June 2025 | Andreas Herten | Jülich Supercomputing Centre, Forschungszentrum Jülich

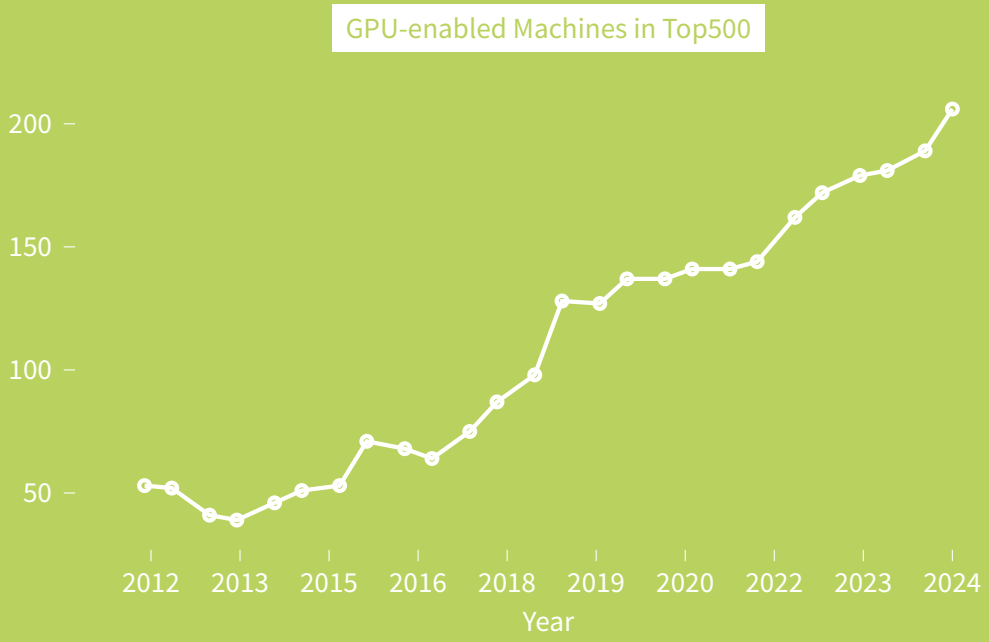
JÜLICH
Forschungszentrum

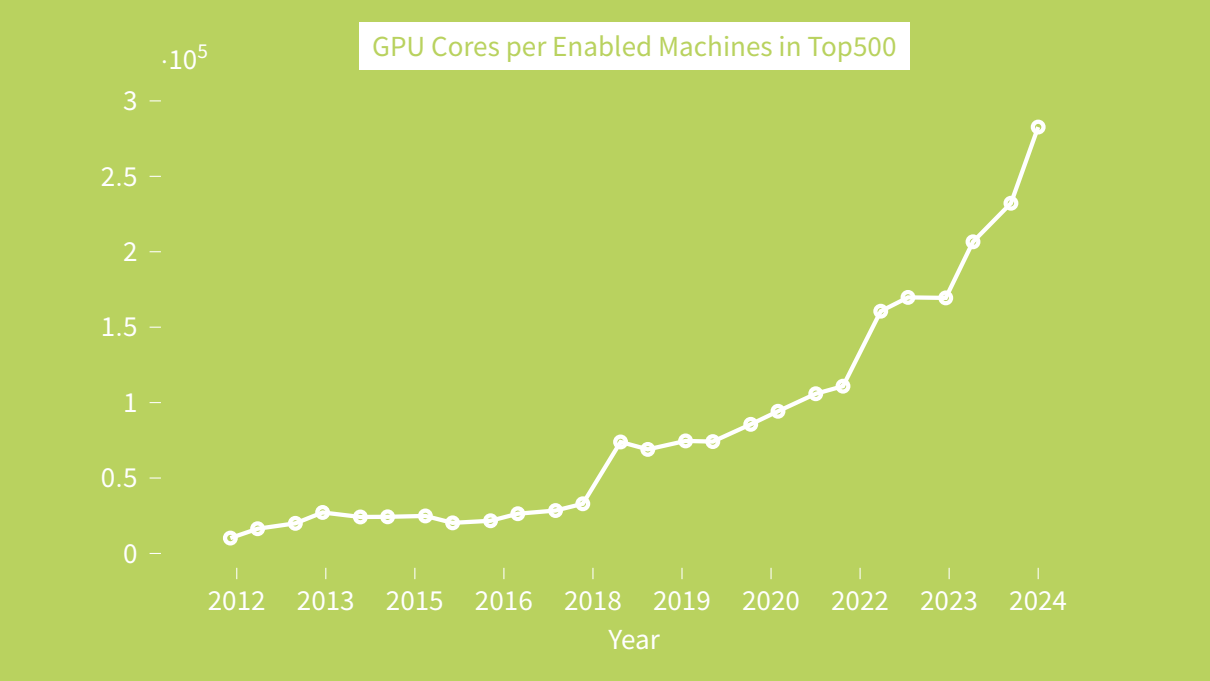
Welcome to

Efficient Distributed GPU Programming for Exascale,

an *ISC25 Tutorial*


GPU Cores in Top500 (SMs etc.)
·10^7
6
5
4
3
2
1
0
2012
2013
2015
2016
2018
2019
2020
2022
2023
2024
Year


GPU-enabled Machines in Top500
200
150
100
50
2012
2013
2015
2016
2018
2019
2020
2022
2023
2024
Year


GPU Cores per Enabled Machines in Top500
·10^5
3
2.5
2
1.5
1
0.5
0
2012
2013
2015
2016
2018
2019
2020
2022
2023
2024
Year

