



INTRODUCTION TO HPC MAELSTROM BOOTCAMP JÜLICH

27 September 2022 | Dr. Andreas Herten | Forschungszentrum Jülich, MAELSTROM

Outline

text3

Introduction

Hardware

- Comparison to PC

- HPC vs. PC

- HPC

HPC System Overview

- Historical Machines

- JUWELS

 - JUWELS Cluster

 - JUWELS Booster

- GPUs

text3

Software

- 1: Core Utilization

- 2: Parallelization

- 3: Distribution

- Enablement

Conclusion



*What is **HPC**?*

***High Performance Computing is
computing with a powerful machine
using the available resources efficiently.***

My interpretation.

Baseline

- What kind of **CPU** does your computer have?

CPU generation, clock speed rate, number of cores, vector length



Baseline

- What kind of **CPU** does your computer have?

CPU generation, clock speed rate, number of cores, vector length

- How much **memory** does your computer have?

Amount of memory, type, links (GB and GB/s)



Baseline

- What kind of **CPU** does your computer have?

CPU generation, clock speed rate, number of cores, vector length

- How much **memory** does your computer have?

Amount of memory, type, links (GB and GB/s)

- What kind of **GPU** do you have?

GPU generation, number of cores, watt intake (TDP)



Baseline

- What kind of **CPU** does your computer have?

CPU generation, clock speed rate, number of cores, vector length

- How much **memory** does your computer have?

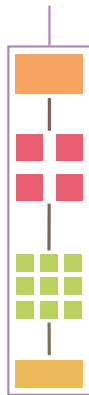
Amount of memory, type, links (GB and GB/s)

- What kind of **GPU** do you have?

GPU generation, number of cores, watt intake (TDP)

- How fast is your **network** ?

Throughput, latency



Baseline

- What kind of **CPU** does your computer have?

CPU generation, clock speed rate, number of cores, vector length

- How much **memory** does your computer have?

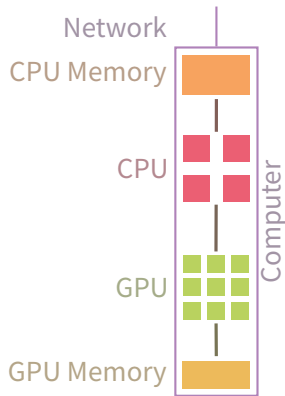
Amount of memory, type, links (GB and GB/s)

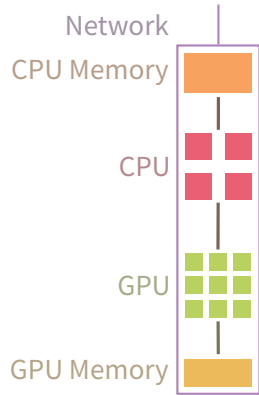
- What kind of **GPU** do you have?

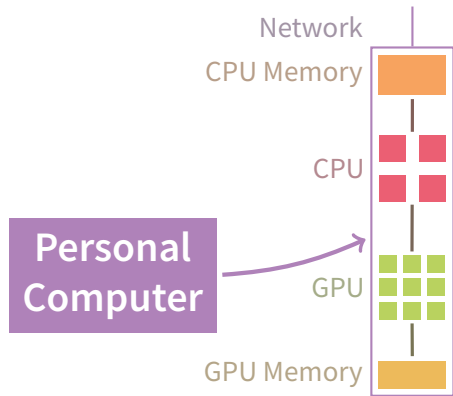
GPU generation, number of cores, watt intake (TDP)

