



JUWELS BOOSTER – EARLY USER EXPERIENCES PERMAVOST WORKSHOP

25 June 2021 | Andreas Herten | Jülich Supercomputing Centre, Forschungszentrum Jülich

Overview

Jülich Supercomputing Centre

JUWELS Booster System

- JUWELS Overall Architecture

- JUWELS Booster Architecture

- JUWELS Integration

JUWELS Timeline

- JUWELS Milestones

- Early Access Program

Early Experiences

System-Related

- Aside: AMD EPYC Core Design

- Intra-Node Affinities

- Inter-Node Affinity

- Job Reportings

Application-Related

- SOMA

- JUQMES

- ParFlow

- JUQCS

- LQCD: Bonn

- Others

Summary and Conclusions

Jülich Supercomputing Centre

Forschungszentrum Jülich Germany, near Cologne,
interdisciplinary research, 6400 employees

Jülich Supercomputing Centre

- Operation of supercomputers
- Education, training
- Application Support, Domain Science Support
- Research & Development

Accelerating Devices Lab Support, research, education for GPUs
et al.; NVIDIA Application Lab at Jülich

Supercomputers

Production JUWELS, JURECA DC, JUSUF

Prototypes JUMAX, DEEP, ...



Jülich Supercomputing Centre

Forschungszentrum Jülich Germany, near Cologne,
interdisciplinary research, 6400 employees

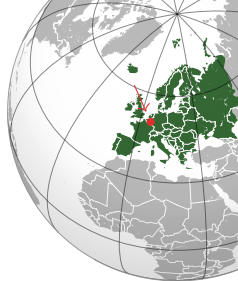
Jülich Supercomputing Centre

- Operation of supercomputers
- Education, training
- Application Support, Domain Science Support
- Research & Development

Accelerating Devices Lab Support, research, education for GPUs
et al.; NVIDIA Application Lab at Jülich

Supercomputers

Production **JUWELS**, JURECA DC, JUSUF
Prototypes JUMAX, DEEP, ...



JUWELS Booster System

JUWELS Overall Architecture

JUWELS Cluster (2018)

- 2511 compute nodes ($2 \times$ Skylake)
- 48 GPU nodes ($4 \times$ V100 w/ NVLink2)
- Mellanox EDR 100 Gbit/s network, fat-tree topology (1:2@L1)
- 12 PFLOP/s



JUWELS Overall Architecture

JUWELS Cluster (2018)

- 2511 compute nodes ($2 \times$ Skylake)
- 48 GPU nodes ($4 \times$ V100 w/ NVLink2)
- Mellanox EDR 100 Gbit/s network, fat-tree topology (1:2@L1)
- 12 PFLOP/s



JUWELS Booster (2020)

- 936 compute nodes ($2 \times$ AMD Rome, $4 \times$ A100 w/ NVLink3)
- Mellanox HDR 200 Gbit/s network, DragonFly+ topology
- 73 PFLOP/s

JUWELS Overall Architecture

JUWELS Cluster (2018)

- 251
- 48
- Me
- fat
- 12



Top500 Nov-2020:

- #1 Europe
- #7 World
- #1* Green500



JUWELS Booster (2020)

- 936 compute nodes (2× AMD Rome, 4× A100 w/ NVLink3)
- Mellanox HDR 200 Gbit/s network, DragonFly+ topology
- 73 PFLOP/s

JUWELS Booster Overview

Node Configuration

Arch Atos Bull Sequana XH2000

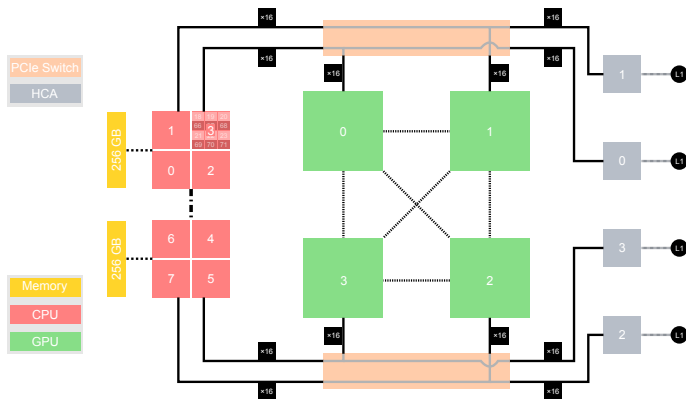
CPU 2 × AMD EPYC 7402:

2_{Socket} × 24_{Core} × 2_{SMT},
2 × 256 GB DDR4-3200 RAM;
NPS-4

GPU 4 × NVIDIA A100 40 GB, NVLink3

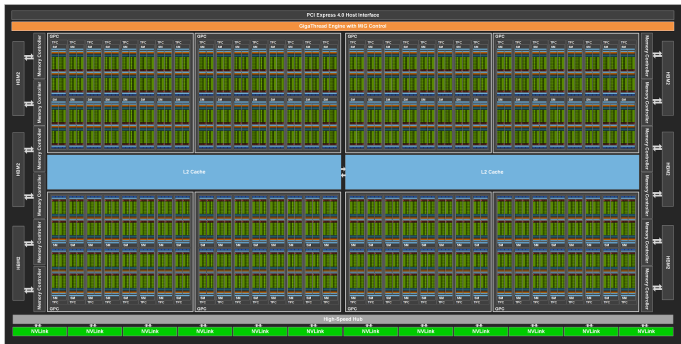
HCA 4 × Mellanox HDR200
(200 Gbit/s) InfiniBand
ConnectX 6

etc 2 × PCIe Gen 4 switch



NVIDIA A100 Tensor Core GPU

- JUWELS Booster:
3744 A100 GPUs
- Vs. V100: More SMs, larger/faster L1, larger/faster L2, faster memory, faster NVLink, faster PCIe, ...
- New features: MIG, async copy to/barrier in shared mem



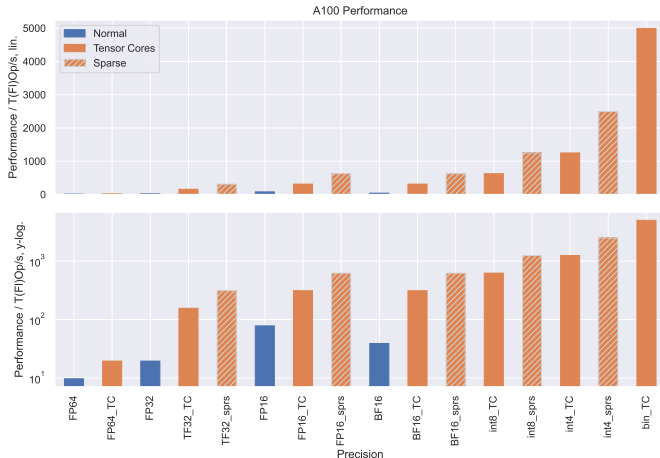
NVIDIA A100 Tensor Core GPU

- JUWELS Booster:
3744 A100 GPUs
- Vs. V100: More SMs, larger/faster L1, larger/faster L2, faster memory, faster NVLink, faster PCIe, ...
- New features: MIG, async copy to/barrier in shared mem
- New Tensor Cores



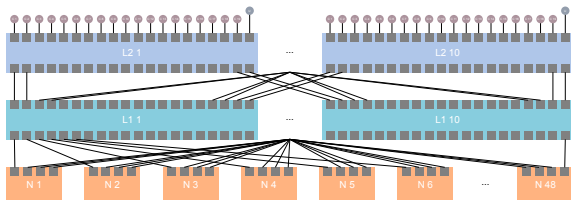
NVIDIA A100 Tensor Core GPU

- JUWELS Booster:
3744 A100 GPUs
- Vs. V100: More SMs, larger/faster L1, larger/faster L2, faster memory, faster NVLink, faster PCIe, ...
- New features: MIG, async copy to/barrier in shared mem
- New Tensor Cores, new performances:
 $73 \text{ PFLOP/s}_{\text{FP64TC}}$ $584 \text{ PFLOP/s}_{\text{FP32TC}}$
 $1.16 \text{ EFLOP/s}_{\text{FP16TC}}$ $18.7 \text{ EOP/s}_{\text{BinTC}}$

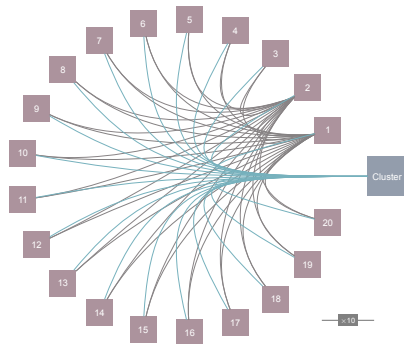


JUWELS Booster Overview

Network Configuration: DragonFly+ Network



In-Cell (48 nodes): Full fat-tree in 2 levels



Inter-Cell (20 cells): 10 links between each pair of cells

JUWELS

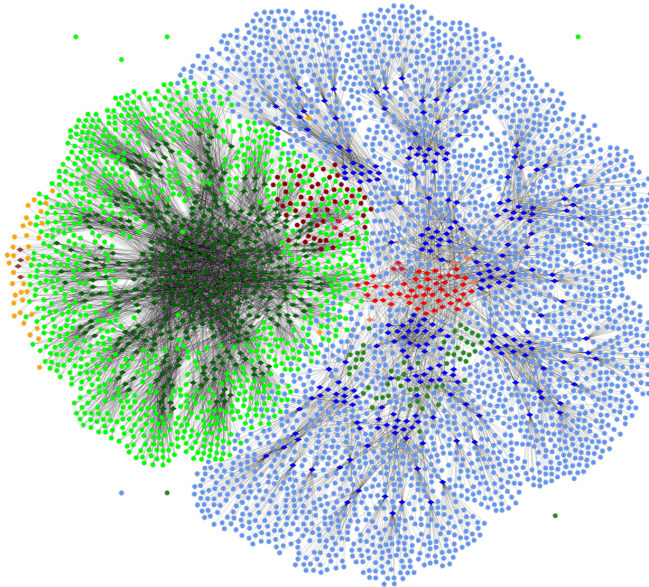
Cluster Booster Integration

Fully integrated system: **JUWELS** with Cluster and Booster modules

- File system: GPFS
- Network: InfiniBand
- Workload management: Slurm
- Resource management: ParaStation / ParaStation Slurm

Picture legend:

- | | |
|--------------------|-------------------|
| ■ Cluster CPU node | ■ Booster node |
| ■ Cluster GPU node | ■ Booster switch |
| ■ Cluster switch | ■ Booster gateway |
| ■ Cluster gateway | ■ JUST |
| ■ Top-level switch | ■ Service node |



JUWELS Software Stack

- Software

- Software management: EasyBuild, LMod
- Compilers: GCC, Intel, NVHPC
- GPU-aware MPIs (ParaStationMPI, OpenMPI; via UCX)

→ https://apps.fz-juelich.de/jsc/llview/juwels_modules_booster/

- Operation

- Operation System: CentOS 8
- Provisioning: Ansible

JUWELS Timeline

JUWELS Milestones

2018 JUWELS Cluster production start

2018 Jun JUWELS Cluster Top500: #23

2019 JUWELS Booster kick-off

2020 Apr JUWELS Booster installation start

2020 May JUWELS Booster Early Access Program first job

2020 Nov JUWELS Booster: production start, first compute-time period; Top500 #7

2021 May JUWELS Booster second compute-time period



JUWELS Milestones

2018 JUWELS Cluster production start

2018 Jun JUWELS Cluster Top500: #23

2019 JUWELS Booster kick-off

2020 Apr JUWELS Booster installation start

2020 May JUWELS Booster **Early Access Program** first job

2020 Nov JUWELS Booster: production start, first compute-time period; Top500 #7

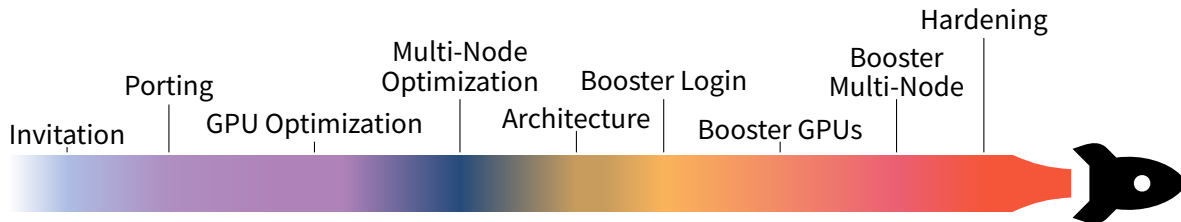
2021 May JUWELS Booster second compute-time period

Early Access Program

- Key program for early science and early feedback
- Started in early 2020
- Invited 14 applications from various scientific domains
 - Aimed for applications that could use JUWELS Booster at scale
 - Some teams already use JUWELS Cluster, others new
- **Offer:** Use JUWELS Booster before general access; **Request:** Help improve system, compute-time allocation
- Endeavor of many parts in **JSC** and beyond
 - **NVIDIA Application Lab:** Steering, GPU optimization, application support, system support
 - Application support, Simulation Labs
 - Performance Optimisation and Productivity team
 - System operations team
 - Vendors: NVIDIA, ParTec, Atos

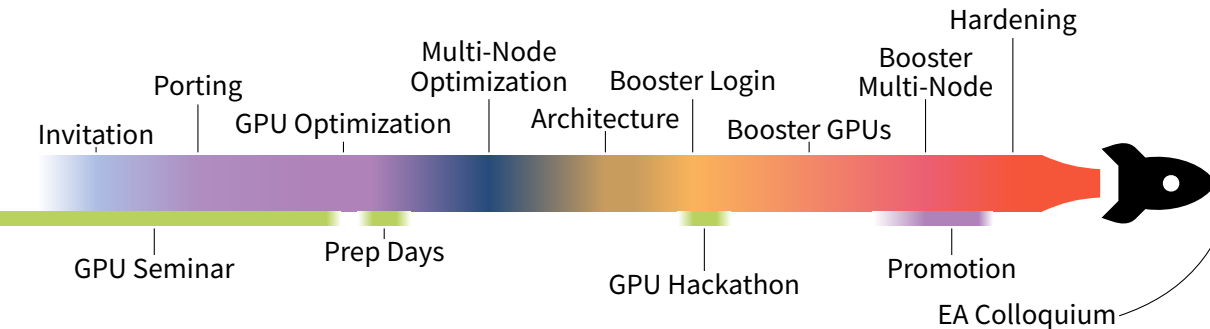
Timeline to Booster

- Preparation Timeline



Timeline to Booster

- Preparation Timeline
- Additionally: events



Applications I

Climate/Meteo/Hydro (*ESM*)

DeepACF  High-resolution Weather Forecast Based on Deep Learning  Lib:DL

 JSC: Bing Gong, Michael Langguth, Amirpasha Mozaffari, Martin Schultz, Scarlet Stadtler

ICON  Next-Generation Physical Weather and Climate Models  OpenACC

 MPI Met: Luis Kornbluh; NVIDIA: Dmitry Alexeev

MPTRAC  Massive Parallel Trajectory Calculations of Volcanic Emissions  OpenACC

 JSC: Sabine Griebach, Lars Hoffmann

ParFlow  Surface, Soil, Ground Water Flow  CUDA C

 IBG-3: Jaro Hokkanen, Stefan Kollet

Biological Matter

Amber  Drug Binding over Biologically Relevant Timescales (MD)  Lib

 JSC/HHU: Holger Gohlke, Christopher Pfleger, Michele Bonus

SOMA  Kinetics of Nanomaterial Formation (Soft Matter)  OpenACC

 U Göttingen: Ludwig Schneider, Niklas Blagojevic

Applications II

- PIConGPU**  Plasma Simulations for Next Generation Particle Accelerators (Plasma)  CUDA C++
 HZDR: Alexander Debus, Anton Lebedev, Rene Widera, Michael Bussmann
- JUQCS-G**  Simulating Universal Quantum Computer (Quantum)  CUDA Fortran
 JSC: Hans De Raedt, Kristel Michiels, Dennis Willsch
- E-train**  Understanding Learning Processes in Brain (Neuro)  Lib:DL
 U Graz: Franz Scherr, Wolfgang Maass; U Sussex: James Knight; INM-6: Sacha van Albada
- NBODY6++GPU**  Dense Star Clusters and Gravitational Waves (Astro)  CUDA Fortran
 U Heidelberg: Rainer Spurzem