State of the GPUnion

Landscape Overview

- Last decade
  - More and more GPUs installed in HPC machines
  - More and more HPC machines with GPUs
  - More and more GPUs in each machine

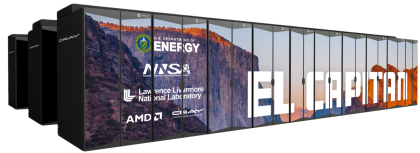
State of the GPUnion

Landscape Overview

- Last decade
  - More and more GPUs installed in HPC machines
  - More and more HPC machines with GPUs
  - More and more GPUs in each machine
- Future
  - GPUs selected as technology for enabling Exascale

  → Even larger GPU machines with larger GPUs

  - Pre-Exascale systems: LUMI(A), Leonardo(N); Perlmutter(N)
  - Exascale systems: Frontier(A), El Capitan(A), Aurora(I); JUPITER(N)



LLNL's El Capitan: ≈ 44 544 GPUs (APUs), 1.74 EFLOP/s, 2024
Rendering by LLNL

(A): AMD, (I): Intel, (N): NVIDIA

JÜLICH Forschungszentrum

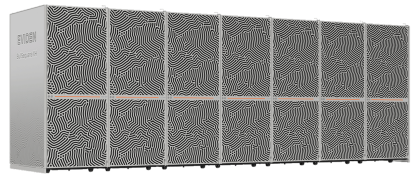
State of the GPUnion

Landscape Overview

- Last decade
  - More and more GPUs installed in HPC machines
  - More and more HPC machines with GPUs
  - More and more GPUs in each machine
- Future
  - GPUs selected as technology for enabling Exascale

  → Even larger GPU machines with larger GPUs

  - Pre-Exascale systems: LUMI[A], Leonardo[N]; Perlmutter[N]
  - Exascale systems: Frontier[A], El Capitan[A], Aurora[I]; JUPITER[N]



JSC's JUPITER: ≈ 24 000 GPUs, 0.8 EFLOP/s, 2025
Rendering by Atos

A: AMD, I: Intel, N: NVIDIA

JÜLICH Forschungszentrum

State of the GPUnion

Landscape Overview

- Last decade
  - More and more GPUs installed in HPC machines
  - More and more HPC machines with GPUs
  - More and more GPUs in each machine
- Future
  - GPUs
  - → Even
  - Pre-Exascale systems: LUMI[A], Leonardo[N]; Perlmutter[N]
  - Exascale systems: Frontier[A], El Capitan[A], Aurora[I]; JUPITER[N]

We help to Tame the Beasts!

JSC's JUPITER: ≈ 24 000 GPUs, 0.8 EFLOP/s, 2025
Rendering by Atos

A: AMD, I: Intel, N: NVIDIA

JÜLICH Forschungszentrum

About Tutorial

Goals

- Prepare for large-scale GPU systems
- Learn GPU+MPI basics
- Show CPU-less GPU+MPI
- Outline advanced libraries to improve scaling efficiency
- Learn interactively!

About Tutorial

Goals

- Prepare for large-scale GPU systems
- Learn GPU+MPI basics
- Show CPU-less GPU+MPI
- Outline advanced libraries to improve scaling efficiency
- Learn interactively!

About Tutorial

Goals

- Prepare for large-scale GPU systems
- Learn GPU+MPI basics
- Show CPU-less GPU+MPI
- Outline advanced libraries to improve scaling efficiency
- Learn interactively!

Non-Goals

- Optimize your GPU application; we teach tools and techniques, you need to apply!
- Learn MPI; we expect base-level knowledge of MPI, you don't need more.
- Discuss general scalability; GPU-independent features (like load balancing) are too broad a topic
- Learn CUDA; we expect principle knowledge of GPU programming
- Showcase all GPU platforms; we use an NVIDIA system and teach NVIDIA libraries and tools, other platforms (AMD) follow along (see last lecture)

About Tutorial

Goals

- Prepare for large-scale GPU systems
- Learn GPU+MPI basics
- Show CPU-less GPU+MPI
- Outline advanced libraries to improve scaling efficiency
- Learn interactively!

Non-Goals

- Optimize your GPU application; we teach tools and techniques, you need to apply!
- Learn MPI; we expect base-level knowledge of MPI, you don't need more.
- Discuss general scalability; GPU-independent features (like load balancing) are too broad a topic
- Learn CUDA; we expect principle knowledge of GPU programming
- Showcase all GPU platforms; we use an NVIDIA system and teach NVIDIA libraries and tools, other platforms (AMD) follow along (see last lecture)

About Tutorial

Goals

- Prepare for large-scale GPU systems
- Learn GPU+MPI basics
- Show CPU-less GPU+MPI
- Outline advanced libraries to improve scaling efficiency
- Learn interactively!

Half-day version of full-day tutorial

Non-Goals

- Optimize your GPU application; we teach tools and techniques, you need to apply!
- Learn MPI; we expect base-level knowledge of MPI, you don't need more.
- Discuss general scalability; GPU-independent features (like load balancing) are too broad a topic
- Learn CUDA; we expect principle knowledge of GPU programming
- Showcase all GPU platforms; we use an NVIDIA system and teach NVIDIA libraries and tools, other platforms (AMD) follow along (see last lecture)

Curriculum

1L	Lecture	Tutorial Overview, Introduction to System
	Onboarding	*Accessing JUPITER*
2L	Lecture	Introduction to MPI-Distributed Computing with GPUs
3H	Hands-On	Multi-GPU Parallelization
		16:00 - 16:30: Coffee Break
5L	Lecture	Optimization Techniques for Multi-GPU Applications
6H	Hands-On	Overlap Communication and Computation with MPI
11L	Lecture	Outline of Advanced Topics and Conclusion