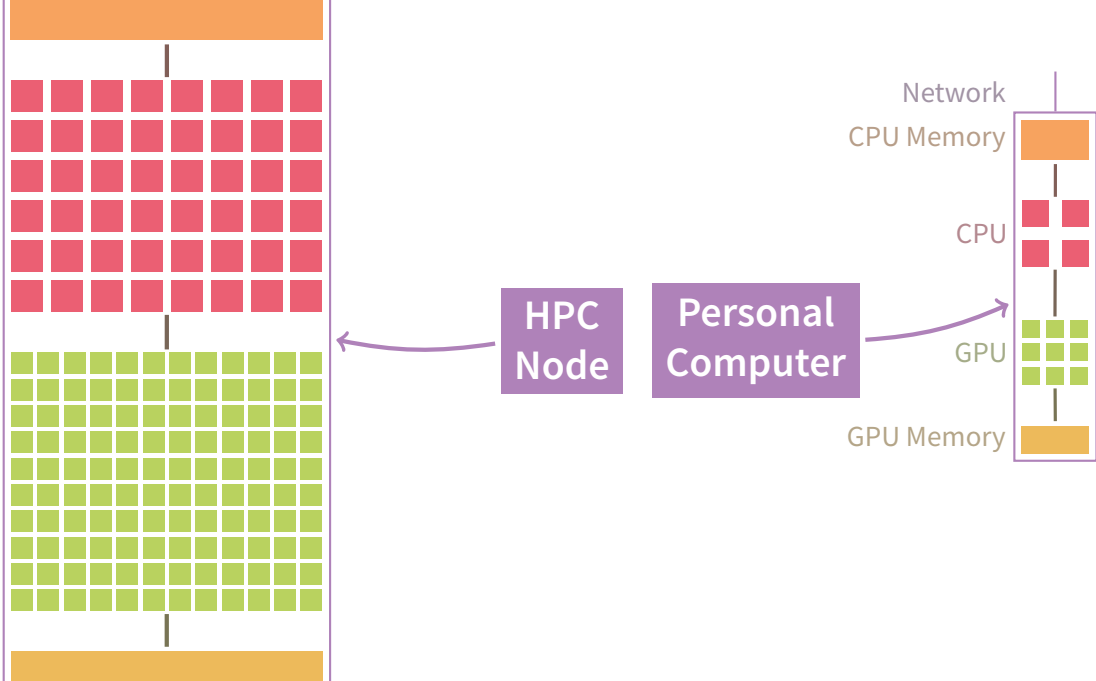
- How fast is your **network** ?