Lattice QCD

- Bonn**  Flavour Singlet Structure of Hadrons
 Lib:QUDA
 U Bonn: Simone Bacchio, Bartosz Kostrzewa, Carsten Urbach
- Wuppertal**  SignQCD – Studying the Hottest Man-made Liquid  Lib:QUDA
 U Wuppertal: Szabolcs Borsányi, Kalman Szabo
- Bielefeld**  HotQCD – Studying Extreme States of Matter  CUDA C++
 U Bielefeld: Christian Schmit, Dennis Bollweg, Frithjof Karsch
- Regensburg**  Baryons with Charm  Lib:Grid
 Peter Boyle, Christoph Lehner, Gunnar Bali, Sara Collins

Applications II

PIConGPU  Plasma Simulations for Next Generation

Particle Accelerators (Plasma)  CUDA C++

 HZDR: Alexander Debus, Anton Lebedev,
Rene Widera, Michael Bussmann

JUQCS-G  Simulating Universal Quantum Computer

(Quantum)  CUDA Fortran

 JSC: Hans De Raedt, K
Dennis Willsch

E-train  Understanding Learning Processes in

Brain (Neuro)  Lib:DL

 U Graz: Franz Scherr, Wolfgang Maass; U
Sussex: James Knight; INM-6: Sacha van
Albada

NBODY6++GPU  Dense Star Clusters and

Gravitational Waves (Astro)  CUDA Fortran

 U Heidelberg: Rainer Spurzem

Lattice QCD

Bonn  Flavour Singlet Structure of Hadrons

 Lib:QUDA

 U Bonn: Simone Bacchio, Bartosz
Kostrzewa, Carsten Urbach

Wuppertal  SignQCD – Studying the Hottest

Quark-matter Liquid  Lib:QUDA

Wuppertal: Szabolcs Borsányi,
Kalman Szabo

Bielefeld  HotQCD – Studying Extreme States of
Matter  CUDA C++

 U Bielefeld: Christian Schmit, Dennis
Bollweg, Frithjof Karsch

Regensburg  Baryons with Charm  Lib:Grid

 Peter Boyle, Christoph Lehner,
Gunnar Bali, Sara Collins

→ Details on each app online 

Early Experiences

System-Related

Feedback during EA Program

Issues

- Performance fluctuations (GPU, node, network)
- OpenMPI segmentation violations
- NCCL hangs
- NVHPC Fortran compiler bugs
- UCX configuration (caches)
- PCIe switch bi-directional bandwidth
- PCIe device crashes
- I/O subsystem maturity

Feedback during EA Program

Issues

- Performance fluctuations (GPU, node, network)
- OpenMPI segmentation violations
- NCCL hangs
- NVHPC Fortran compiler bugs
- UCX configuration (caches)
- PCIe switch bi-directional bandwidth
- PCIe device crashes
- I/O subsystem maturity

Peculiarities

- Software versions
- AMD CPUs / NUMA domains
- PCIe switch
- GPU device affinity
- Network design (DragonFly+)

Feedback during EA Program

Issues

- Performance fluctuations (GPU, node, network)
- OpenMPI segmentation violations
- NCCL hangs
- NVHPC Fortran compiler bugs
- UCX configuration (caches)
- PCIe switch bi-directional bandwidth
- PCIe device crashes
- I/O subsystem maturity

Peculiarities

- Software versions
- AMD CPUs / NUMA domains
- PCIe switch
- GPU device affinity
- Network design (DragonFly+)

AMD EPYC Rome

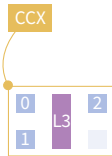
Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies

AMD EPYC Rome

Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies
 - 3 cores Core-Complex (CCX), shared L3 (*max 4 cores*)



AMD EPYC Rome

Some Processor Basics

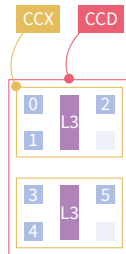
- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies

3 cores

Core-Complex (CCX), shared L3 (*max 4 cores*)

2 CCXs

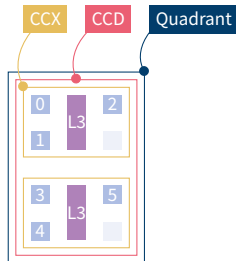
Core Complex Die (CCD)



AMD EPYC Rome

Some Processor Basics

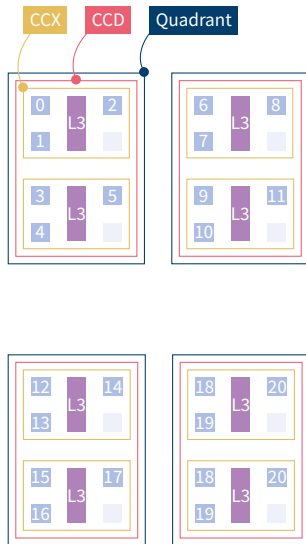
- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies
 - 3 cores Core-Complex (CCX), shared L3 (*max 4 cores*)
 - 2 CCXs Core Complex Die (CCD)
 - 1 CCD 1 quadrant (*max 2 CCDs per quadrant*)



AMD EPYC Rome

Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
 - EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies
- | | |
|-------------|--|
| 3 cores | Core-Complex (CCX), shared L3 (<i>max 4 cores</i>) |
| 2 CCXs | Core Complex Die (CCD) |
| 1 CCD | 1 quadrant (<i>max 2 CCDs per quadrant</i>) |
| 4 quadrants | 1 socket; NPS-4: 4 NUMA domains per socket |



AMD EPYC Rome

Some Processor Basics

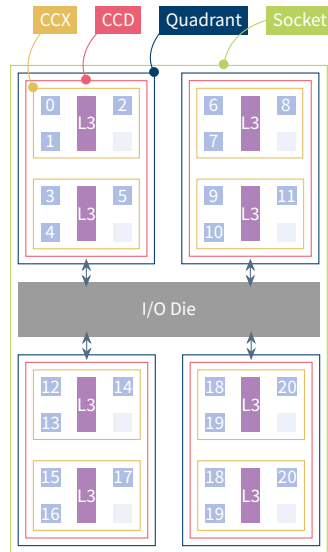
- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies
 - 3 cores Core-Complex (CCX), shared L3 (*max 4 cores*)
 - 2 CCXs Core Complex Die (CCD)
 - 1 CCD 1 quadrant (*max 2 CCDs per quadrant*)
 - 4 quadrants 1 socket; NPS-4: 4 NUMA domains per socket
 - 2 sockets 1 node (*only 1 shown*)



AMD EPYC Rome

Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
 - EPYC processor: Built as Multi-Chip Module
 - Many building blocks, hierarchies
- | | |
|-------------|--|
| 3 cores | Core-Complex (CCX), shared L3 (<i>max 4 cores</i>) |
| 2 CCXs | Core Complex Die (CCD) |
| 1 CCD | 1 quadrant (<i>max 2 CCDs per quadrant</i>) |
| 4 quadrants | 1 socket; NPS-4: 4 NUMA domains per socket |
| 2 sockets | 1 node (<i>only 1 shown</i>) |
| I/O Die | Connections between quadrants and outside |

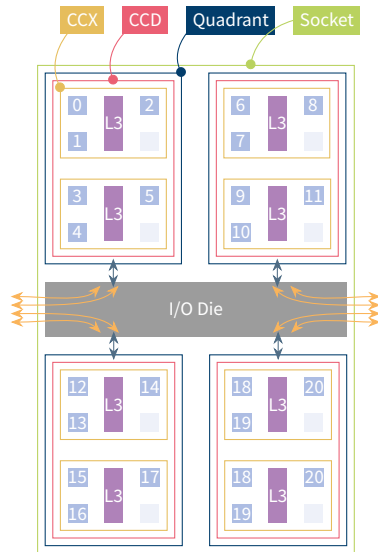


AMD EPYC Rome

Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies

3 cores	Core-Complex (CCX), shared L3 (<i>max 4 cores</i>)
2 CCXs	Core Complex Die (CCD)
1 CCD	1 quadrant (<i>max 2 CCDs per quadrant</i>)
4 quadrants	1 socket; NPS-4: 4 NUMA domains per socket
2 sockets	1 node (<i>only 1 shown</i>)
I/O Die	Connections between quadrants and outside
RAM	8 memory channels, 2 per quadrant



AMD EPYC Rome

Some Processor Basics

- JUWELS Booster: AMD EPYC Rome 7402 (24 core (SMT-2) $\times$ 2 sockets)
- EPYC processor: Built as Multi-Chip Module
→ Many building blocks, hierarchies

3 cores Core-Complex (CCX), shared L3 (*max 4 cores*)

2 CCXs Core Complex Die (CCD)

1 CCD 1 quadrant (*max 2 CCDs per quadrant*)

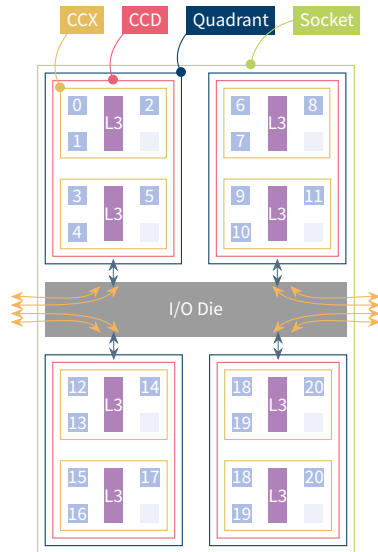
4 quadrants 1 socket; NPS-4: 4 NUMA domains per socket

2 sockets 1 node (*only 1 shown*)

I/O Die Connections between quadrants and outside

RAM 8 memory channels, 2 per quadrant

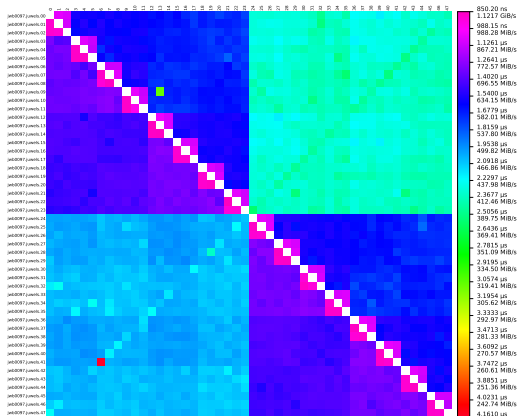
⇒ **Complex topology!**



EYPC Core-to-Core

Linktest

- Test core-to-core connections with JSC's Linktest
 - Linktest: MPI Ping Pong usually for large machines
 - Here: Within a node
 - EPYC structure clearly visible
- ⇒ Thread placement very important, especially for L3-using application
- We saw 30 % performance increase by proper placement
- *Configuration: GCC, ParaStationMPI, serial*

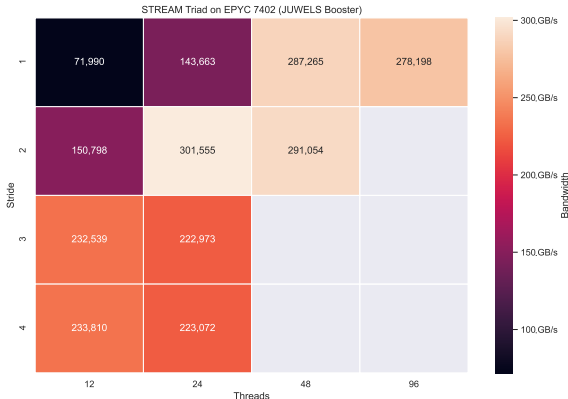


Memory

STREAM

- EPYC Rome: 8 memory channels, 2 per quadrant
 - Thread placement important for memory performance
 - STREAM benchmark! *all in Appendix*
- ⇒ Thread placement very important, also for memory-intensive application

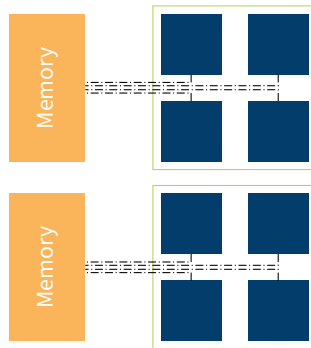
*STREAM config: AOCC 3.0.0, -O3 -mmodel=large
-ffp-contract=fast -fnt-store, array size:
2500000000, best of 200*



PCIe Affinity

3 C 2 CCX 1 CCD 4 Q 2 S

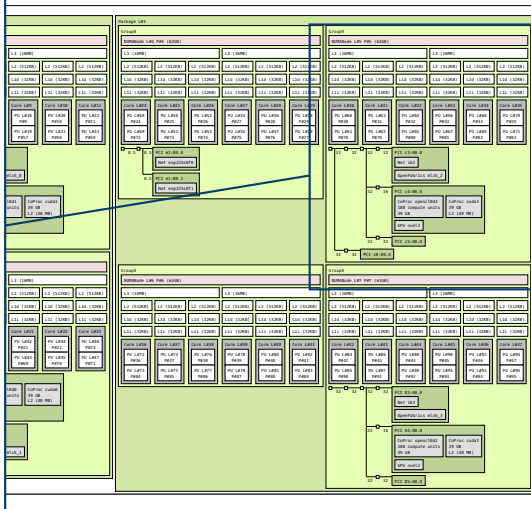
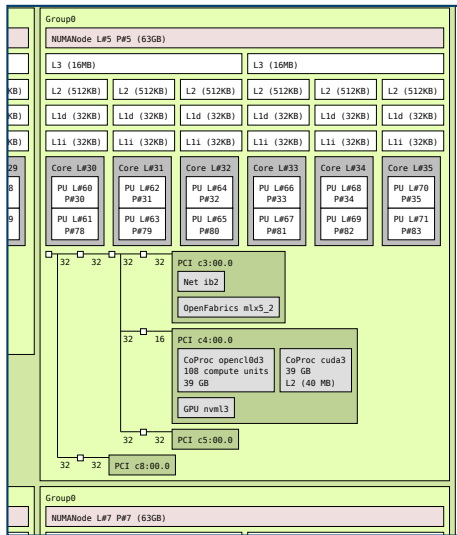
- PCIe via I/O die → also hierarchy
- GPUs connected via PCIe (through PCIe switch)





PCIe Affinity

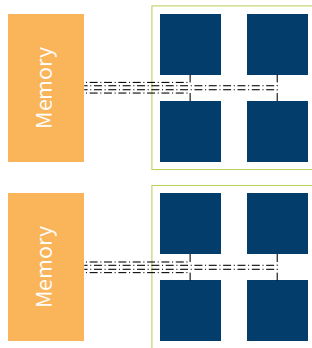
2 CCD 1 CCD 4 Q 2 S



PCIe Affinity

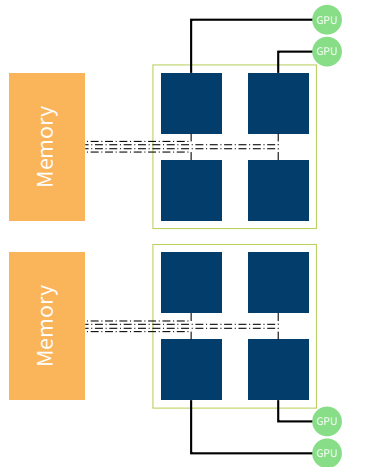
3 C 2 CCX 1 CCD 4 Q 2 S

- PCIe via I/O die → also hierarchy
- GPUs connected via PCIe (through PCIe switch)
- PCIe 4.0 lanes: 2×16 , each 16 connected to 1 quadrant