Curriculum

1L	Lecture	Tutorial Overview, Introduction to System
	Onboarding	*Accessing JUPITER*
2L	Lecture	Introduction to MPI-Distributed Computing with GPUs
3H	Hands-On	Multi-GPU Parallelization
		16:00 - 16:30: Coffee Break
4L	Lecture	Performance and Debugging Tools
5L	Lecture	Optimization Techniques for Multi-GPU Applications
6H	Hands-On	Overlap Communication and Computation with MPI
11L	Lecture	Outline of Advanced Topics and Conclusion
7L	Lecture	Overview of NCCL and NVSHMEM in MPI Programs
8H	Hands-On	Using NCCL and NVSHMEM
9L	Lecture	Device-initiated Communication with NVSHMEM
10H	Hands-On	Device-initiated Communication with NVSHMEM

Curriculum

All material online!

1L	Lecture	Tutorial Overview, Introduction to System
	Onboarding	*Accessing JUPITER*
2L	Lecture	Introduction to MPI-Distributed Computing with GPUs
3H	Hands-On	Multi-GPU Parallelization
		16:00 - 16:30: Coffee Break
4L	Lecture	Performance and Debugging Tools
5L	Lecture	Optimization Techniques for Multi-GPU Applications
6H	Hands-On	Overlap Communication and Computation with MPI
11L	Lecture	Outline of Advanced Topics and Conclusion
7L	Lecture	Overview of NCCL and NVSHMEM in MPI Programs
8H	Hands-On	Using NCCL and NVSHMEM
9L	Lecture	Device-initiated Communication with NVSHMEM
10H	Hands-On	Device-initiated Communication with NVSHMEM