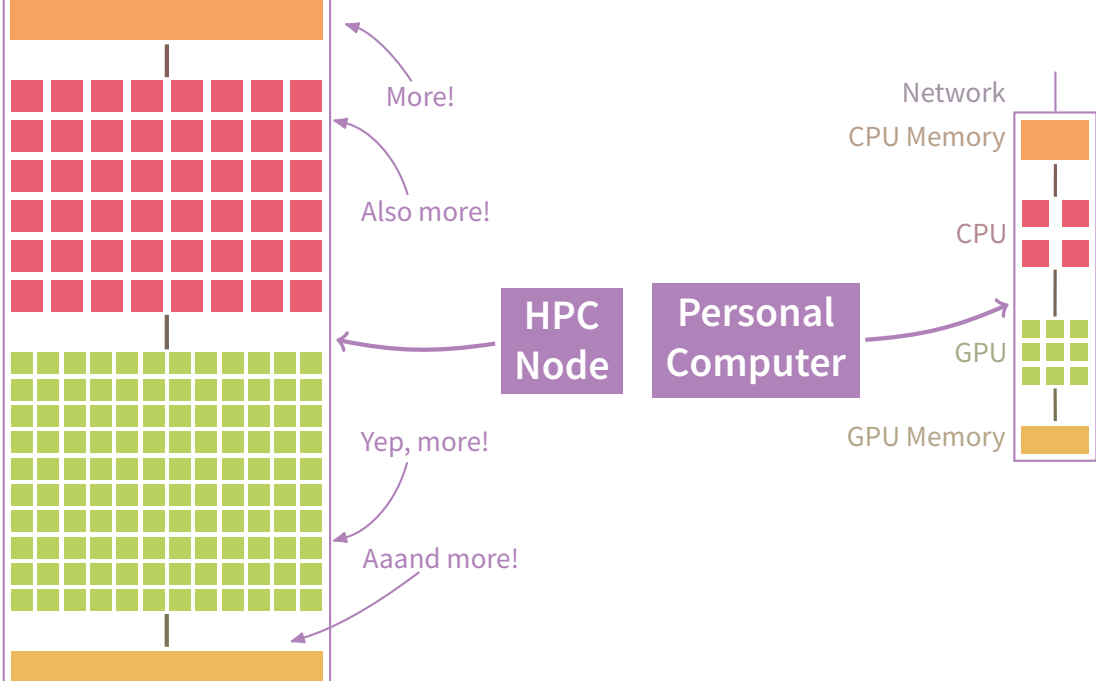
Throughput, latency

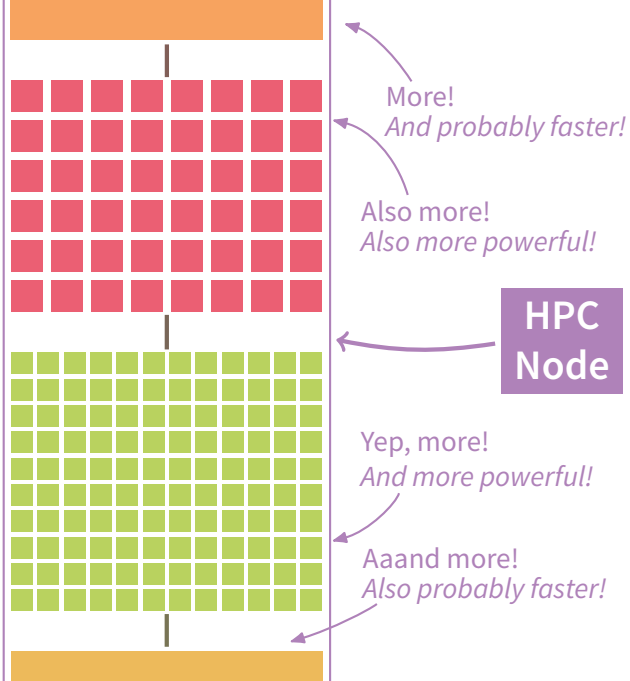




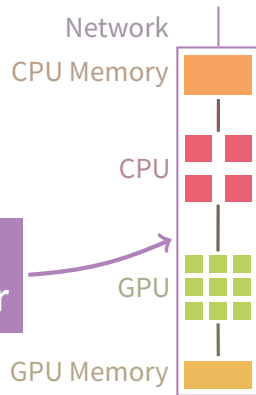


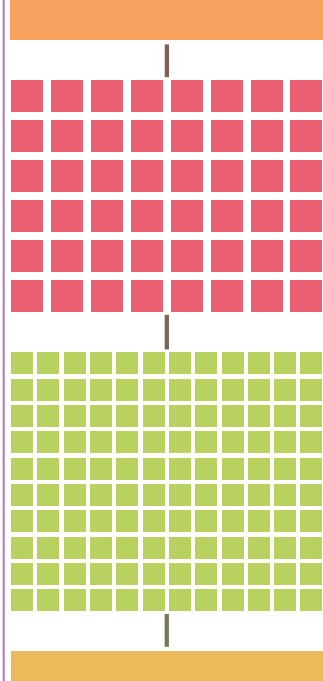






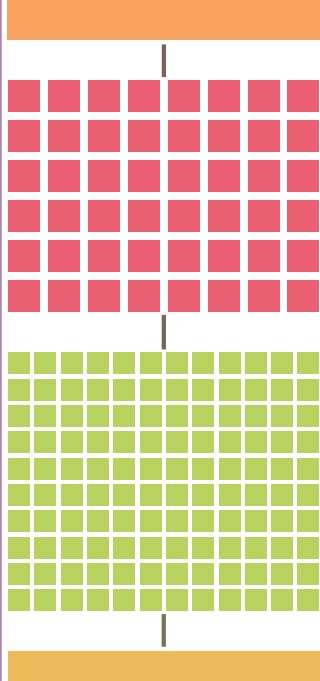
Personal Computer





HPC
Node



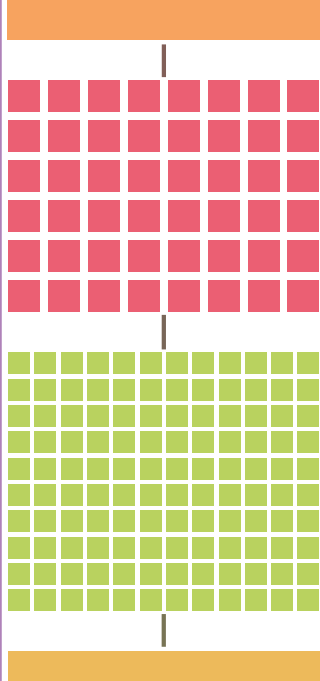


CPU

64 cores, 2.2 GHz, 2× multi-threading, large caches, advanced instructions, 220 W TDP