PCIe Affinity

- PCIe via I/O die → also hierarchy
 - GPUs connected via PCIe (through PCIe switch)
 - PCIe 4.0 lanes: 2×16 , each 16 connected to 1 quadrant
- *True GPU affinity only by half of chiplets*
But impact application-dependent and possibly quite low



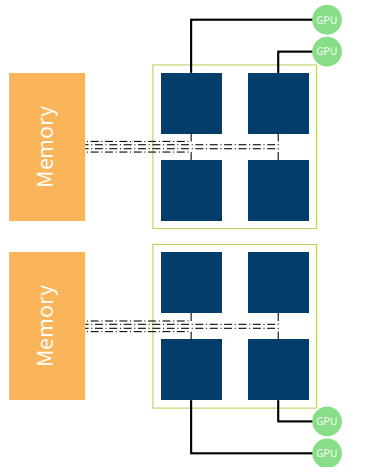
PCIe Affinity

- PCIe via I/O die → also hierarchy
- GPUs connected via PCIe (through PCIe switch)
- PCIe 4.0 lanes: 2×16 , each 16 connected to 1 quadrant

→ *True GPU affinity only by half of chiplets*

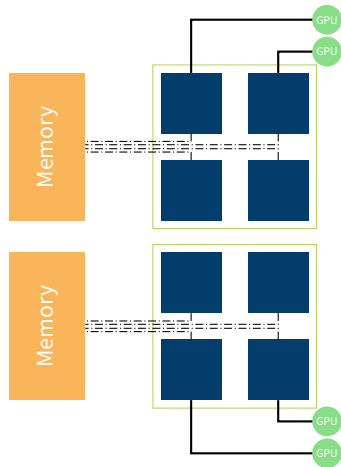
But impact application-dependent and possibly quite low

- InfiniBand HCA adapters also via PCIe switch
 - Same affinity considerations
 - 1:1 relation between HCA and GPU



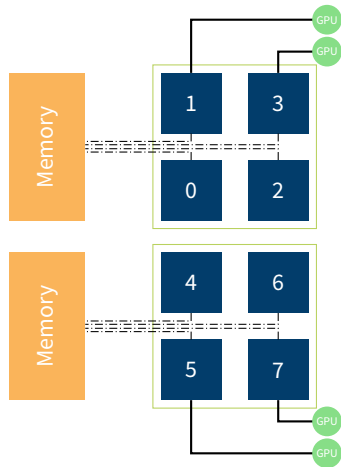
NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!



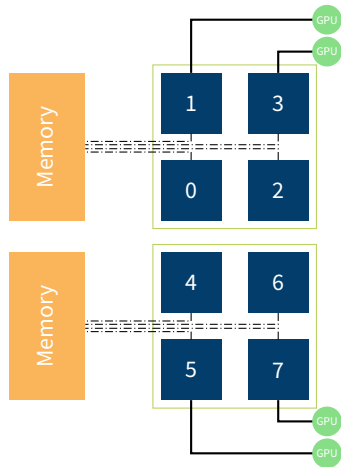
NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!



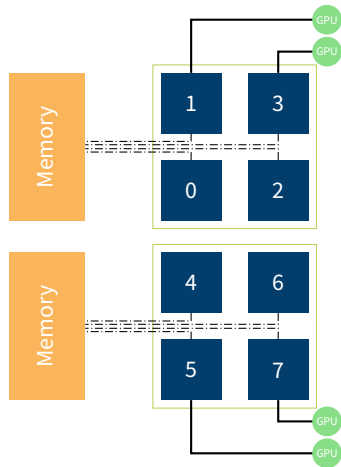
NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!
- Right! 1, 3, 5, 7



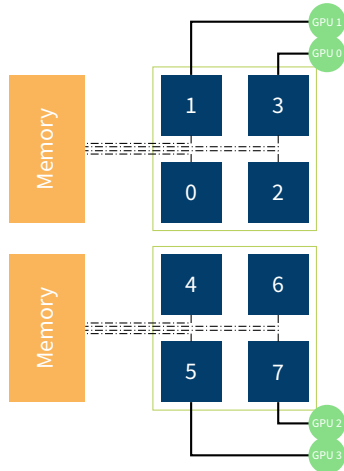
NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!
- Right! 1, 3, 5, 7
- *But GPU 0 is not attached to NUMA domain 0...*



NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!
- Right! 1, 3, 5, 7
- *But GPU 0 is not attached to NUMA domain 0...*

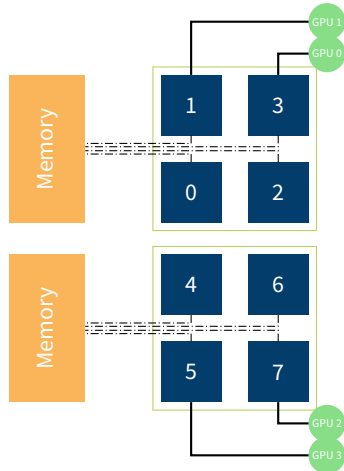


NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers of domains with affinity!
- Right! 1, 3, 5, 7
- *But GPU 0 is not attached to NUMA domain 0...*

- Complete affinity table

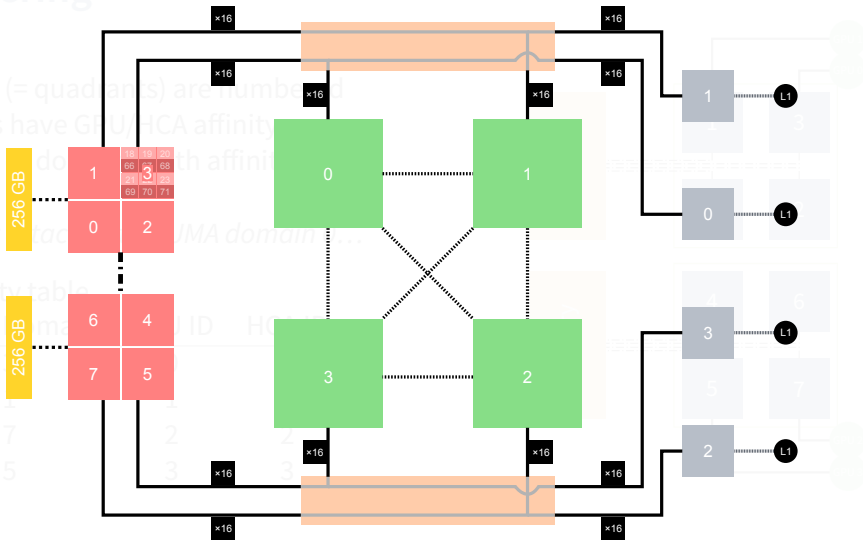
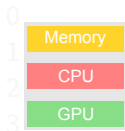
Rank	NUMA Domain	GPU ID	HCA ID
0	3	0	0
1	1	1	1
2	7	2	2
3	5	3	3



NUMA Numbering

- NUMA domains (= quadrants) are numbered
- Not all domains have GPU/HCA affinity
- Guess numbers for domains with affinity
- Right! 1, 3, 5, 7
- But GPU 0 is not in domain 0 ...
- Complete affinity table

Rank NUMA domain GPU ID HCA ID



Affinity

- Changes to environment/Slurm to match node topology
- Recipe: GPU first, CPU second

Affinity

- Changes to environment/Slurm to match node topology
- Recipe: GPU first, CPU second
- Example Slurm task distribution for 5 tasks, each 1 core:

Task 1 First core of NUMA domain 3

Task 2 First core of NUMA domain 1

Task 3 First core of NUMA domain 7

Task 4 First core of NUMA domain 5

Task 5 First core of NUMA domain 2

Affinity

- Changes to environment/Slurm to match node topology
- Recipe: GPU first, CPU second
- Example Slurm task distribution for 5 tasks, each 1 core:
 - Task 1 First core of NUMA domain 3
 - Task 2 First core of NUMA domain 1
 - Task 3 First core of NUMA domain 7
 - Task 4 First core of NUMA domain 5
 - Task 5 First core of NUMA domain 2
- Fill up NUMA domain with `--cpus-per-task N`

Affinity

- Changes to environment/Slurm to match node topology
- Recipe: GPU first, CPU second
- Example Slurm task distribution for 5 tasks, each 1 core:
 - Task 1 First core of NUMA domain 3
 - Task 2 First core of NUMA domain 1
 - Task 3 First core of NUMA domain 7
 - Task 4 First core of NUMA domain 5
 - Task 5 First core of NUMA domain 2
- Fill up NUMA domain with `--cpus-per-task N`
- Realized by CPU affinity masks, environment variables (GPU, HCA)



Affinity

- Changes to environment/Slurm to match node topology
- Recipe: GPU first, CPU second
- Example Slurm task distribution for 5 tasks, each 1 core:
 - Task 1 First core of NUMA domain 3
 - Task 2 First core of NUMA domain 1
 - Task 3 First core of NUMA domain 7
 - Task 4 First core of NUMA domain 5
 - Task 5 First core of NUMA domain 2
- Fill up NUMA domain with `--cpus-per-task N`
- Realized by CPU affinity masks, environment variables (GPU, HCA)

Affinity

CPU Affinity Mask I

- *PSSlurm* sets CPU affinity masks with *GPU first*

Affinity

CPU Affinity Mask I

- PSSLurm sets CPU affinity masks with *GPU first*
- CPU mask example for 5 tasks, each 1 core:

```
$ srun --cpu-bind=verbose -n 5 bash -c "" |& sort
cpu_bind=THREADS - jwb0001, task 0 0 [18049]: mask 0x40000 set
cpu_bind=THREADS - jwb0001, task 1 1 [18050]: mask 0x40 set
cpu_bind=THREADS - jwb0001, task 2 2 [18055]: mask 0x400000000000 set
cpu_bind=THREADS - jwb0001, task 3 3 [18059]: mask 0x40000000 set
cpu_bind=THREADS - jwb0001, task 4 4 [18061]: mask 0x1000 set
```

Affinity

CPU Affinity Mask I

- PSSlurm sets CPU affinity masks with *GPU first*

- CPU mask example for 5 tasks, each 1 core:

```
$ srun --cpu-bind=verbose -n 5 bash -c "" |& sort
```

```
cpu_bind=THREADS - jwb0001, task 0 0 [18049]: mask 0x40000 set
```

```
cpu_bind=THREADS - jwb0001, task 1 1 [18050]: mask 0x40 set
```

```
cpu_bind=THREADS - jwb0001, task 2 2 [18055]: mask 0x400000000000 set
```

```
cpu_bind=THREADS - jwb0001, task 3 3 [18059]: mask 0x40000000 set
```

```
cpu_bind=THREADS - jwb0001, task 4 4 [18061]: mask 0x1000 set
```

- Visualize masks in binary representation (*SMT-2 omitted*)

hex2bin

```
000000 000000 000000 100000 000000 000000 000000 000000 # 0x40000, GPU 0
000000 100000 000000 000000 000000 000000 000000 000000 # 0x40, GPU 1
000000 000000 000000 000000 000000 000000 000000 100000 # 0x400000000000, GPU 2
000000 000000 000000 000000 000000 100000 000000 000000 # 0x40000000, GPU 3
000000 000000 100000 000000 000000 000000 000000 000000 # 0x1000, no GPU
```


Affinity

CPU Affinity Mask II

- PSSlurm sets CPU affinity masks with *GPU first*
- CPU mask example for 2 tasks, each 2 cores:

```
$ srun --cpu-bind=verbose -n 2 --cpus-per-task 2 bash -c "" |& sort
cpu_bind=THREADS - jwb0001, task 0 0 [18402]: mask 0xc0000 set
cpu_bind=THREADS - jwb0001, task 1 1 [18403]: mask 0xc0 set
```

hex2bin

```
000000 000000 000000 110000 000000 000000 000000 000000 # 0xc0000, GPU 0
000000 110000 000000 000000 000000 000000 000000 000000 # 0xc0, GPU 1
```

- *PSSlurm* sets `$CUDA_VISIBLE_DEVICES` to associated GPU

Affinity

GPU Affinity

- PSSlurm sets `$CUDA_VISIBLE_DEVICES` to associated GPU
- GPU affinity for 2 tasks:

```
$ srun -n 2 --cpu-bind=verbose env | grep 'PMI_RANK\|CUDA_VIS'  
cpu_bind=THREADS - jwb0001, task 0 0 [18307]: mask 0x40000 set  
cpu_bind=THREADS - jwb0001, task 1 1 [18309]: mask 0x40 set  
PMI_RANK=0  
CUDA_VISIBLE_DEVICES=0  
PMI_RANK=1  
CUDA_VISIBLE_DEVICES=1
```

Affinity

GPU Affinity

- *PSSlurm* sets `$CUDA_VISIBLE_DEVICES` to associated GPU

- GPU affinity for 2 tasks:

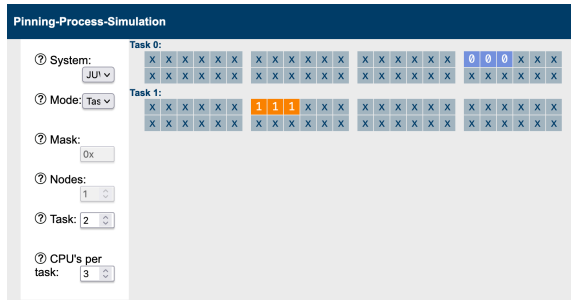
```
$ srun -n 2 --cpu-bind=verbose env | grep 'PMI_RANK\|CUDA_VIS'  
cpu_bind=THREADS - jwb0001, task 0 0 [18307]: mask 0x40000 set  
cpu_bind=THREADS - jwb0001, task 1 1 [18309]: mask 0x40 set  
PMI_RANK=0  
CUDA_VISIBLE_DEVICES=0  
PMI_RANK=1  
CUDA_VISIBLE_DEVICES=1
```

- Special case: GPU affinity for exactly 1 task:

```
$ srun -n 1 env | grep 'PMI_RANK\|CUDA_VIS'  
PMI_RANK=0  
CUDA_VISIBLE_DEVICES=0,1,2,3
```

Handling Affinity Handling

- Node affinity now more important than ever! (*maybe...*)
- JSC provides tools and documentation as first-level support!
- Mask generator
- CLI mask tool 
- Extensive system documentation 



apps.fz-juelich.de/jsc/llview/pinning/

DragonFly+ Topology

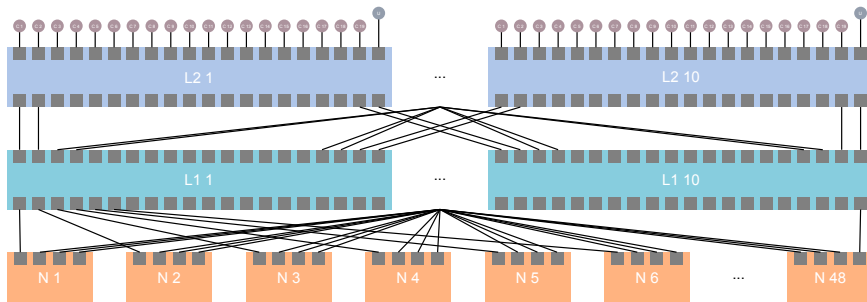
Overview

- 3744 end points; adaptive routing
- HDR200: 200 Gbit/s per link per direction
- Dragonfly+ Topology selected as good way to balance factors (space, investment, performance)
- Two-level topology: local (in-cell), non-local (inter-cell)
- Still learning practical implications of topology