Tutorial Team

Simon Garcia
Scalable Computer Architectures
Sandia National Laboratories

Andreas Herten
Accelerating Devices Lab
Jülich Supercomputing Centre

Lena Oden
Prof. for Computer Engineering
University Hagen

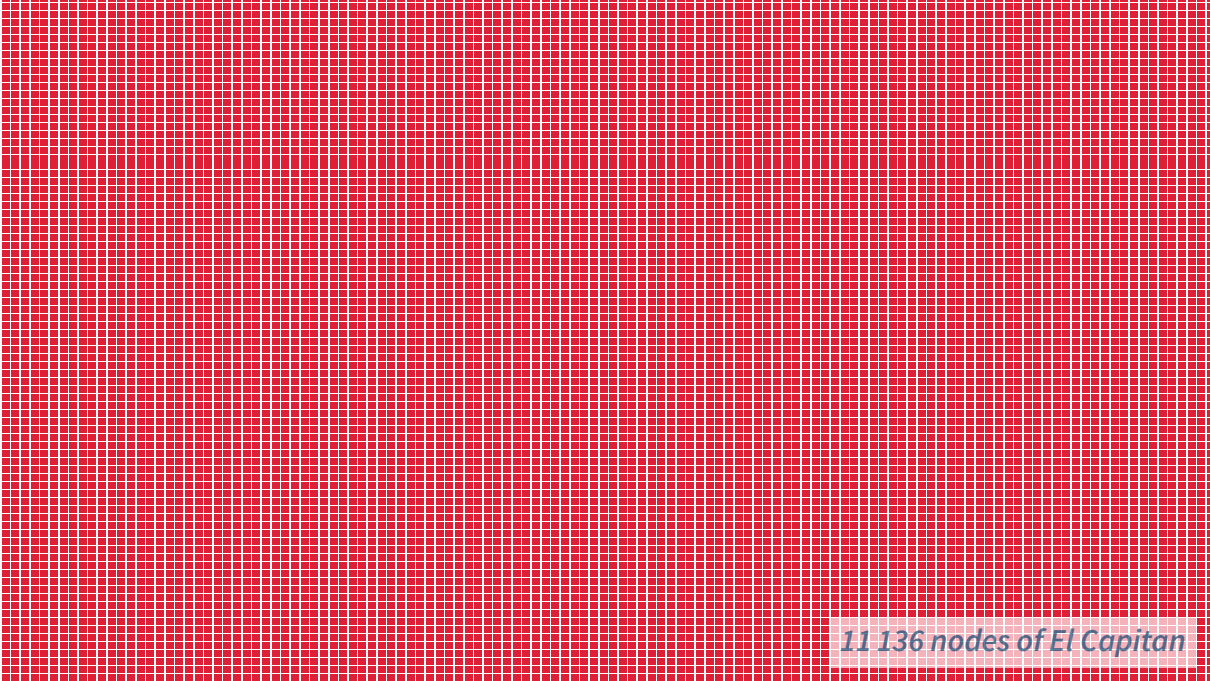
Support by

Jiri Kraus, NVIDIA

Markus Hrywniak, NVIDIA

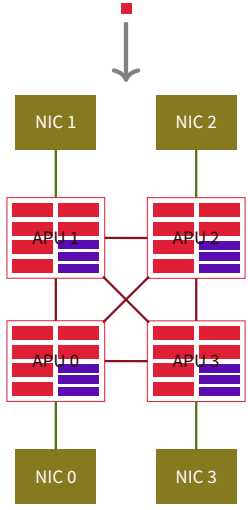
JÜLICH Forschungszentrum

Exascale GPU Systems


11 136 nodes of El Capitan

El Capitan

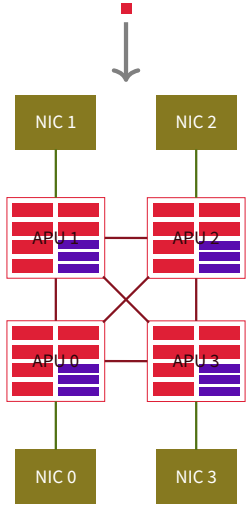
El Capitan



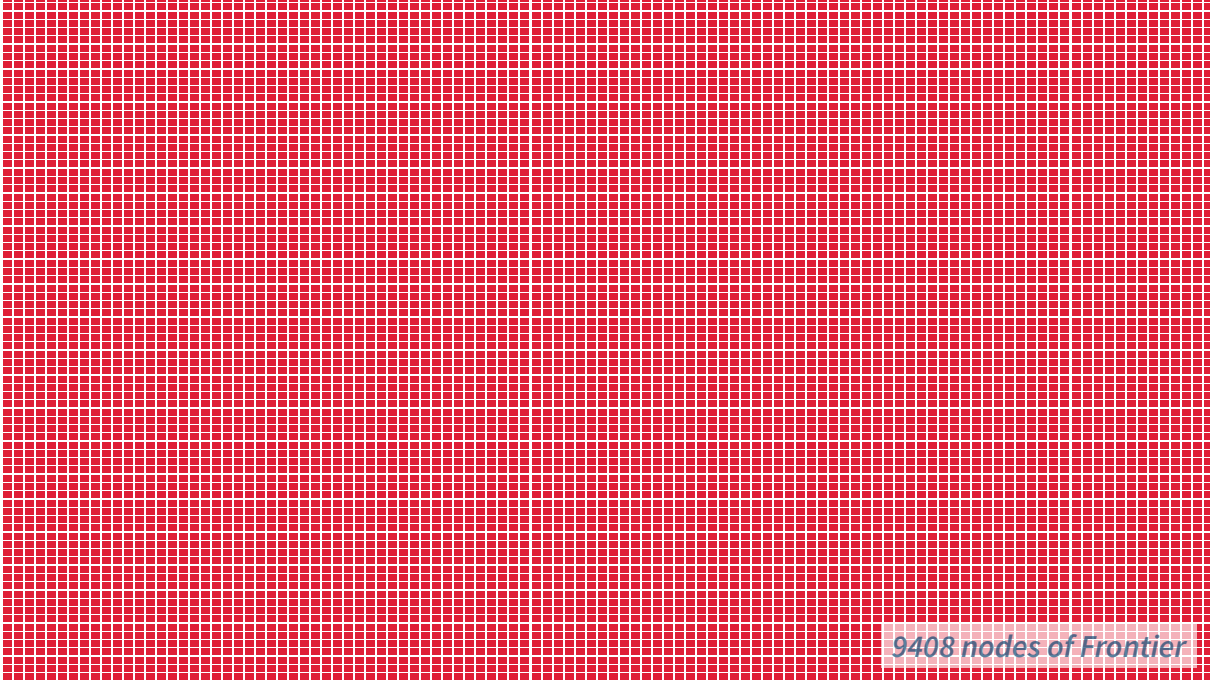
El Capitan

- At Lawrence Livermore National Lab, US
- Largest Exascale system
- HPL: 1.7 EFLOP/s
- APU: 4× AMD Instinct MI300A
  CPU+GPU chiplets combined, 128 GB per APU (HBM)
- Network: 4× HPE Slingshot, 4 × 50 GB/s