HPC
Node

Specs based on JURECA DC



Memory

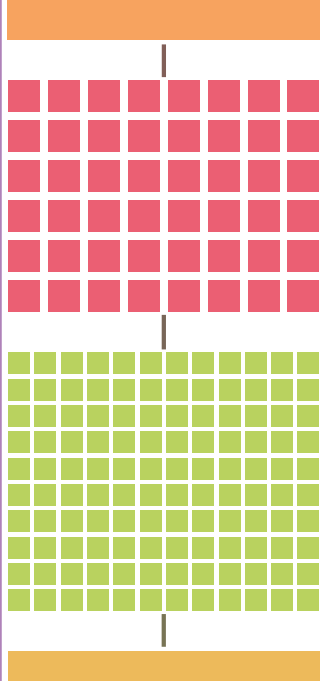
1 TB size, DDR4, 3200 MHz rate, 190 GiB/s bandwidth

CPU

64 cores, 2.2 GHz, 2× multi-threading, large caches, advanced instructions, 220 W TDP

HPC
Node

Specs based on JURECA DC



Memory

1 TB size, DDR4, 3200 MHz rate, 190 GiB/s bandwidth

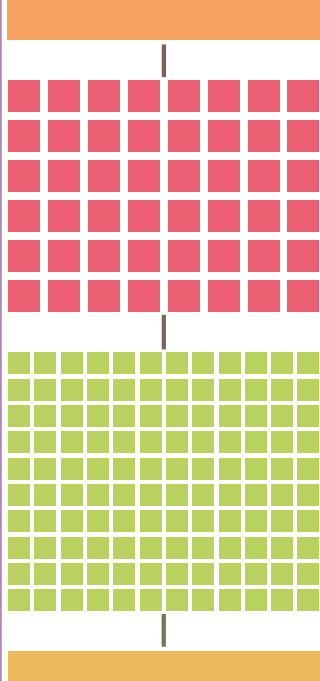
CPU

64 cores, 2.2 GHz, 2× multi-threading, large caches, advanced instructions, 220 W TDP

HPC Node

GPU

108 cores, 1400 MHz, 2048 bit vector size, double-precision support, 400 W TDP, no graphics



Memory

1 TB size, DDR4, 3200 MHz rate, 190 GiB/s bandwidth

CPU

64 cores, 2.2 GHz, 2× multi-threading, large caches, advanced instructions, 220 W TDP

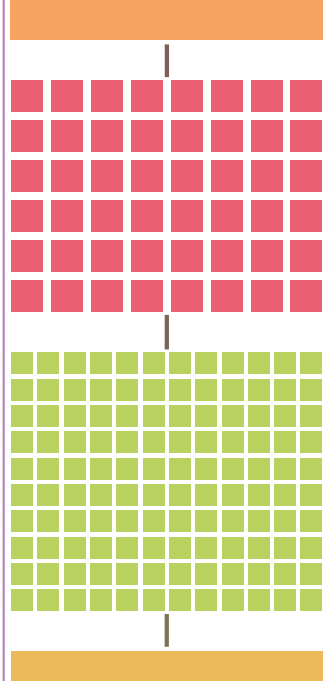
HPC Node

GPU

108 cores, 1400 MHz, 2048 bit vector size, double-precision support, 400 W TDP, no graphics