DragonFly+ Topology

In-Cell Topology

- In-cell: full fat-tree in 2 levels
 - 48 nodes, each 4 links (*strided to 4 L1 switches*)
 - 10 L1 switches, 48 port (*each L1 switch: 2 links to L2, strided to 10 L2 switches*)
 - 10 L2 switches, 48 port (*each L2 switch: 1 link to L2 switch of all other cells, 1 link to cluster*)

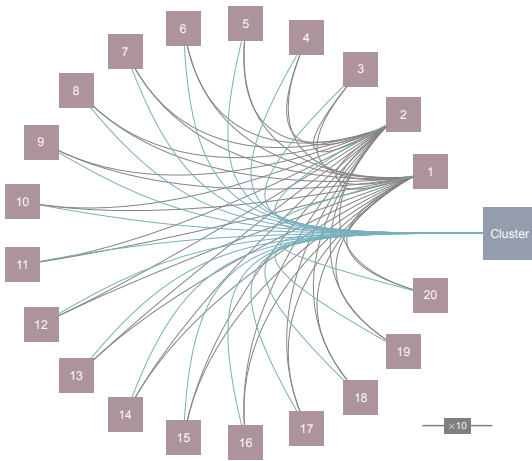


DragonFly+ Topology

Intra-Cell Topology

- Inter-cell: 20 cells
 - 10 links between each pair of cells
 - 10 links per cell to JUWELS Cluster
 - Filesystem access via cell 20
 - Bi-section bandwidth between N cells:

$$\mathcal{B}(N) = \lfloor (N/2)^2 \rfloor \times (10 \times \text{bw}_1)$$



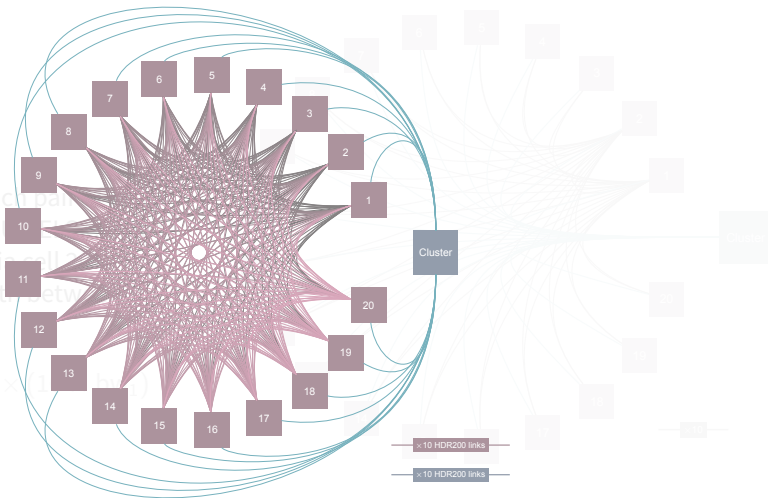
DragonFly+ Topology

Intra-Cell Topology

- Inter-cell: 20 cells

- 10 links between each pair of cells
- 10 links per cell to Jülich cluster
- Filesystem access via cell 1
- Bi-section bandwidth between cells:

$$B(N) = \lfloor (N/2)^2 \rfloor \times 10 \text{ Gb/s}$$



Interactive Job Reporting

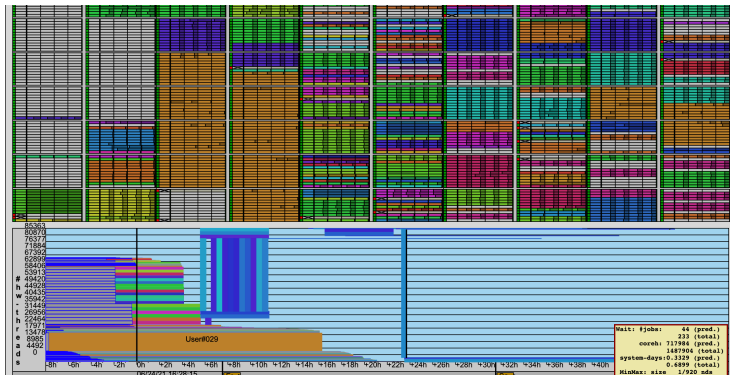
LLview



- LLview: JSC's online tool for scheduler statistics
- Already around for some time, now extended with GPU-related info
- System overview (nodes, GPUs; timeline)

→ llview.fz-juelich.de

- JSC: Access via JuDoor

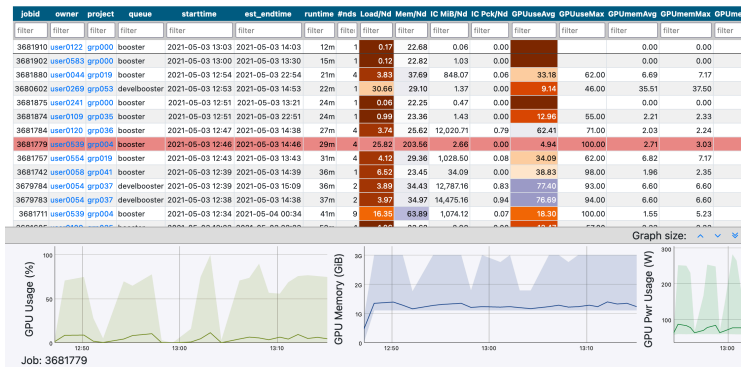


Interactive Job Reporting

LLview



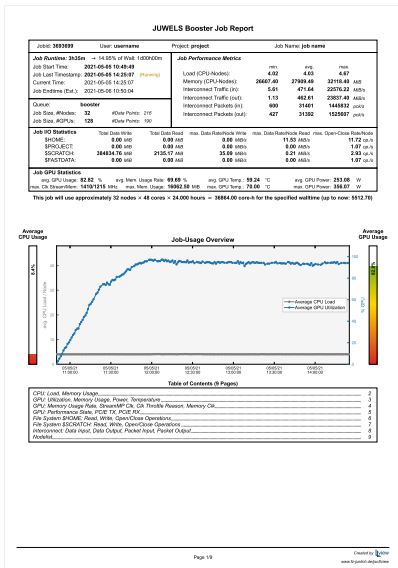
- LLview: JSC's online tool for scheduler statistics
- Already around for some time, now extended with GPU-related info
- System overview (nodes, GPUs; timeline)
- Job focus (many metrics)
→ llview.fz-juelich.de
- JSC: Access via JuDoor



PDF Job Reports

From LLview

- **New:** Detailed PDF job reports
- Automatically generated
- Many metrics, graphs
- v2.0 live since yesterday
- Via LLview job view



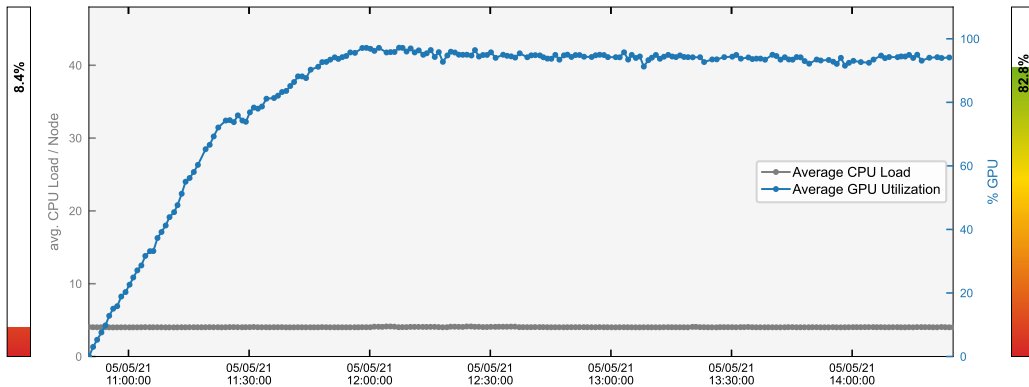
Jobid: 3693699		User: username	Project: project		Job Name: job name	
Job Runtime: 3h35m → 14.95% of Wall: 1d00h00m			Job Performance Metrics			
Job Start Time: 2021-05-05 10:49:49				min.	avg.	max.
Job Last Timestamp: 2021-05-05 14:25:07 (Running)			Load (CPU-Nodes):	4.02	4.03	4.67
Current Time: 2021-05-05 14:25:07			Memory (CPU-Nodes):	26607.40	27909.49	32118.40 MiB
Job Endtime (Est.): 2021-05-06 10:50:04			Interconnect Traffic (in):	5.61	471.64	22576.22 MiB/s
			Interconnect Traffic (out):	1.13	462.61	23837.40 MiB/s
Queue: booster			Interconnect Packets (in):	600	31401	1445832 pck/s
Job Size, #Nodes: 32 #Data Points: 216			Interconnect Packets (out):	427	31392	1525607 pck/s
Job Size, #GPUs: 128 #Data Points: 190						
Job I/O Statistics		Total Data Write	Total Data Read	max. Data Rate/Node Write	max. Data Rate/Node Read	max. Open-Close Rate/Node
\$HOME:		0.00 MiB	0.00 MiB	0.00 MiB/s	11.53 MiB/s	11.72 op./s
\$PROJECT:		0.00 MiB	0.00 MiB	0.00 MiB/s	0.00 MiB/s	1.07 op./s
\$SCRATCH:		384834.76 MiB	2135.17 MiB	35.09 MiB/s	0.21 MiB/s	2.93 op./s
\$FASTDATA:		0.00 MiB	0.00 MiB	0.00 MiB/s	0.00 MiB/s	1.07 op./s
Job GPU Statistics						
avg. GPU Usage: 82.82 %		avg. Mem. Usage Rate: 69.69 %		avg. GPU Temp.: 59.24 °C	avg. GPU Power: 253.08 W	
max. Clk Stream/Mem: 1410/1215 MHz		max. Mem. Usage: 16062.50 MiB		max. GPU Temp.: 70.00 °C	max. GPU Power: 356.07 W	

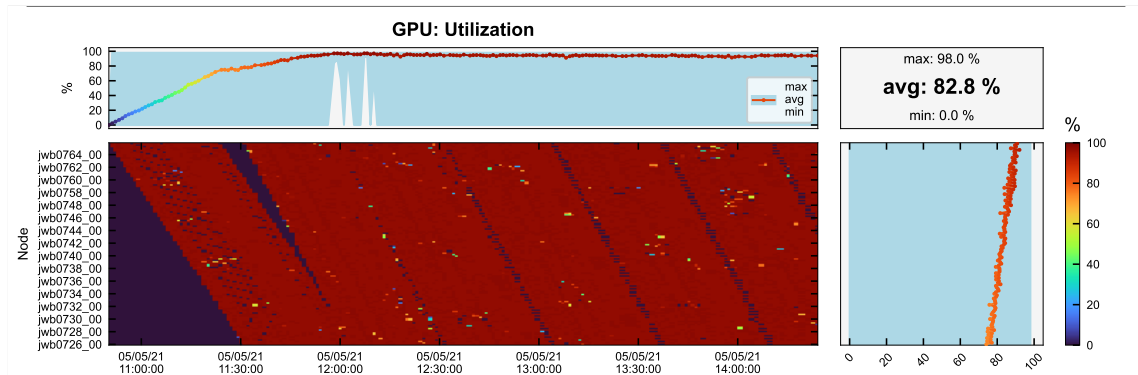
This job will use approximately 32 nodes × 48 cores × 24.000 hours = 36864.00 core-h for the specified walltime (up to now: 5512.70)