→ 11 136 nodes, 44 544 GPUs, 44 544 network devices

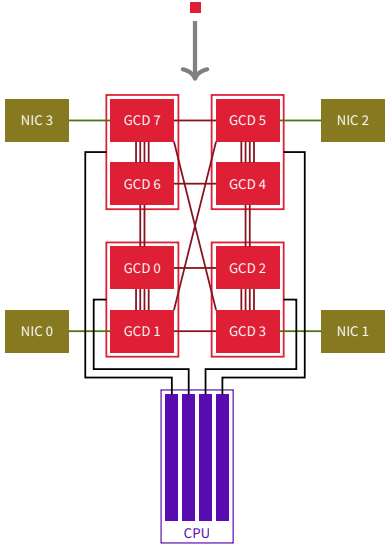


JÜLICH Forschungszentrum


9408 nodes of Frontier

Frontier

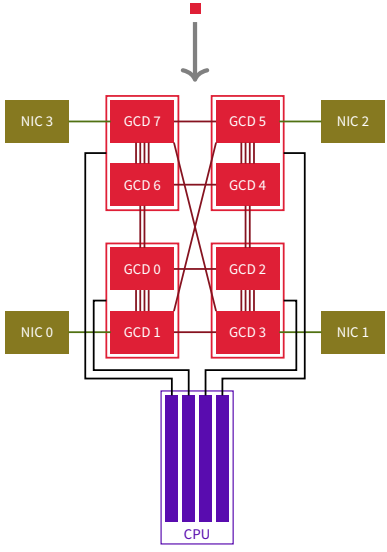
Frontier


NIC 3
GCD 7
GCD 5
NIC 2
GCD 6
GCD 4
GCD 0
GCD 2
NIC 0
GCD 1
GCD 3
NIC 1
CPU

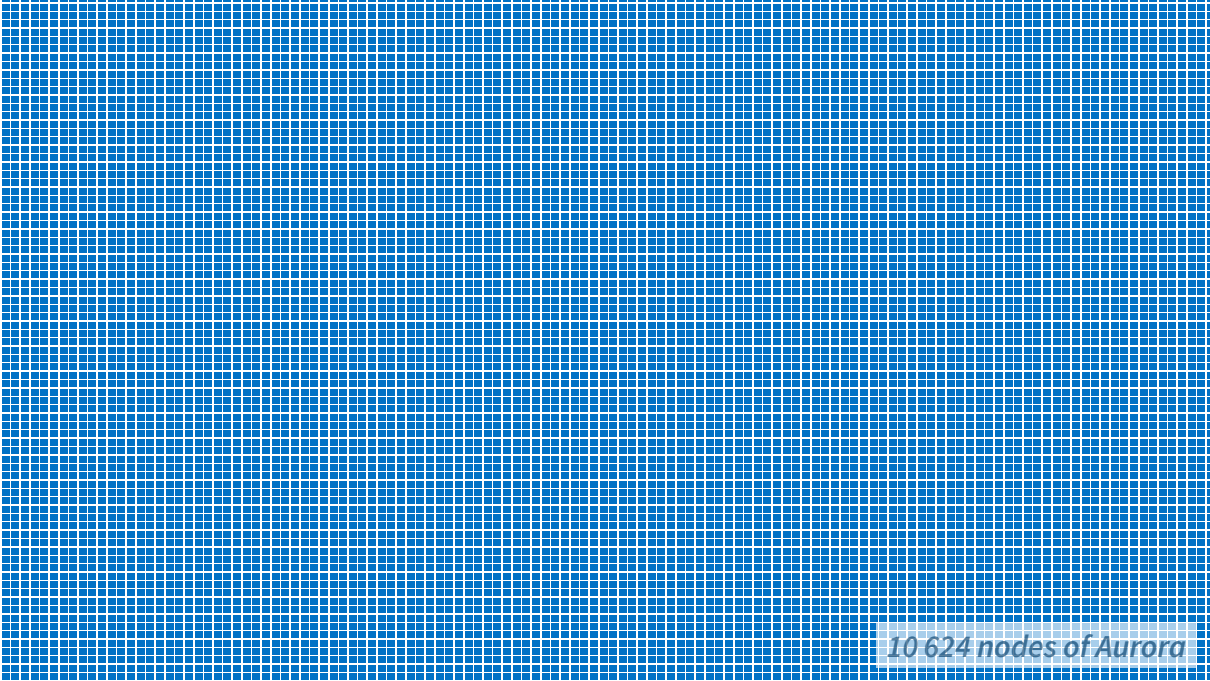
Frontier

- At Oak Ridge Nation Lab, US
- First Exascale system
- HPL: 1.4 EFLOP/s
- GPU: 4× AMD Radeon Instinct MI250X
  2 *GCDs* per GPU, 64 GB memory per GCD
- CPU: 1× AMD Epyc Trento, 64 cores; 4 NUMA domains, one per GCD; 512 GB DDR memory
- Network: 4× HPE Slingshot, 4 × 50 GB/s