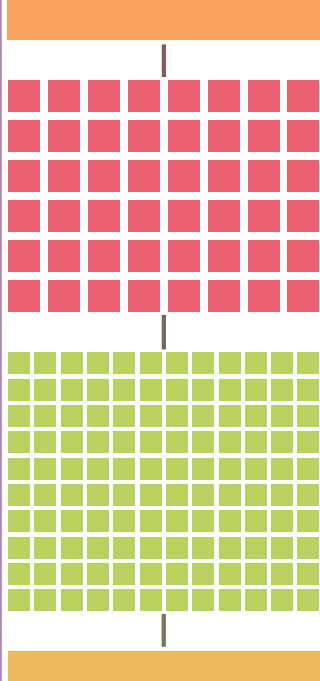
GPU VRAM

40 GB (80 GB also available), HBM2, 1.555 GB/s bandwidth



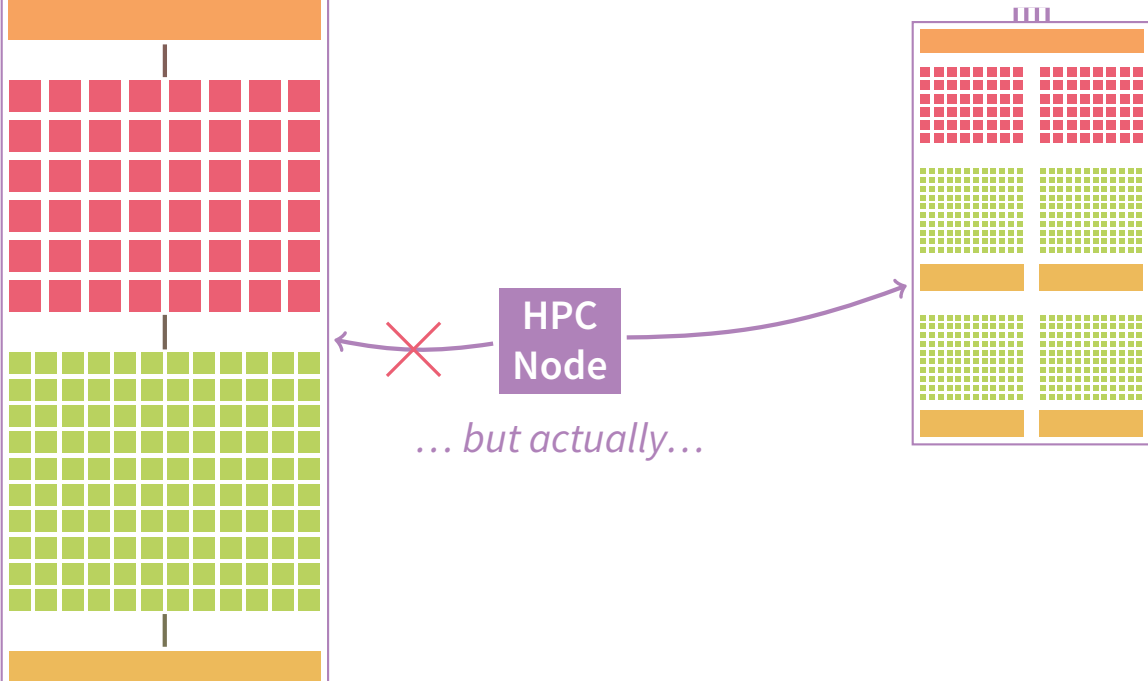
HPC
Node





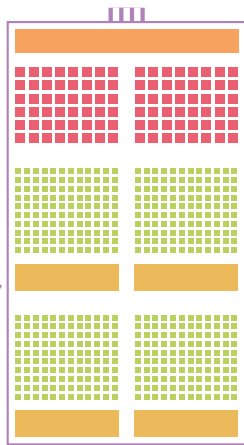
HPC
Node

... but actually...

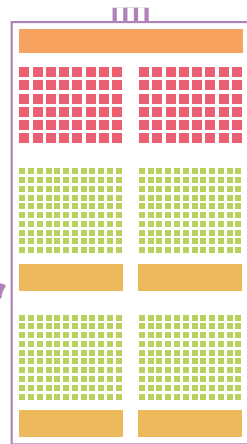


- Usually, 2 **CPU**s sockets, each with 64 cores; use mostly as one CPU with one memory
- 4 distinct **GPU**s, connected with each other (600 GB/s)
- 4 **network** connections, each 200 Gbit/s in each direction (*InfiniBand HDR-200*)