Average
CPU Usage

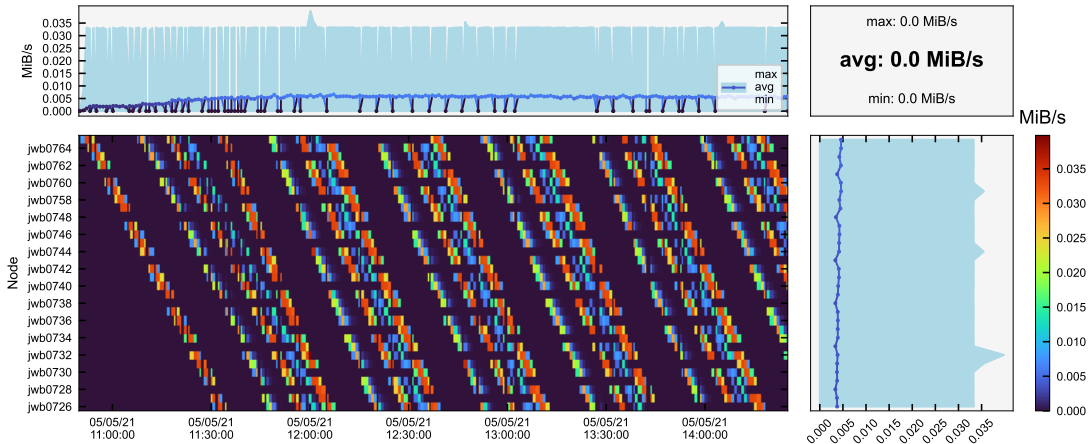
Job-Usage Overview

Average
GPU Usage





File System \$SCRATCH: Read



Nodelist

1 jwb0726 1 GPU0: 74.1% 14.5GiB 1332MHz 2 GPU1: 77.1% 14.5GiB 1341MHz 3 GPU2: 74.5% 14.5GiB 1334MHz 4 GPU3: 76.5% 14.5GiB 1339MHz Interconnect group: 11	2 jwb0727 5 GPU0: 74.5% 14.5GiB 1332MHz 6 GPU1: 76.8% 14.5GiB 1344MHz 7 GPU2: 75.2% 14.5GiB 1339MHz 8 GPU3: 77.1% 14.5GiB 1345MHz Interconnect group: 11	3 jwb0728 9 GPU0: 74.8% 14.5GiB 1336MHz 10 GPU1: 78.3% 14.5GiB 1346MHz 11 GPU2: 75.0% 14.5GiB 1336MHz 12 GPU3: 77.4% 14.5GiB 1344MHz Interconnect group: 11	4 jwb0729 13 GPU0: 76.9% 14.5GiB 1347MHz 14 GPU1: 77.3% 14.5GiB 1346MHz 15 GPU2: 77.6% 14.5GiB 1346MHz 16 GPU3: 77.8% 14.5GiB 1346MHz Interconnect group: 11	5 jwb0730 17 GPU0: 76.1% 14.5GiB 1342MHz 18 GPU1: 78.2% 14.5GiB 1346MHz 19 GPU2: 78.7% 14.5GiB 1347MHz 20 GPU3: 77.0% 14.5GiB 1342MHz Interconnect group: 11	6 jwb0731 21 GPU0: 77.5% 14.5GiB 1342MHz 22 GPU1: 78.2% 14.5GiB 1346MHz 23 GPU2: 78.8% 14.5GiB 1347MHz 24 GPU3: 79.1% 14.5GiB 1349MHz Interconnect group: 11
7 jwb0732 25 GPU0: 78.6% 14.5GiB 1349MHz 26 GPU1: 75.8% 14.5GiB 1342MHz 27 GPU2: 79.1% 14.5GiB 1349MHz 28 GPU3: 78.9% 14.5GiB 1349MHz Interconnect group: 11	8 jwb0733 29 GPU0: 78.7% 14.5GiB 1350MHz 30 GPU1: 78.1% 14.5GiB 1344MHz 31 GPU2: 79.4% 14.5GiB 1351MHz 32 GPU3: 79.8% 14.5GiB 1349MHz Interconnect group: 11	9 jwb0734 33 GPU0: 78.4% 14.5GiB 1346MHz 34 GPU1: 79.3% 14.5GiB 1347MHz 35 GPU2: 80.8% 14.5GiB 1352MHz 36 GPU3: 80.4% 14.5GiB 1352MHz Interconnect group: 11	10 jwb0735 37 GPU0: 79.7% 14.5GiB 1351MHz 38 GPU1: 79.3% 14.5GiB 1347MHz 39 GPU2: 81.8% 14.5GiB 1356MHz 40 GPU3: 80.4% 14.5GiB 1351MHz Interconnect group: 11	11 jwb0736 41 GPU0: 80.9% 14.5GiB 1354MHz 42 GPU1: 79.3% 14.5GiB 1347MHz 43 GPU2: 80.8% 14.5GiB 1352MHz 44 GPU3: 79.2% 14.5GiB 1349MHz Interconnect group: 11	12 jwb0737 45 GPU0: 80.8% 14.5GiB 1354MHz 46 GPU1: 79.3% 14.5GiB 1347MHz 47 GPU2: 81.3% 14.5GiB 1354MHz 48 GPU3: 81.3% 14.5GiB 1354MHz Interconnect group: 12
13 jwb0738 49 GPU0: 81.2% 14.5GiB 1356MHz 50 GPU1: 79.2% 14.5GiB 1354MHz 51 GPU2: 80.9% 14.5GiB 1361MHz 52 GPU3: 82.2% 14.5GiB 1361MHz Interconnect group: 12	14 jwb0739 53 GPU0: 81.6% 14.5GiB 1359MHz 54 GPU1: 80.3% 14.5GiB 1359MHz 55 GPU2: 81.3% 14.5GiB 1361MHz 56 GPU3: 82.4% 14.5GiB 1362MHz Interconnect group: 12	15 jwb0740 57 GPU0: 81.6% 14.6GiB 1356MHz 58 GPU1: 81.1% 14.6GiB 1357MHz 59 GPU2: 81.8% 14.6GiB 1357MHz 60 GPU3: 82.4% 14.6GiB 1361MHz Interconnect group: 12	16 jwb0741 61 GPU0: 82.1% 14.6GiB 1359MHz 62 GPU1: 83.0% 14.6GiB 1359MHz 63 GPU2: 82.3% 14.6GiB 1362MHz 64 GPU3: 83.6% 14.6GiB 1364MHz Interconnect group: 12	17 jwb0742 65 GPU0: 84.3% 14.6GiB 1364MHz 66 GPU1: 83.7% 14.6GiB 1362MHz 67 GPU2: 83.9% 14.6GiB 1362MHz 68 GPU3: 84.4% 14.6GiB 1364MHz Interconnect group: 12	18 jwb0743 69 GPU0: 82.1% 14.6GiB 1359MHz 70 GPU1: 83.6% 14.6GiB 1362MHz 71 GPU2: 81.4% 14.6GiB 1362MHz 72 GPU3: 84.4% 14.6GiB 1369MHz Interconnect group: 12
19 jwb0744 73 GPU0: 83.9% 14.6GiB 1362MHz 74 GPU1: 83.9% 14.6GiB 1364MHz 75 GPU2: 83.4% 14.6GiB 1362MHz 76 GPU3: 84.1% 14.6GiB 1366MHz Interconnect group: 12	20 jwb0745 77 GPU0: 84.3% 14.6GiB 1366MHz 78 GPU1: 84.9% 14.6GiB 1366MHz 79 GPU2: 83.6% 14.6GiB 1367MHz 80 GPU3: 84.6% 14.6GiB 1367MHz Interconnect group: 12	21 jwb0746 81 GPU0: 84.3% 14.6GiB 1367MHz 82 GPU1: 84.3% 14.6GiB 1366MHz 83 GPU2: 85.8% 14.6GiB 1372MHz 84 GPU3: 81.6% 14.6GiB 1361MHz Interconnect group: 12	22 jwb0747 85 GPU0: 86.4% 14.6GiB 1371MHz 86 GPU1: 86.0% 14.6GiB 1369MHz 87 GPU2: 85.5% 14.6GiB 1367MHz 88 GPU3: 84.2% 14.6GiB 1366MHz Interconnect group: 12	23 jwb0748 89 GPU0: 87.5% 14.6GiB 1374MHz 90 GPU1: 85.3% 14.6GiB 1369MHz 91 GPU2: 86.8% 14.6GiB 1371MHz 92 GPU3: 84.4% 14.6GiB 1369MHz Interconnect group: 12	24 jwb0757 93 GPU0: 87.4% 14.6GiB 1374MHz 94 GPU1: 85.7% 14.6GiB 1371MHz 95 GPU2: 87.2% 14.6GiB 1376MHz 96 GPU3: 84.0% 14.6GiB 1372MHz Interconnect group: 12
25 jwb0758 97 GPU0: 89.4% 14.6GiB 1376MHz 98 GPU1: 86.0% 14.6GiB 1372MHz 99 GPU2: 88.7% 14.6GiB 1383MHz 100 GPU3: 83.8% 14.6GiB 1375MHz Interconnect group: 12	26 jwb0759 101 GPU0: 89.6% 14.6GiB 1380MHz 102 GPU1: 85.6% 14.6GiB 1372MHz 103 GPU2: 89.3% 14.6GiB 1384MHz 104 GPU3: 86.6% 14.6GiB 1376MHz Interconnect group: 12	27 jwb0760 105 GPU0: 89.4% 14.6GiB 1382MHz 106 GPU1: 85.9% 14.6GiB 1374MHz 107 GPU2: 88.7% 14.6GiB 1382MHz 108 GPU3: 87.7% 14.6GiB 1376MHz Interconnect group: 12	28 jwb0761 109 GPU0: 89.5% 14.6GiB 1385MHz 110 GPU1: 86.6% 14.6GiB 1374MHz 111 GPU2: 90.1% 14.6GiB 1385MHz 112 GPU3: 86.8% 14.6GiB 1376MHz Interconnect group: 12	29 jwb0762 113 GPU0: 88.8% 14.6GiB 1382MHz 114 GPU1: 86.9% 14.6GiB 1376MHz 115 GPU2: 87.6% 14.6GiB 1380MHz 116 GPU3: 90.0% 14.6GiB 1382MHz Interconnect group: 12	30 jwb0763 117 GPU0: 88.6% 14.6GiB 1382MHz 118 GPU1: 89.5% 14.6GiB 1380MHz 119 GPU2: 88.3% 14.6GiB 1382MHz 120 GPU3: 90.6% 14.6GiB 1387MHz Interconnect group: 12
		31 jwb0764 121 GPU0: 89.4% 14.6GiB 1384MHz 122 GPU1: 89.8% 14.6GiB 1382MHz 123 GPU2: 89.3% 14.6GiB 1383MHz 124 GPU3: 92.1% 14.6GiB 1389MHz Interconnect group: 12	32 jwb0765 125 GPU0: 87.4% 14.6GiB 1380MHz 126 GPU1: 90.2% 14.6GiB 1385MHz 127 GPU2: 89.6% 14.6GiB 1385MHz 128 GPU3: 90.8% 15.6GiB 1387MHz Interconnect group: 12		

Early Experiences

Early Experiences: Application-Related

Overview

- Some first results by users
- Mainly EA participants
- Most results preliminary
- Results partly on machine under construction

Early Experiences

System-Related

Aside: AMD EPYC Core Design

Intra-Node Affinities

Inter-Node Affinity

Job Reportings

Application-Related

SOMA

JUQMES

ParFlow

JUQCS

LQCD: Bonn

Others



Early Experiences

Application-Related: *Soft Matter: SOMA*

Soft Matter: SOMA

- **SOMA: Soft, coarse-grained Monte-Carlo Acceleration**

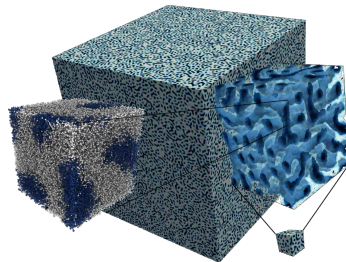
L. Schneider and M. Müller, *Comput. Phys. Commun.* 235C 463–476 (2019) and GPU Seminar Talk

- Kinetics of nanomaterial formation; multi-component polymer systems (battery materials, membranes, ...)
- Unique: Resolve details of polymer, but study lengths relevant to engineering

👥 Team: L. Schneider, N. Blagojevic, L. Pigard, M. Müller, et al

→ gitlab.com/InnocentBug/SOMA/

- C, OpenACC, MPI
- Frequent JUWELS user



GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN



SOMA
yes, soft matters



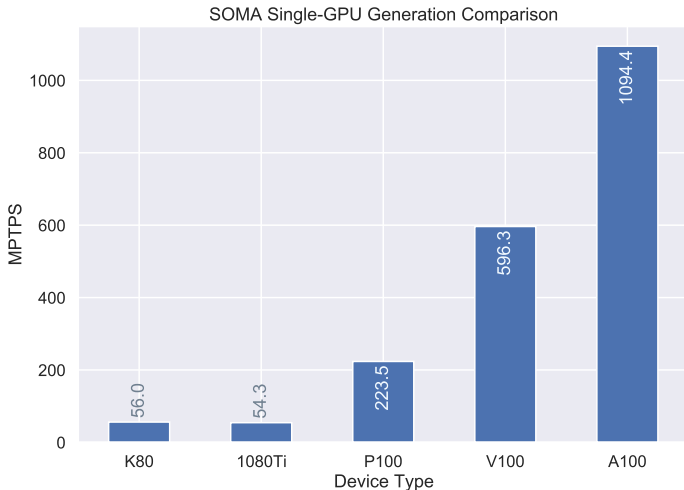
JÜLICH
Forschungszentrum

JÜLICH
SUPERCOMPUTING
CENTRE

Soft Matter: SOMA

Comparison of GPU Generations

- Long experience with various GPU architectures
- **Update to new generations early!**
- Some algorithmic changes between generations; also feature additions
- *PTPS: Particle Timesteps Per Second*



Soft Matter: SOMA

Kernel Comparison: Memory Chart

- Many random accesses

→ Benefit from larger L1, L2 caches

→ More FP64 throughput

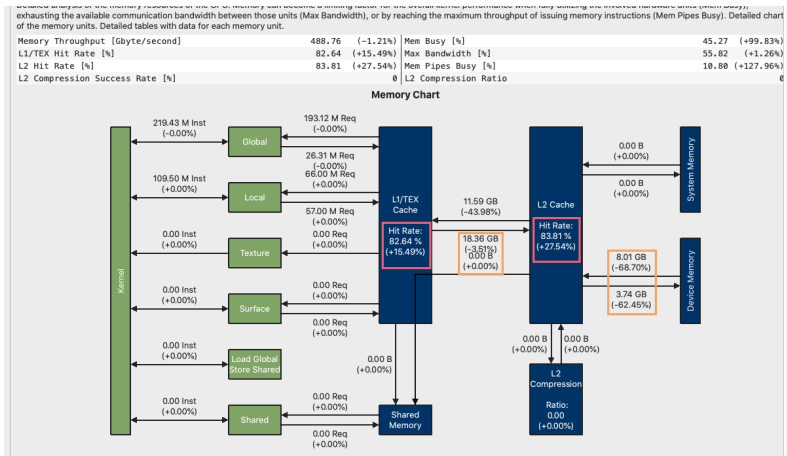
- Knock-on effect: less memory traffic

- Kernel runtime:

V100 25.8 ms

A100 21.5 ms

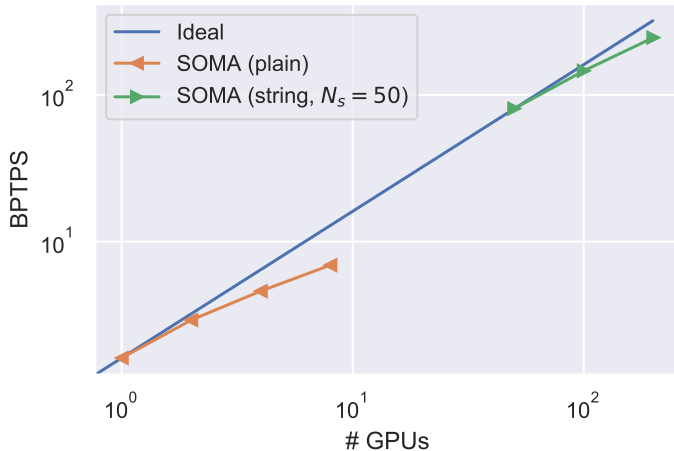
A100* 18.9 ms



Soft Matter: SOMA

New Method for Scaling

- **Scale of Booster: New algorithms, implementations with more scalability!**
- New project for Booster: *String* Method
- String-coupled SOMA ensemble simulation
- Master thesis of N. Blagojevic



Early Experiences

Application-Related: *JUQMES*

Quantum Computing: JUQMES

- **JUQMES:** Jülich Quantum Master Equation Simulator

D. Willsch, et al.

- Simulation of physical realizations of quantum computers

- Compute-intensive

- Team: Research group Quantum Information Processing

- C++, OpenACC, CUDA C

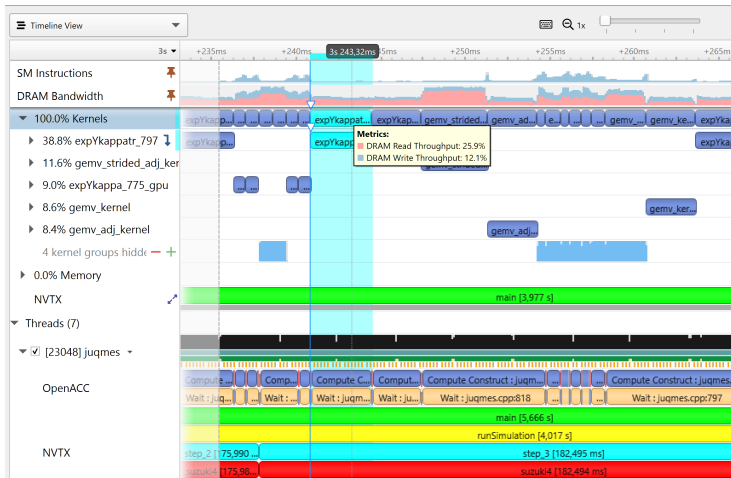
- Frequent JUWELS user



JUQMES

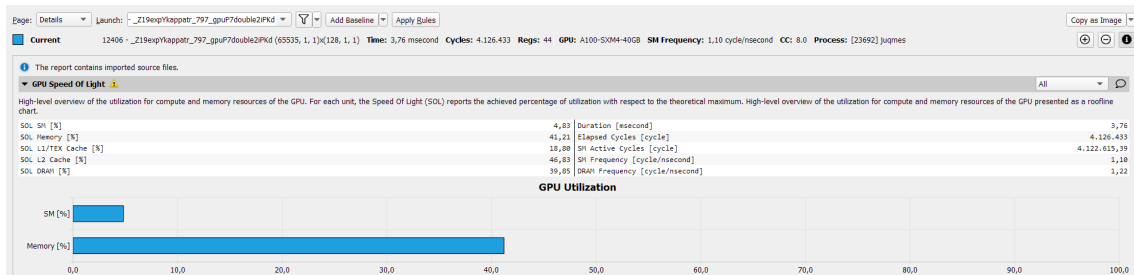
Nsight Systems

- Start of analysis with **Nsight Systems**
 - Instrumented with **NVTX** ranges (main, runSimulation, ...)
 - Focus: Kernel expYkappatr
 - Low Streaming Multiprocessor (SM) utilization
 - Low memory utilization
- What is limiter?