→ 9408 nodes, 37 632 GPUs, 75 264 GCDs, 37 632 network devices

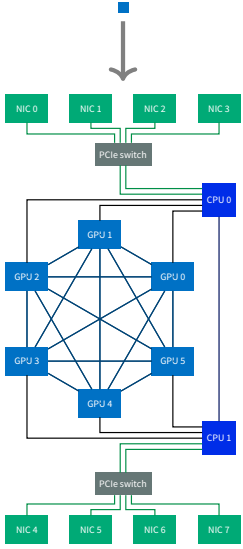


JÜLICH Forschungszentrum


10 624 nodes of Aurora

Aurora

Aurora

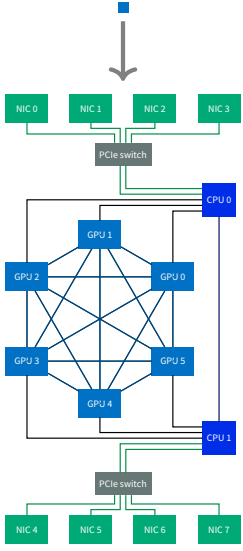


JÜLICH Forschungszentrum

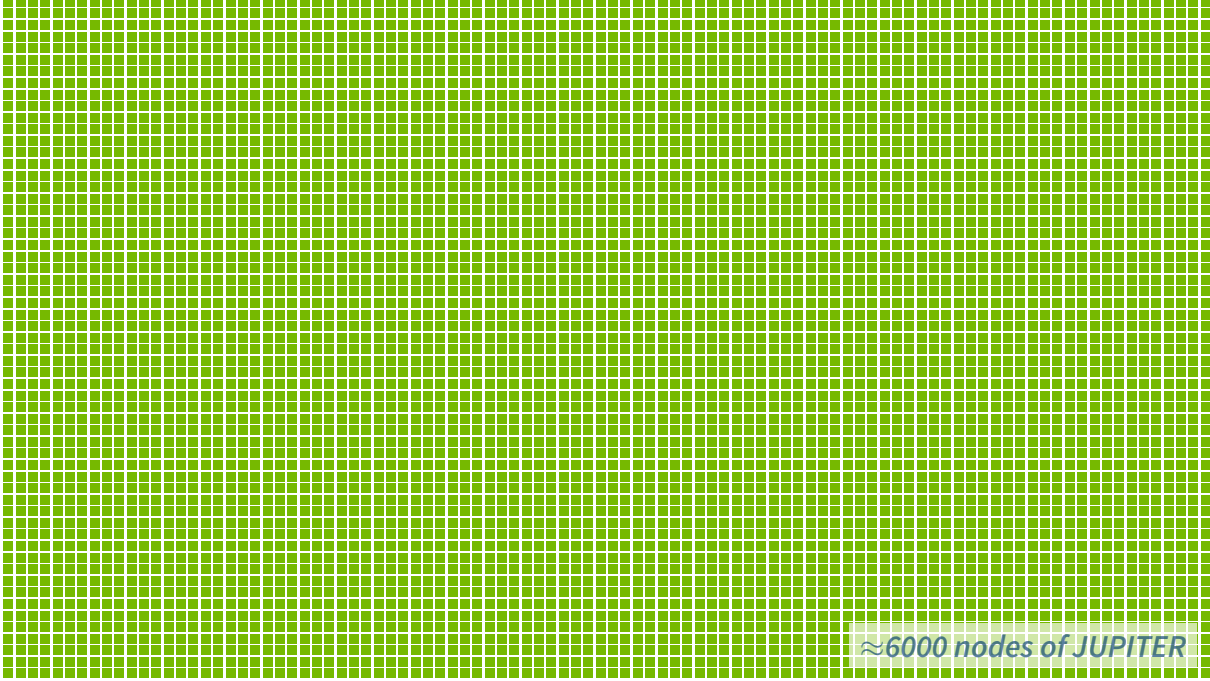
Aurora

- At Argonne National Lab, US
- Highest theoretical peak
- HPL: 1 EFLOP/s
- GPU: 6× Intel Ponte Vecchio (*Data Center GPU Max*) 128 GB memory per GPU
- CPU: 2× Intel Sapphire Rapids, 2× 56 cores; 2× 56 GB HBM memory; 1 TB DDR memory
- Network: 4× HPE Slingshot, 8 × 50 GB/s

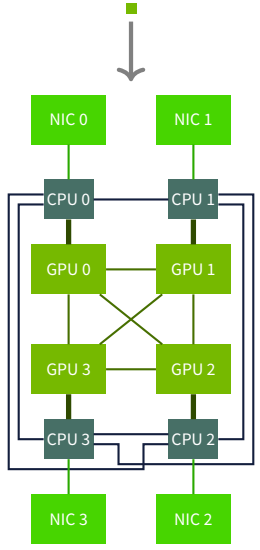
→ 10 624 nodes, 63 744 GPUs, 84 992 network devices



JÜLICH Forschungszentrum


≈6000 nodes of JUPITER

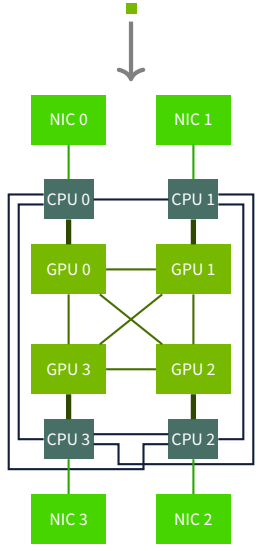
JUPITER



JUPITER

- At Jülich Supercomputing Center, Germany; procured by EuroHPC JU
- First European Exascale system *soon!*
- **Booster**, Cluster
- HPL: 0.8 EFLOP/s
- GPU: 4× NVIDIA H100 *Grace-Hopper flavor*
  96 GB memory per GPU
- CPU: 4× NVIDIA Grace, 4× 72 cores; 4× 120 GB LPDDR5X memory
- Network: 4× NVIDIA InfiniBand NDR200, 4 × 25 GB/s

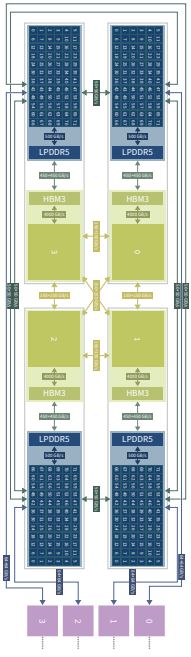
→ ≈6000 nodes, ≈24 000 GPUs, ≈24 000 network devices



JUPITER

- At Jülich Supercomputing Center, Germany; procured by EuroHPC JU
- First European Exascale system *soon!*
- **Booster**, Cluster
- HPL: 0.8 EFLOP/s
- GPU: 4× NVIDIA H100 *Grace-Hopper flavor*
  96 GB memory per GPU
- CPU: 4× NVIDIA Grace, 4× 72 cores; 4× 120 GB LPDDR5X memory
- Network: 4× NVIDIA InfiniBand NDR200, 4 × 25 GB/s

→ ≈6000 nodes, ≈24 000 GPUs, ≈24 000 network devices

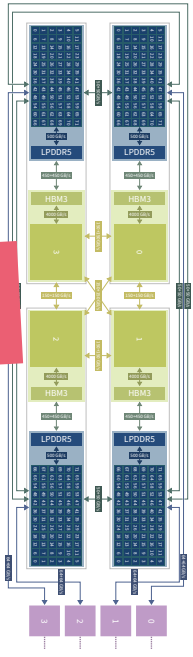


JUPITER

- At Jülich Supercomputing Center, Germany; procured by EuroHPC JU
- First European Exascale system *soon!*
- **Booster**, Cluster
- HPL: 0.8 EFLOP/s
- GPU: 4× NVIDIA
  96 GB memory per
- CPU: 4× NVIDIA Grace, 4× 72 cores; 4× 120 GB LPDDR5X memory
- Network: 4× NVIDIA InfiniBand NDR200, 4 × 25 GB/s