HPC
Node

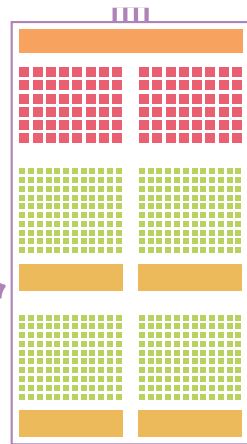


HPC
Node

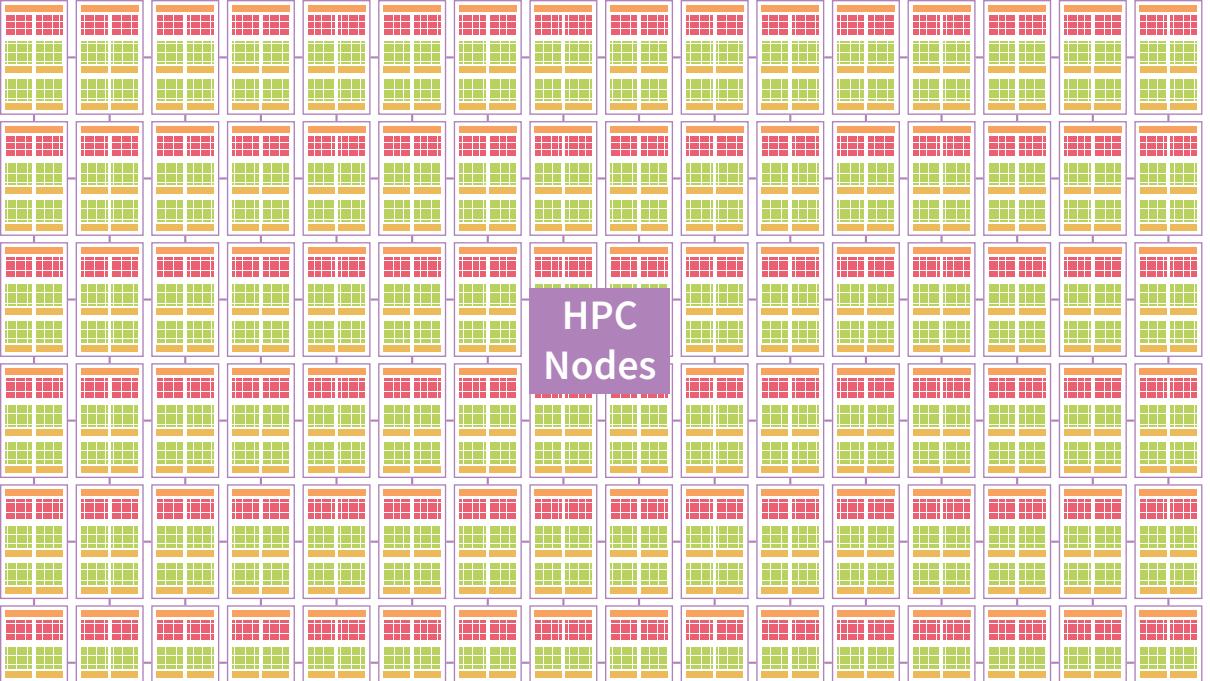


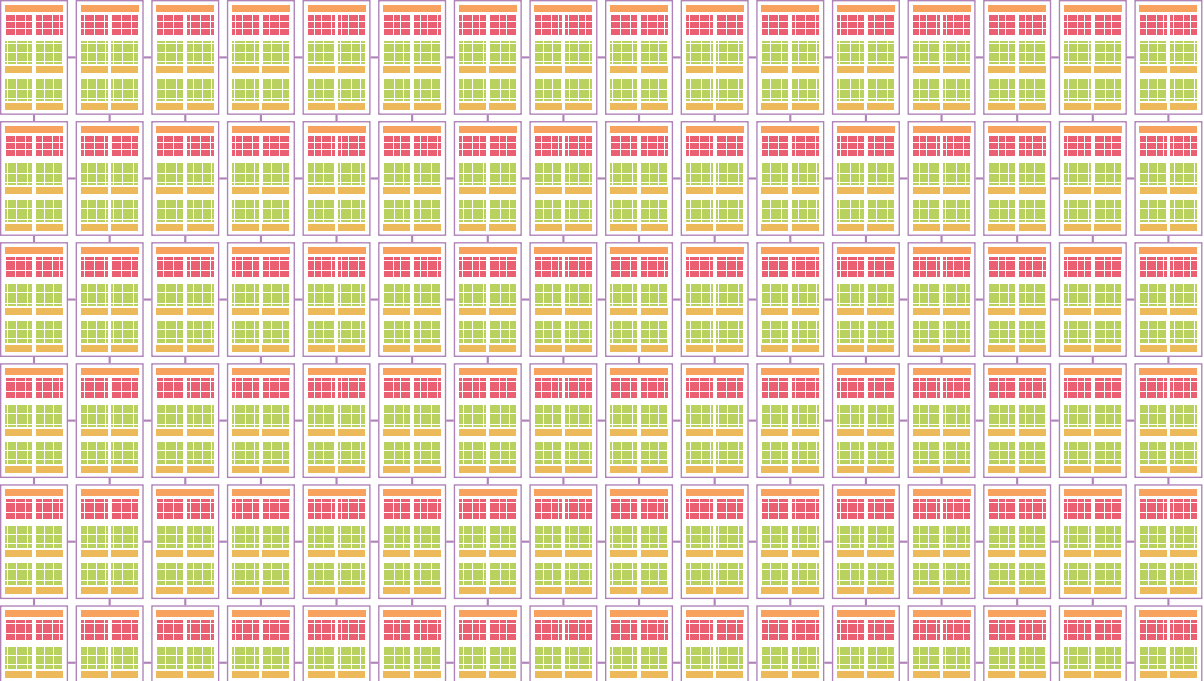
Specs based on JURECA DC, JUWELS

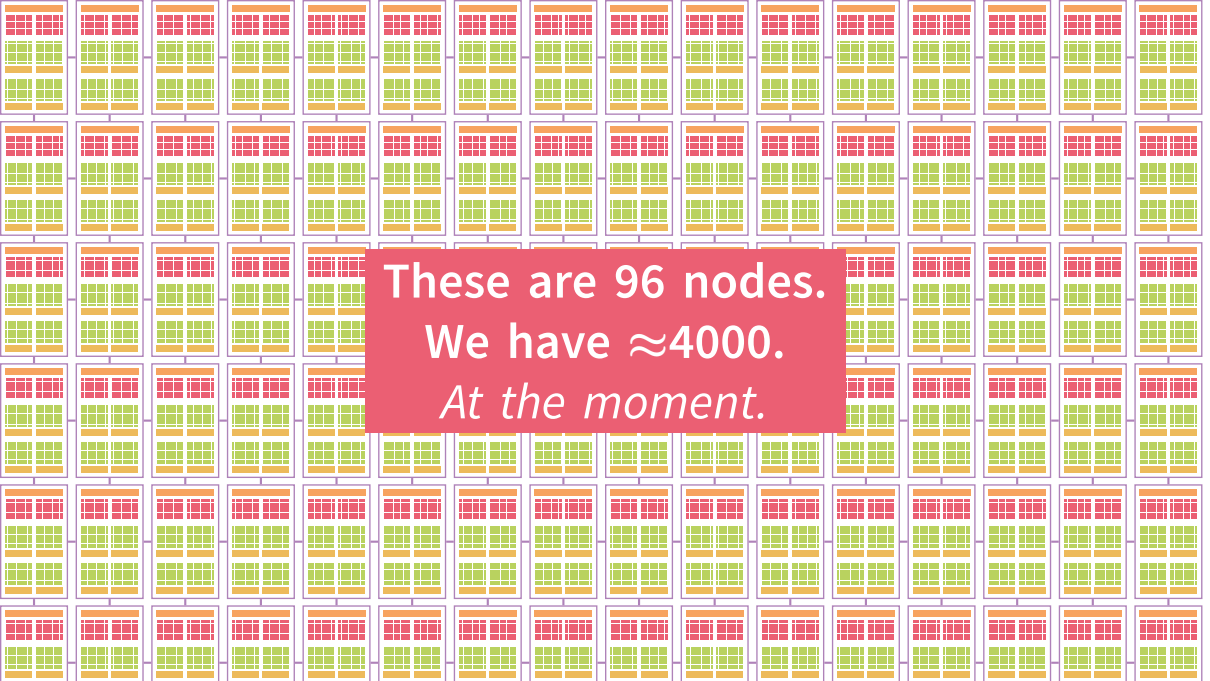
HPC
Nodes



Specs based on JURECA DC, JUWELS







These are 96 nodes.
We have ≈ 4000 .
At the moment.

***High Performance Computing is
computing with a powerful machine
using the available resources efficiently.***

My interpretation.

**High Performance Computing is
computing with a powerful machine
using the available resources efficiently.**

Powerful Machines

- Now
- Powerful nodes (large CPUs, accelerating GPUs, much memory)
 - Many nodes (well-connected through high-speed interconnect)
- **Beefed-up versions of commodity computers, with slight specializations; many**

Powerful Machines

- Now**
- Powerful nodes (large CPUs, accelerating GPUs, much memory)
 - Many nodes (well-connected through high-speed interconnect)
- **Beefed-up versions of commodity computers, with slight specializations; many**
- Past**
- First computers: Supercomputers! Mainframe machines: Large installations with most powerful hardware at the time
 - PC era: Even then, specialized computers, like vector machines, or many low-speed CPUs (well-connected)
 - Recent history: x86, then PowerPC, then GPU accelerators, then specialized Arm CPUs

Supercomputers in Pictures



- CDC 6600 supercomputer
- Around 1965
- First supercomputer
- 3 MFLOP/s
- See [Wikipedia](#) for more
- Picture by Control Data Corporation

Supercomputers in Pictures



- CDC 6600 supercomputer
- Around 1965
- First supercomputer
- 3 MFLOP/s
- See [Wikipedia](#) for more
- Picture by Control Data Corporation

Supercomputers in Pictures



HPC performance measured in **FLOP/s**.

- Floating-point (like 3.14) operations per second

Supercomputers in Pictures



HPC performance measured in **FLOP/s**.