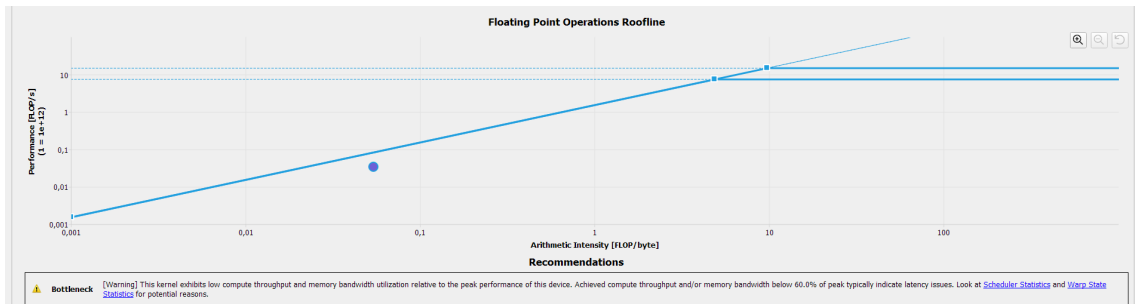


- Focus analysis with **Nsight Compute**
- Important metric: **GPU Utilization** as *Speed-of-Light*

- Low SM utilization
 - Low memory utilization
- Latency bound? (stalls? parallelism)



- **Roofline Model:** Performance (FLOP/s) vs. Arithmetic Intensity (FLOP/B)
- 60 % below roofline!
→ Latency bound!
- Further tip available



Bottleneck: This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60 % of peak typically indicate latency issues. Look at Scheduler Statistics and Warp State Statistics for potential reasons.

- See statistics for scheduler and warp state
 - Many non-eligible warps, many cycles per instructions
 - Many *LG Throttles* and *Long Scoreboard Stalls* (*not shown*)
- See source view!

► Scheduler Statistics ⓘ

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	8,40	No Eligible [%]	95,14
Eligible Warps Per Scheduler [warp]	0,06	One or More Eligible [%]	4,86
Issued Warp Per Scheduler	0,05		

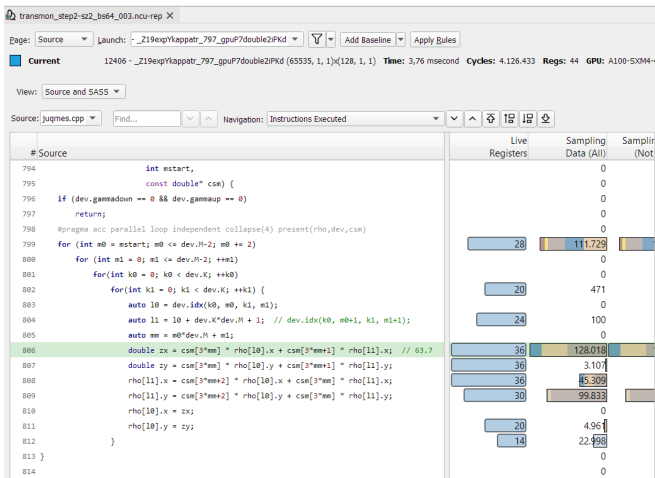
► Warp State Statistics ⓘ

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [cycle]	172,91	Avg. Active Threads Per Warp	32
Warp Cycles Per Executed Instruction [cycle]	172,95	Avg. Not Predicated Off Threads Per Warp	29,38

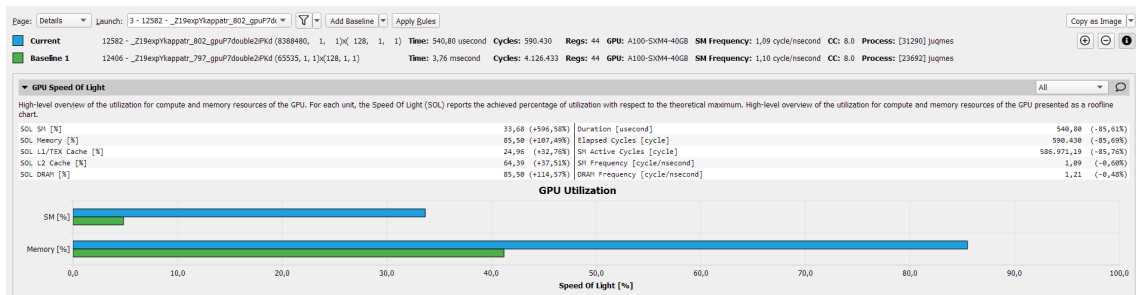
- Source view: Source vs. SASS
- With register sampling
- Pin-point stalls in source
- Non-optimal data access

⇒ Loop re-order!

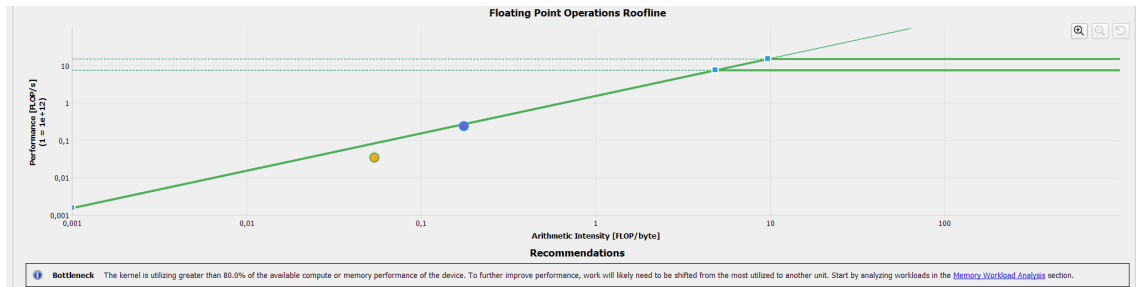


■ Improve GPU utilization

- Compare view Nsight Compute
(compare **previous baseline** with **optimized version**)



- Compare view Nsight Compute (compare **previous baseline** with **optimized version**)
- Improve GPU utilization
- Improve Roofline



- Compare view Nsight Compute (compare **previous baseline** with **optimized version**)

- Improve GPU utilization
- Improve Roofline
- Improve scheduler statistics

► Scheduler Statistics ⓘ

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	9,17 (+9,14%)	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0,61 (+904,82%)	No Eligible [%]	66,05 (-30,58%)
Issued Warp Per Scheduler	0,34 (+598,51%)	One or More Eligible [%]	33,95 (+598,51%)

► Warp State Statistics ⓘ

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [warp]	27,02	Avg. Active Threads Per Warp	32 (+0,00%)
Warp Cycles Per Issue Active [warp]	27,02	Avg. Not Predicated Off Threads Per Warp	29,44 (+0,19%)
Warp Cycles Per Executed Instruction [cycle]	27,02 (-84,38%)		

- Compare view Nsight Compute
(compare **previous baseline** with **optimized version**)

- Improve GPU utilization
 - Improve Roofline
 - Improve scheduler statistics
- ⇒ 35 % runtime improvement for benchmark, 6× for kernel

► Scheduler Statistics ⓘ

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	9,17 (+9,14%)	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0,61 (+904,82%)	No Eligible [%]	66,05 (-30,58%)
Issued Warp Per Scheduler	0,34 (+598,51%)	One or More Eligible [%]	33,95 (+598,51%)

► Warp State Statistics ⓘ

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [warp]	27,02	Avg. Active Threads Per Warp	32 (+0,00%)
Warp Cycles Per Issue Active [warp]	27,02	Avg. Not Predicated Off Threads Per Warp	29,44 (+0,19%)
Warp Cycles Per Executed Instruction [cycle]	27,02 (-84,38%)		

Early Experiences

Application-Related: *Earth-System Modeling: ParFlow*

Earth-System Modeling: ParFlow

- **ParFlow:** Numerical model for groundwater and surface water flow

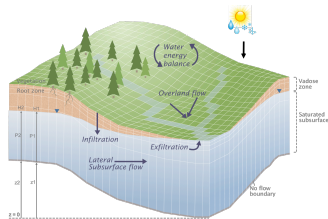
J. Hokkanen, S. Kollet, et al, EGU General Assembly 2020, 4–8 May 2020, EGU2020-12904, and GPU Seminar Talk

- Model hydrologic processes, hill-slope to continental scale; forecasting, water cycle research, climate change; since 1990s
- Finite-difference scheme with implicit time integration

👥 Team: J. Hokkanen, S. Kollet

→ parflow.org

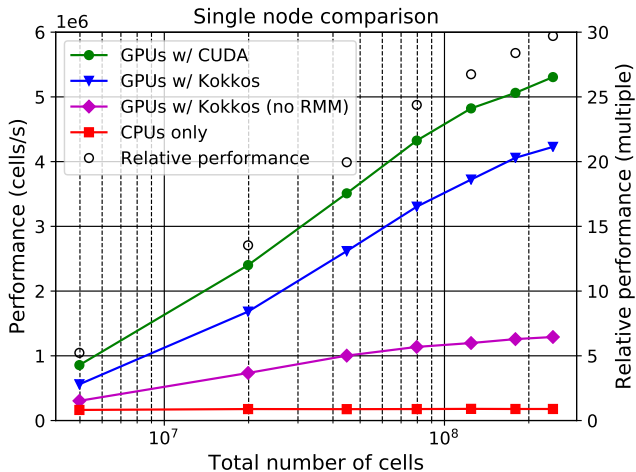
- C, C++, CUDA, MPI
- Fresh GPU port in preparation for Booster



Earth-System Modeling: ParFlow

Single-Node Performance

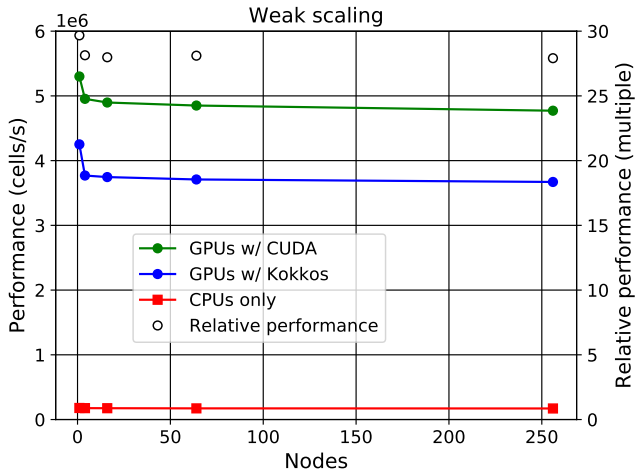
- Comparing CPU of Booster node with GPUs
- **Good speed-up, max. 29×**
- Memory pool (*RMM*) gives extra boost
- Larger problem sizes solvable per node
- Currently extending to Kokkos



Earth-System Modeling: ParFlow

Weak Scaling

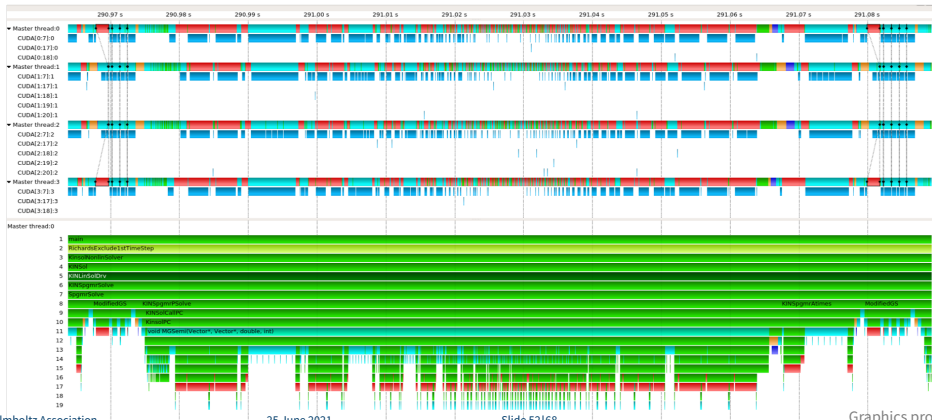
- Fixed problem size per node
- **26× speed-up achieved over $\mathcal{O}(100)$ nodes**



Earth-System Modeling: ParFlow

POP Performance Assessment

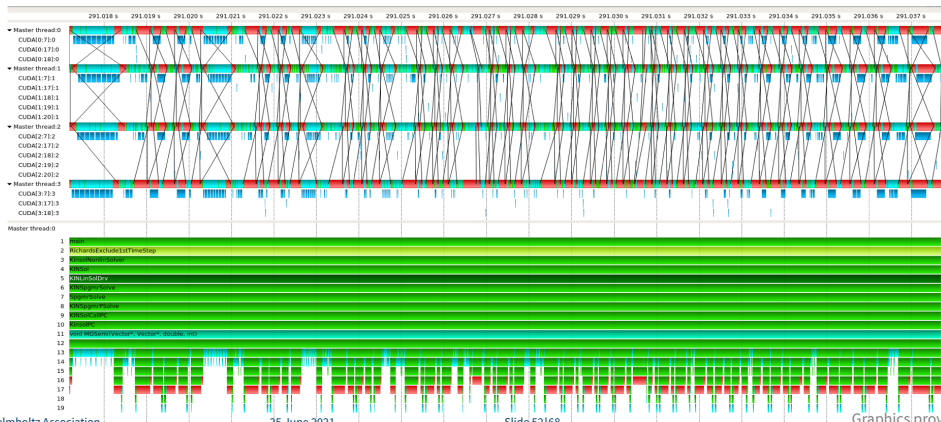
- Performance Assessment by I. Zhukov (JSC, POP CoE) at mid of 2020
- 1 node, 4 GPUs
- Instrumentation with Score-P, visualization with Vampir



Earth-System Modeling: ParFlow

POP Performance Assessment

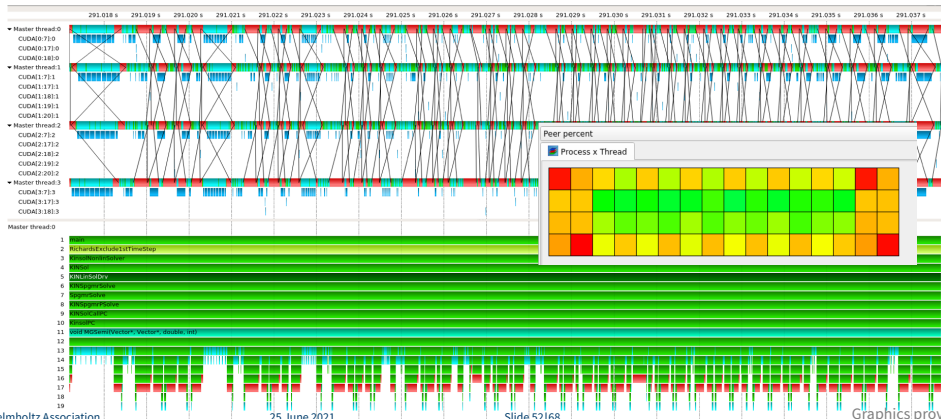
- Performance Assessment by I. Zhukov (JSC, POP CoE) at mid of 2020
- 1 node, 4 GPUs
- Instrumentation with Score-P, visualization with Vampir
- See MPI connections



Earth-System Modeling: ParFlow

POP Performance Assessment

- Performance Assessment by I. Zhukov (JSC, POP CoE) at mid of 2020
- 1 node, 4 GPUs
- Instrumentation with Score-P, visualization with Vampir
- See MPI connections, **late** senders (16 nodes)



Early Experiences

Application-Related: *JUQCS*

Quantum Computing: JUQCS

- **JUQCS:** Jülich Universal Quantum Computer Simulator

De Raedt et al., *Comp. Phys. Comm.* 237 47–61 (2019)