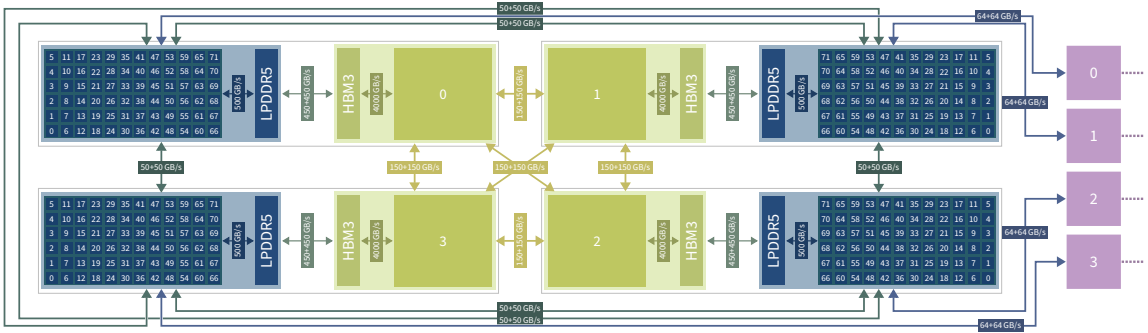
→ ≈6000 nodes, ≈24 000 GPUs, ≈24 000 network devices

Today: JUPITER!



System: JUPITER Booster

Node Topology

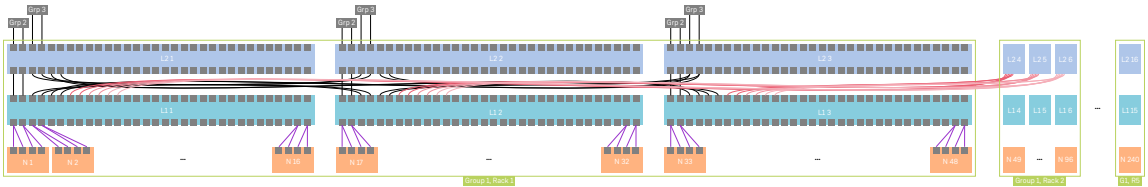


- 4 GH200 superchips, each 120 GB LPDDR5X & 96 GB HBM3 memory → 4 NUMA domains
- 1:1:1 affinity of CPU, GPU, HCA; coherent access between GPU-GPU, CPU-GPU, CPU-CPU
  No fear: Slurm – and we – take care about affinity!

JÜLICH Forschungszentrum

JUPITER Booster System Topology

- NVIDIA Mellanox Quantum InfiniBand NDR200 network
- Dragonfly+ topology: 25 sparsely interconnected groups of 240 well-connected nodes (5 racks, each 48 nodes)
- Per group: 15 L1 switches, 16 L2 switches *(rack 3 has one more)*
- 400 Gbit/s links between switches, 200 Gbit/s links from nodes



JÜLICH Forschungszentrum

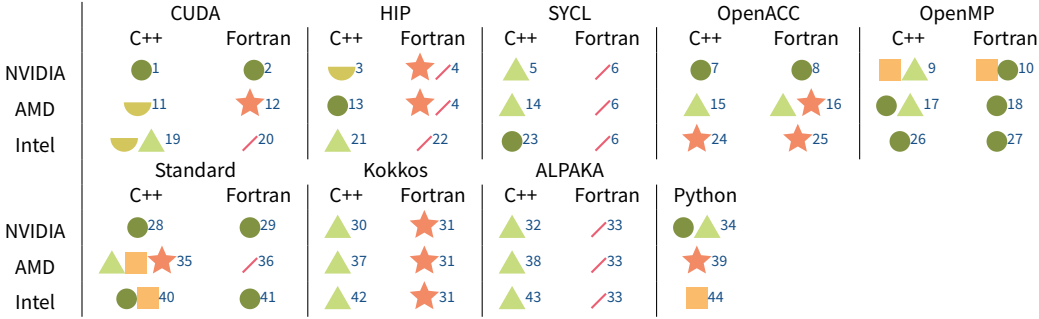
Vendors, Models

GPU Vendors in Tutorial

- First Exascale machine: Frontier, with AMD GPUs!
- This tutorial: NVIDIA examples
  - We have most experience with NVIDIA
  - JUPITER: NVIDIA GPUs
- But: Everything similar for other vendors (especially AMD)
- More on other vendors in last session

GPU Programming Models

- Full vendor support
- Indirect, but comprehensive support, by vendor
- Vendor support, but not (yet) entirely comprehensive
- Comprehensive support, but not by vendor
- Limited, probably indirect
- Vendor support, but not (yet) support – but at least some
- No direct support available
- C++ C++ (sometimes also C)
- Fortran Fortran



See appendix for explanations, or doi:10.34732/xdvblg-r1bvif/doi:10.48550/arXiv.2309.05445

JÜLICH Forschungszentrum

Programming Model in Tutorial

- Tutorial: **CUDA** for GPU programming
- Many other possibilities, especially for NVIDIA GPUs
- Some: higher-level abstractions, mapping back to CUDA
- Conceptionally all similar
- No significant changes needed to use MPI

→ Transfer CUDA knowledge to other models

Summary and Conclusions

Summary and Conclusions

- Exascale and Pre-Exascale systems mainly based on GPUs, with thousands of devices
- Many advanced technologies in place to enable large-scale GPU applications
- Tutorial with team experienced in distributed GPU workloads
- Supercomputer of tutorial: JUPITER (first EU Exascale system)
- This tutorial: Only first half of usual tutorial 😕

Summary and Conclusions

- Exascale and Pre-Exascale systems mainly based on GPUs, with thousands of devices
- Many advanced technologies in place to enable large-scale GPU applications
- Tutorial with team experienced in distributed GPU workloads
- Supercomputer of tutorial: JUPITER (first EU Exascale system)
- This tutorial: Only first half of usual tutorial 😕

Thank you for your attention!

a.herten@fz-juelich.de

JÜLICH Forschungszentrum

Appendix

Appendix

Vendor/Programming Model Table

Appendix

Vendor/Programming Model Table:

GPU Vendor/Programming Model Table I

- 1: CUDA C/C++ is supported on NVIDIA GPUs through the CUDA Toolkit
- 2: CUDA Fortran, a proprietary Fortran extension, is supported on NVIDIA GPUs via the NVIDIA HPC SDK
- 3: HIP programs can directly use NVIDIA GPUs via a CUDA backend; HIP is maintained by AMD
- 4: No such thing like HIP for Fortran, but AMD offers Fortran interfaces to HIP and ROCm libraries in hipfort
- 5: SYCL can be used on NVIDIA GPUs with *experimental* support either in SYCL directly or in DPC++, or via hipSYCL
- 6: No such thing like SYCL for Fortran
- 7: OpenACC C/C++ supported on NVIDIA GPUs directly (and best) through NVIDIA HPC SDK; additional, somewhat limited support by GCC C compiler and in LLVM through Clacc
- 8: OpenACC Fortran supported on NVIDIA GPUs directly (and best) through NVIDIA HPC SDK; additional, somewhat limited support by GCC Fortran compiler and Flacc
- 9: OpenMP in C++ supported on NVIDIA GPUs through NVIDIA HPC SDK (albeit with a few limits), by GCC, and Clang; see OpenMP ECP BoF on status in 2022.
- 10: OpenMP in Fortran supported on NVIDIA GPUs through NVIDIA HPC SDK (but not full OpenMP feature set available), by GCC, and Flang
- 28: pSTL features supported on NVIDIA GPUs through NVIDIA HPC SDK
- 29: Standard Language parallel features supported on NVIDIA GPUs through NVIDIA HPC SDK
- 30: Kokkos supports NVIDIA GPUs by calling CUDA as part of the compilation process
- 31: Kokkos is a C++ model, but an official compatibility layer (*Fortran Language Compatibility Layer*, FLCL) is available.

GPU Vendor/Programming Model Table II

- 32: Alpaka supports NVIDIA GPUs by calling CUDA as part of the compilation process; also, an OpenMP backend can be used
- 33: Alpaka is a C++ model
- 34: There is a vast community of offloading Python code to NVIDIA GPUs, like CuPy, Numba, cuNumeric, and many others; NVIDIA actively supports a lot of them, but has no direct product like *CUDA for Python*; so, the status is somewhere in between
- 11: hipify by AMD can translate CUDA calls to HIP calls which runs natively on AMD GPUs
- 12: AMD offers a Source-to-Source translator to convert some CUDA Fortran functionality to OpenMP for AMD GPUs (gpufort); in addition, there are ROCm library bindings for Fortran in hipfort OpenACC/CUDA Fortran Source-to-Source translator
- 13: HIP is the preferred native programming model for AMD GPUs
- 14: SYCL can use AMD GPUs, for example with hipSYCL or DPC++ for HIP AMD
- 15: OpenACC C/C++ can be used on AMD GPUs via GCC or Clacc; also, Intel's OpenACC to OpenMP Source-to-Source translator can be used to generate OpenMP directives from OpenACC directives
- 16: OpenACC Fortran can be used on AMD GPUs via GCC; also, AMD's gpufort Source-to-Source translator can move OpenACC Fortran code to OpenMP Fortran code, and also Intel's translator can work
- **??**: AMD offers a dedicated, Clang-based compiler for using OpenMP on AMD GPUs: AOMP; it supports both C/C++ (Clang) and Fortran (Flang, example)
- **??**: Currently, no (known) way to launch Standard-based parallel algorithms on AMD GPUs
- 37: Kokkos supports AMD GPUs through HIP
- 38: Alpaka supports AMD GPUs through HIP or through an OpenMP backend

GPU Vendor/Programming Model Table III

- 39: AMD does not officially support GPU programming with Python (also not semi-officially like NVIDIA), but third-party support is available, for example through Numba (currently inactive) or a HIP version of CuPy
- 19: SYCLomatic translates CUDA code to SYCL code, allowing it to run on Intel GPUs; also, Intel's DPC++ Compatibility Tool can transform CUDA to SYCL
- 20: No direct support, only via ISO C bindings, but at least an example can be found on GitHub; it's pretty scarce and not by Intel itself, though
- 21: CHIP-SPV supports mapping CUDA and HIP to OpenCL and Intel's Level Zero, making it run on Intel GPUs
- 22: No such thing like HIP for Fortran
- 23: SYCL is the prime programming model for Intel GPUs; actually, SYCL is only a standard, while Intel's implementation of it is called DPC++ (*Data Parallel C++*), which extends the SYCL standard in various places; actually actually, Intel namespaces everything *oneAPI* these days, so the *full* proper name is Intel oneAPI DPC++ (which incorporates a C++ compiler and also a library)
- **??**: OpenACC can be used on Intel GPUs by translating the code to OpenMP with Intel's Source-to-Source translator
- **??**: Intel has extensive support for OpenMP through their latest compilers
- **??**: Intel supports pSTL algorithms through their DPC++ Library (oneDPL; GitHub). It's heavily namespaced and not yet on the same level as NVIDIA
- 41: With Intel oneAPI 2022.3, Intel supports DO CONCURRENT with GPU offloading
- 42: Kokkos supports Intel GPUs through SYCL

GPU Vendor/Programming Model Table IV

- 43: Alpaka v0.9.0 introduces experimental SYCL support; also, Alpaka can use OpenMP backends
- 44: Not a lot of support available at the moment, but notably DPNP, a SYCL-based drop-in replacement for Numpy, and numba-dpex, an extension of Numba for DPC++.