- Universal quantum computing on digital computer

- Network-, memory-intensive computations

- Team: Research group Quantum Information Processing

- Fortran, CUDA Fortran

- Frequent JUWELS user



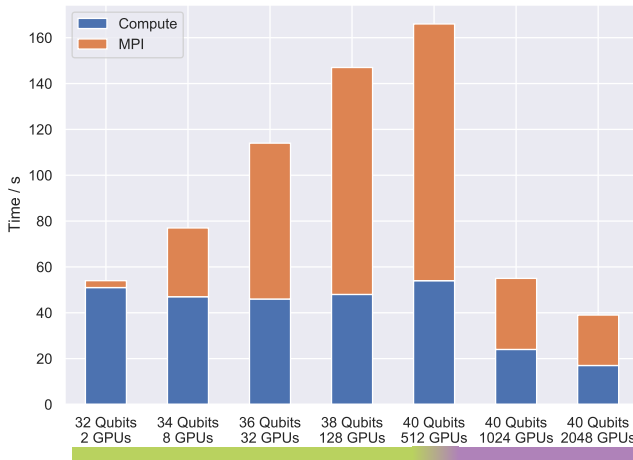
JUQCS

■ 40 qubits:

- > 16 TiB memory needed
→ 512 A100s
- Each quantum operation: Update states, 8 TB transfer

■ Weak scaling:
Compute constant, MPI
as expected

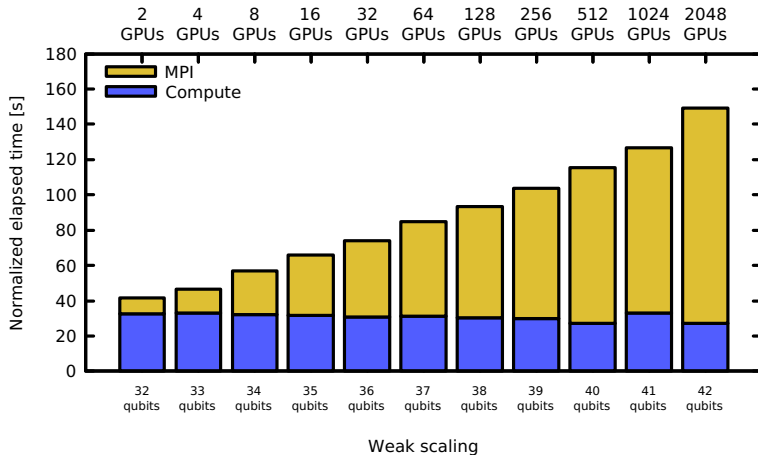
■ Strong scaling: Still
investigate DragonFly+
topology



JUQCS

More Weak Scaling

- Weak scaling to 2048 GPUs / 42 qubits
- Good behavior, but MPI still limiter

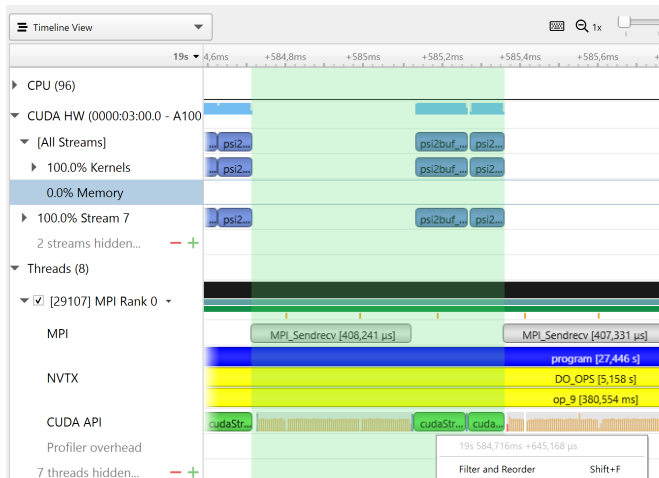


JUQCS

Performance Improvements

- Investigating performance improvements with **Nsight Systems**
- Focus on gabs between kernels

→ Used by MPI

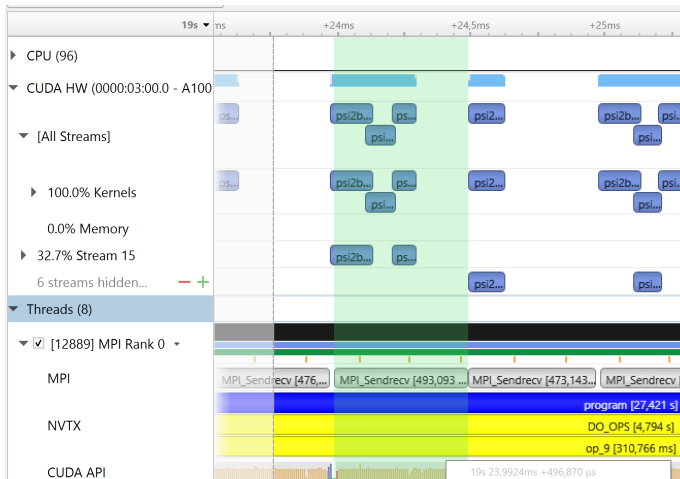


JUQCS

Performance Improvements

- Implement double-buffering
- Overlap GPU kernel and MPI call

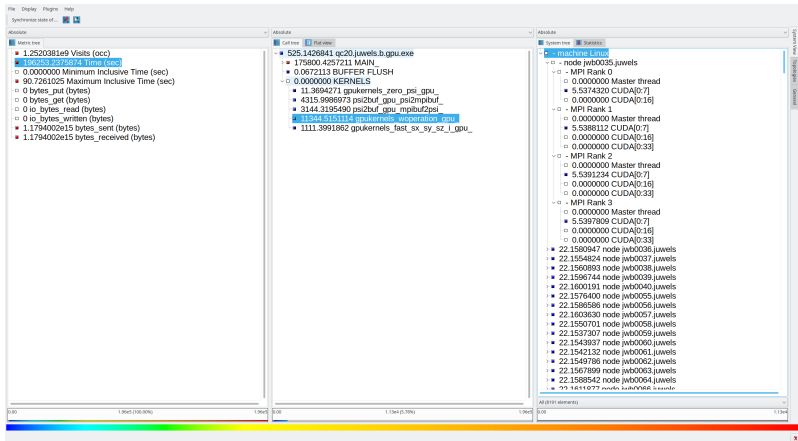
⇒ $\approx 30\%$ run-time improvement for part of iteration



JUQCS

Performance Analysis

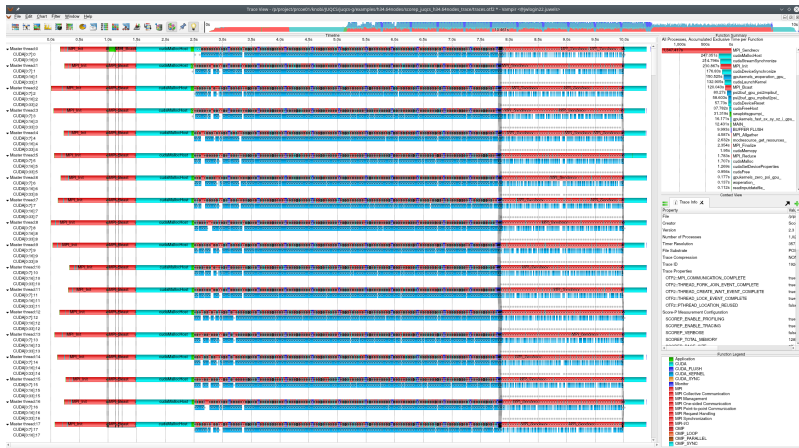
- First analysis with Score-P, Cube, Scalasca, Vampir
- See overview in Cube (2048 GPUs)



Performance Analysis

- [illegible]

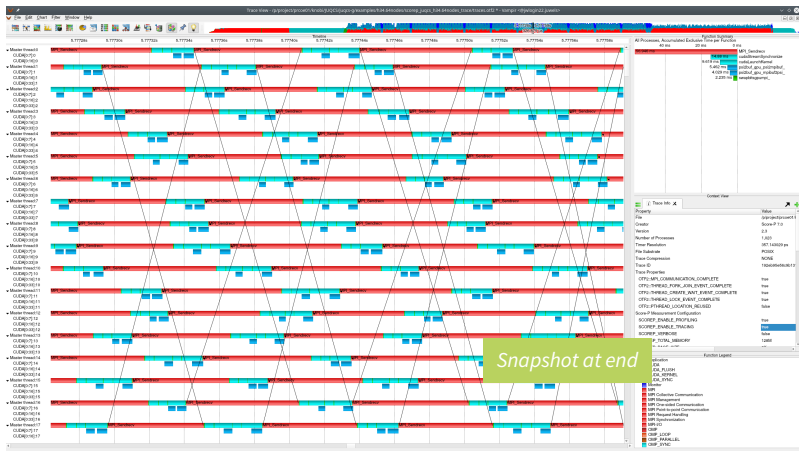
- First analysis with Score-P, Cube, Scalasca, Vampir
- See overview in Cube (2048 GPUs)
- Study late senders with in Scalasca (2048 GPUs)
- Analyze MPI flow with Vampir (256 nodes)



Performance Analysis

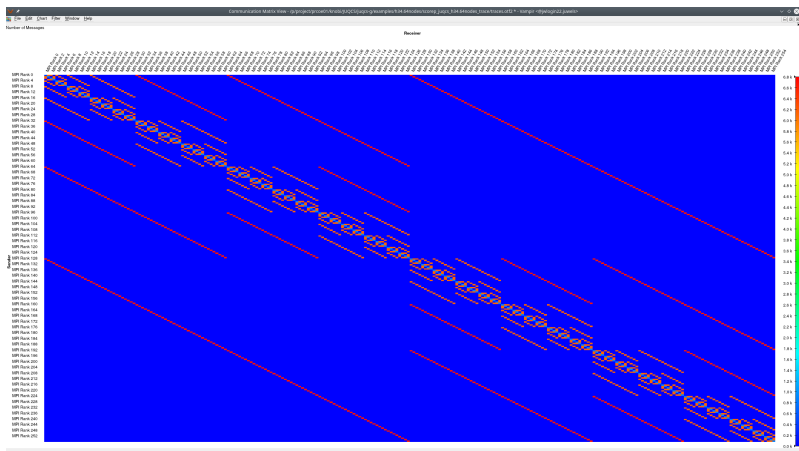
- [illegible]

- First analysis with Score-P, Cube, Scalasca, Vampir
- See overview in Cube (2048 GPUs)
- Study late senders with in Scalasca (2048 GPUs)
- Analyze MPI flow with Vampir (256 nodes) MPI communication pattern difference



Performance Analysis

- First analysis with Score-P, Cube, Scalasca, Vampir
- See overview in Cube (2048 GPUs)
- Study late senders with in Scalasca (2048 GPUs)
- Analyze MPI flow with Vampir (256 nodes) MPI communication pattern difference, MPI communication matrix



Early Experiences

Application-Related: *LQCD: Bonn*

LQCD: Bonn

- **ETMC:** Extended Twisted Mass Collaboration

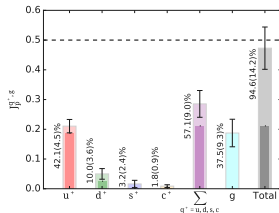
C. Alexandrou and S. Bacchio et al, Phys. Rev. D 101 094513 (2020)

- Study of the Flavour Singlet Structure of Hadrons

👥 Team: S. Bacchio, B. Kostrzewa, et al; Uni Bonn, Uni Cyprus, Cyprus Institute, Uni Rome, ...

→ github.com/etmc, PLEGMA, QUDA, tmLQCD

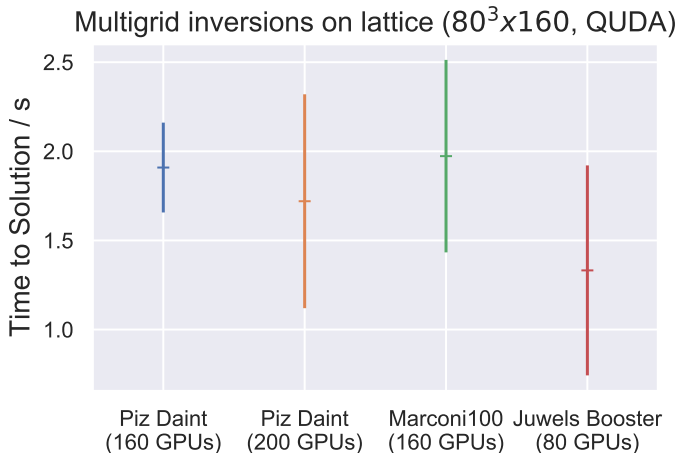
- C/C++, CUDA, MPI, OpenMP
- Frequent JUWELS user



LQCD: Bonn

Comparison of GPU HPC Machines

- Multigrid inversion
- Mean time-to-solution, spread
- Systems
 - Piz Daint Haswell, P100; DragonFly
 - Marconi100 POWER9, V100; DragonFly+
- JUWELS Booster: Low time to solution; but large spread (being investigated)



Early Experiences

Application-Related: *Others*

Distributed Training with DASO

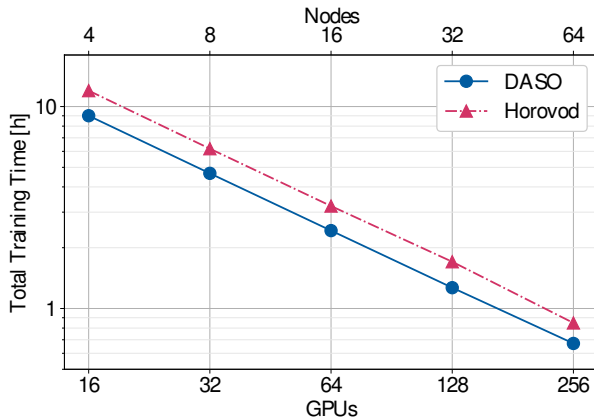
Deep-Learning

- **Publication:** Accelerating Neural Network Training with Distributed Asynchronous and Selective Optimization (DASO)



Authors: D. Coquelin et. al; KIT, DLR

- Unique: 25 % improvement over Horovod
- Status: Preprint
[arXiv:2104.05588](https://arxiv.org/abs/2104.05588) [cs.LG]



Large-Scale Ab-Initio Molecular Dynamics

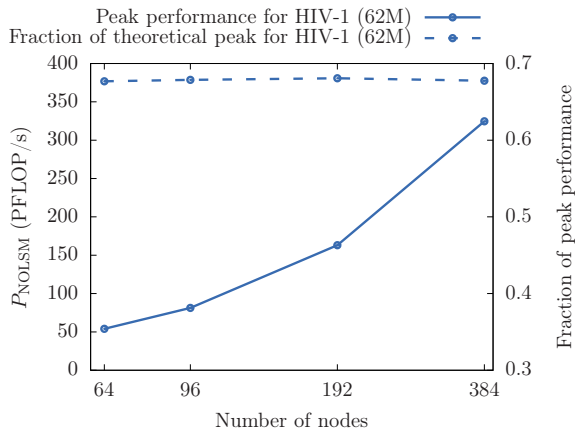
Molecular Dynamics

- **Publication:** Enabling Electronic Structure-Based Ab-Initio Molecular Dynamics Simulations with Hundreds of Millions of Atoms



Authors: R. Schade et. al;
Paderborn University

- **Unique:** FP16/FP32 mixed precision, 1536 GPUs, 324 PFLOP/s
- **Status:** Preprint
[arXiv:2104.08245](https://arxiv.org/abs/2104.08245)
[physics.comp-ph]



Summary and Conclusions

Summary and Conclusions

- JUWELS Booster: European flagship system based on A100 GPUs and HDR200 InfiniBand network
- Highly scalable system design with $> 70 \text{ PFLOP/s}_{\text{FP64}}$ compute performance and 749 Tbit/s acc. injection bandwidth
- In production since end of November, some applications earlier through Early Access Program
- First results incoming; second allocation period started
- Most important tools for performance analysis: Nsight Systems, Nsight Compute, Score-P stack; great overview at in [B. Mohr's and M. Knobloch's recent article](#)
- Important models: GPU Utilization, cache hit rate, Roofline (see especially [N. Ding's and S. Williams GPU Roofline paper](#))

Summary and Conclusions

- JUWELS Booster: European flagship system based on A100 GPUs and HDR200 InfiniBand network
- Highly scalable system design with $> 70 \text{ PFLOP/s}_{\text{FP64}}$ compute performance and 749 Tbit/s acc. injection bandwidth
- In production since end of November, some applications earlier through Early Access Program
- First results incoming; second allocation period started
- Most important tools for performance analysis: Nsight Systems, Nsight Compute, S. Williams GPU Profiler stack; great overview at in [B. Mohr's and M. Knobloch's recent article](#)
- Important models: GPU Utilization, cache hit rate, Roofline (see [S. Williams GPU Roofline paper](#))

Thank you
for your attention!
a.herten@fz-juelich.de

Acknowledgements

Acknowledgments

- JSC Application-Oriented Technology Development: Kaveh Haghighi-Mood
- JSC Application Support: Ilya Zhukov, Michael Knobloch, Thomas Breuer, Max Holicki, Filipe Guimaraes
- JSC High Performance Systems: Dorian Krause, Damian Alvarez, Benedikt von St. Vieth
- NVIDIA Collaborators: Markus Hrywniak, Jiri Kraus, Mathias Wagner
- Application owners: Jaro Hokkanen, Dennis Willsch, Hans de Raedt, Ludwig Schneider

Acknowledgements

Acknowledgments

- JSC Application-Oriented Technology Development: Kaveh Haghighi-Mood
- JSC Application Support: Ilya Zhukov, Michael Knobloch, Thomas Breuer, Max Holicki, Filipe Guimaraes
- JSC High Performance Systems: Dorian Krause, Damian Alvarez, Benedikt von St. Vieth
- NVIDIA Collaborators: Markus Hrywniak, Jiri Kraus, Mathias Wagner
- Application owners: Jaro Hokkanen, Dennis Willsch, Hans de Raedt, Ludwig Schneider

**Thank you
for your attention!**
a.herten@fz-juelich.de



Appendix

Appendix

- Tools Wishlist

- Network Performance

- STREAM Benchmark

Supplement: Application Experiences

- PICongPU

- Others

- Pre-Training, Transfer Learning

- References

Tools Wishlist

OSU Micro-Benchmarks: Bandwidth

- Guideline for Tensor Core potential
- Consider node, network affinity / topology

Appendix

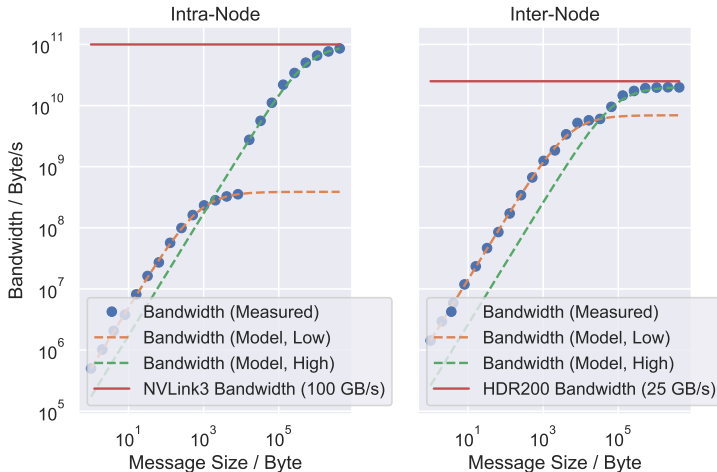
Network Performance

Network Performance

OSU Micro-Benchmarks: Bandwidth

- OSU Microbenchmarks: device-device bandwidth (osu_bw D D)
- Good results, expected limiters
- Intra-node: NVLink3 bandwidth
- Inter-node: HDR200 bandwidth
- Model fits show 2 regimes (--- / - - -)

JUWELS Booster Device-Device Bandwidth (osu_bw)



Appendix

STREAM Benchmark

STREAM

Config

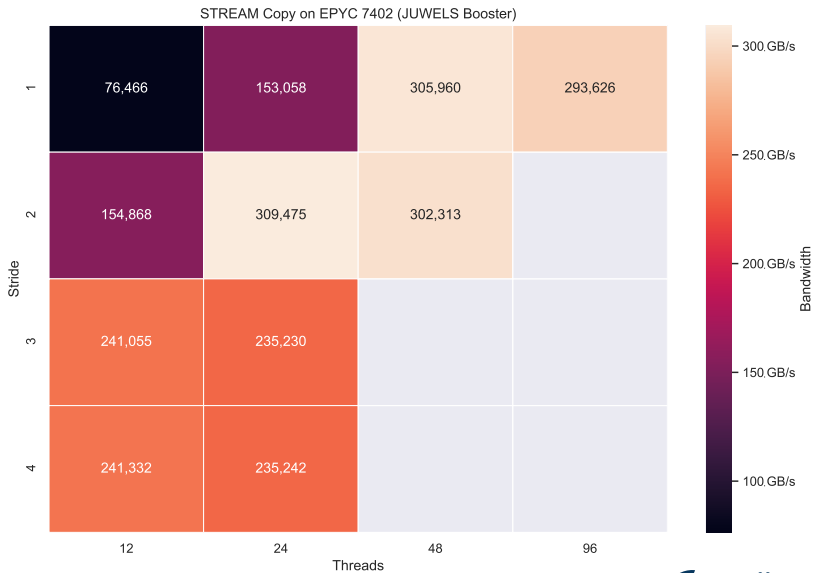
AOCC 3.0.0 on EPYC 7402

```
$ clang -O3 -mmodel=large -fopenmp -ffp-contract=fast -fnt-store -DSTREAM_ARRAY_SIZE=2500000000  
↪ -DNTIMES=200 -DSTREAM_TYPE=double -o stream.exe stream.c  
$ srun --cpu-bind=none \  
    env \  
        OMP_NUM_THREADS=24 \  
        OMP_PLACES='{0}:24:2' \  
        OMP_PLACES="cores" \  
        OMP_PROC_BIND="true" \  
        OMP_MAX_ACTIVE_LEVELS=1 \  
        OMP_STACKSIZE=256M \  
        OMP_SCHEDULE=static \  
        OMP_DYNAMIC=false \  
        OMP_THREAD_LIMIT=256 \  
        OMP_DISPLAY_ENV=true \  
    ./stream.exe
```



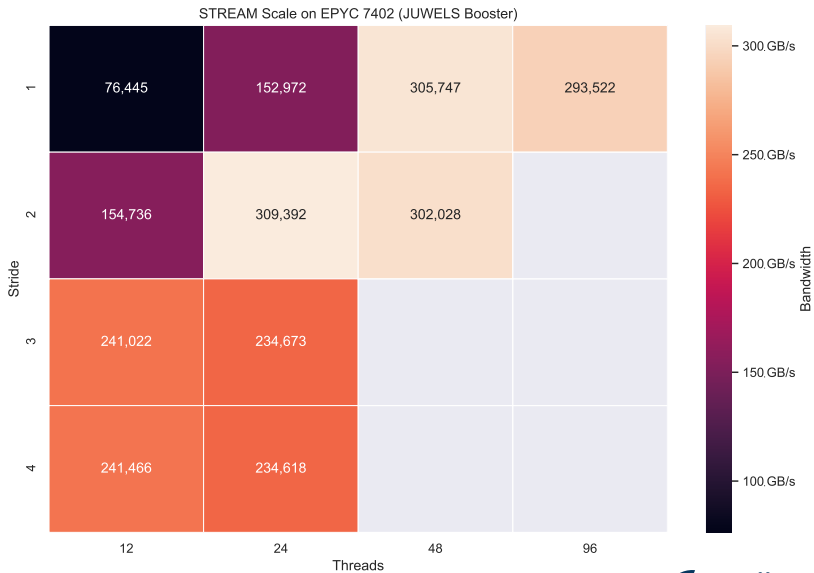
STREAM

Copy



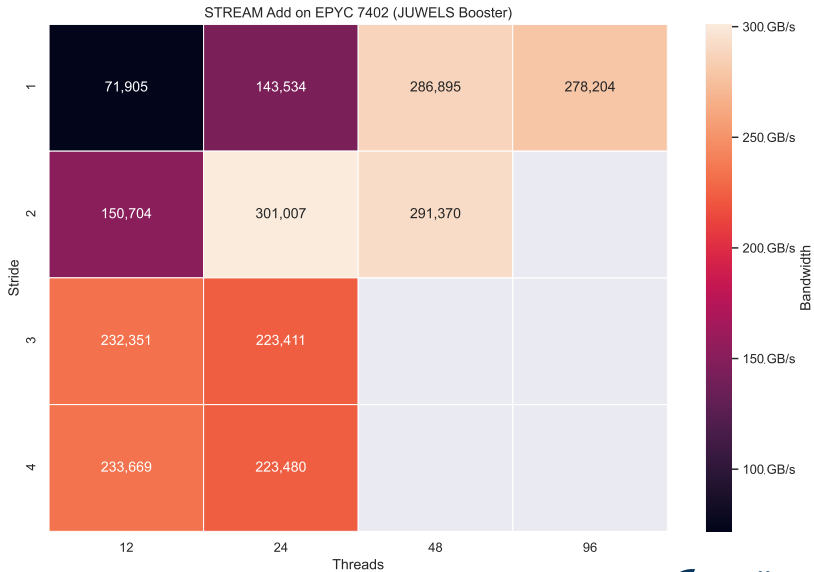
STREAM

Scale



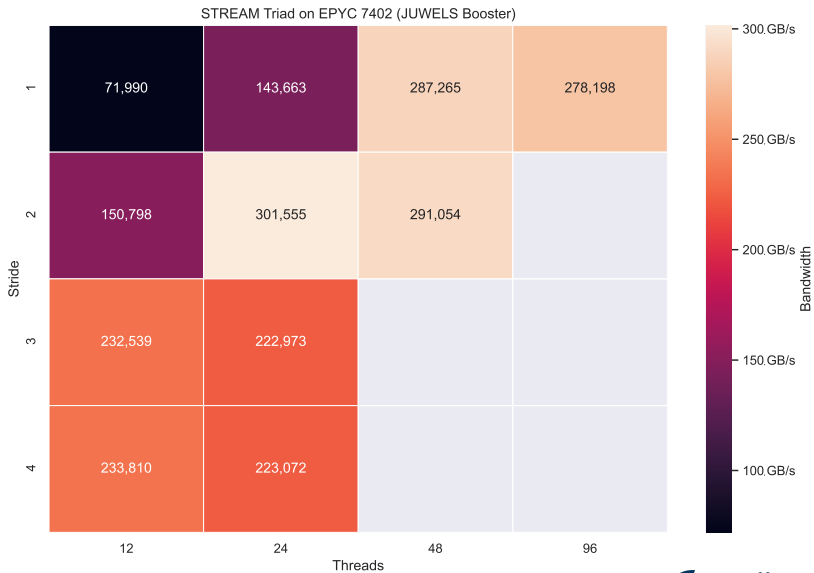
STREAM

Add



STREAM

Triad



Supplement: Application Experiences

- **PICongPU**: Plasma simulation

H. Burau et al, IEEE Transactions on Plasma Science 38 10 (2010)

- Particle-in-cell simulation for Exascale-level GPUs

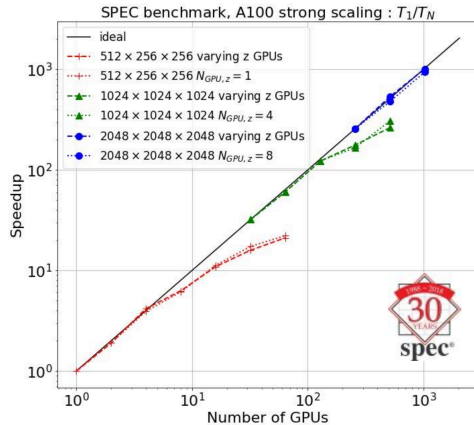
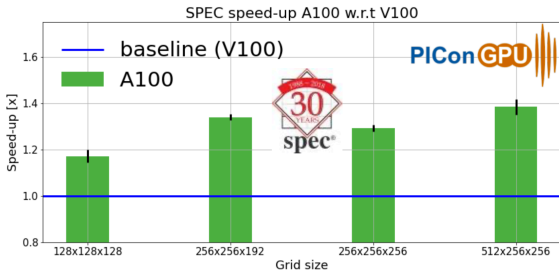
 Team: A. Lebedev, A. Debus, M. Bussmann, et. al

→ github.com/ComputationalRadiationPhysics/picongpu

- C/C++, CUDA, MPI, Alpaka

PIConGPU

Results



Strong scaling for different grid sizes

Large-Scale Pre-Training on Transfer Learning for Images

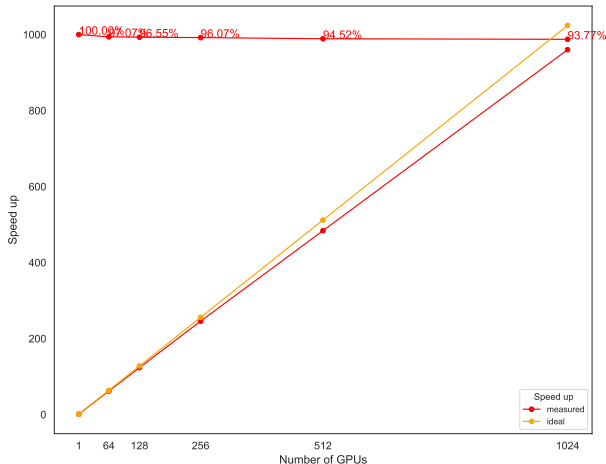
Deep-Learning

- **Publication:** Effect of large-scale pre-training on full and few-shot transfer learning for natural and medical images



Authors: Mehdi Cherti, Jenia Jitsev; JSC

- **Status:** Preprint (under review)
[arXiv:2106.00116 \[cs.LG\]](https://arxiv.org/abs/2106.00116)



Supplement: Application Experiences

References

References I

- [5] Hans De Raedt et al. “Massively parallel quantum computer simulator, eleven years later.” In: *Computer Physics Communications* 237 (2019), pp. 47–61. ISSN: 0010-4655 (page 104).
- [6] Michael Knobloch and Bernd Mohr. “Tools for GPU Computing – Debugging and Performance Analysis of Heterogenous HPC Applications.” In: *Supercomputing Frontiers and Innovations* 7.1 (2020). ISSN: 2313-8734 (pages 122, 123).
- [7] Nan Ding and Samuel Williams. “An Instruction Roofline Model for GPUs.” In: *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2019, pp. 7–18 (pages 122, 123).

References: Images, Graphics I

- [1] Forschungszentrum Jülich. *Forschungszentrum Bird's Eye*. (Pages 3, 4).
- [2] Rob984 via Wikimedia Commons. *Europe orthographic Caucasus Urals boundary (with borders)*. (Pages 3, 4).
- [3] Forschungszentrum Jülich GmbH (Ralf-Uwe Limbach). *JUWELS Cluster*.
- [4] Forschungszentrum Jülich GmbH (Ralf-Uwe Limbach). *JUWELS Booster*.