- Floating-point (like 3.14) operations per second
- Example: Processor with 2 GHz; 10 cores; per core: 2 multiplications and 2 additions (*FMA*) per cycle



Supercomputers in Pictures



HPC performance measured in **FLOP/s**.

- Floating-point (like 3.14) operations per second
- Example: Processor with 2 GHz; 10 cores; per core: 2 multiplications and 2 additions (*FMA*) per cycle

$$2 \times 10^9 \text{ 1/s 1/core} * 10 \text{ core} * \\ * (2 + 2) \text{ floating-point operation}$$



Supercomputers in Pictures



HPC performance measured in **FLOP/s**.

- Floating-point (like 3.14) operations per second
- Example: Processor with 2 GHz; 10 cores; per core: 2 multiplications and 2 additions (*FMA*) per cycle

$$\begin{aligned} & 2 \times 10^9 \text{ 1/s 1/core} * 10 \text{ core} * \\ & * (2 + 2) \text{ floating-point operation} \\ & = 2 * 10^9 * 10 * 4 \text{ fl-op/s} \end{aligned}$$



Supercomputers in Pictures



HPC performance measured in **FLOP/s**.

- Floating-point (like 3.14) operations per second
- Example: Processor with 2 GHz; 10 cores; per core: 2 multiplications and 2 additions (*FMA*) per cycle

$$\begin{aligned} & 2 \times 10^9 \text{ 1/s 1/core} * 10 \text{ core} * \\ & \quad * (2 + 2) \text{ floating-point operation} \\ & = 2 * 10^9 * 10 * 4 \text{ fl-op/s} \\ & = 80 * 10^9 \text{ FLOP/s} \\ & = 80 \text{ GFLOP/s} \end{aligned}$$



Supercomputers in Pictures



- Cray-1 supercomputer
- Around 1978
- Very successful
- 160 MFLOP/s
- Probably pictured at NERSC

Supercomputers in Pictures



- Intel XP/S 140 supercomputer
- Around 1994
- 3680 Intel i860 RISC processors; large-scale parallel system
- 143 GFLOP/s
- Picture by top500.org

Supercomputers in Pictures



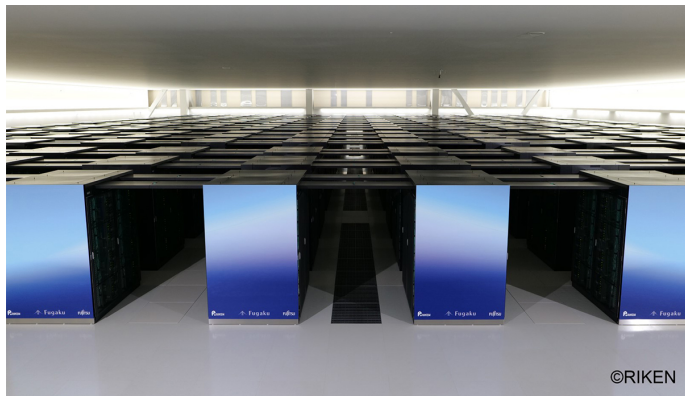
- JUGENE supercomputer
- 2008
- 294 912 PowerPC 450 cores; energy-efficient
- 800 TFLOP/s
- Picture by top500.org

Supercomputers in Pictures



- Summit supercomputer
- 2018
- 27 000 GPUs hosted by POWER9 CPUs; first #1 GPU supercomputer
- 200 PFLOP/s
- Picture by Oak Ridge National Lab

Supercomputers in Pictures



- Fugaku supercomputer
- 2020
- 7 630 848 Arm A64FX cores; #1 supercomputer
- 537 PFLOP/s
- Picture by RIKEN



JUWELS Cluster – Jülich's Scalable System

- 2500 nodes with Intel Xeon CPUs (2×24 cores)
- 46 + 10 nodes with 4 NVIDIA Tesla V100 cards (16 GB memory)
- 10.4 (CPU) + 1.6 (GPU) PFLOP/s peak performance (Top500: #86)



JUWELS Booster – Scaling Higher!

- 936 nodes with AMD EPYC Rome CPUs (2×24 cores)
- Each with 4 NVIDIA A100 Ampere GPUs (each: $\text{FP64TC: } 19.5$ TFLOP/s, 40 GB memory)
 $\text{FP64: } 9.7$
- InfiniBand DragonFly+ HDR-200 network; 4×200 Gbit/s per node



Top500 List Nov 2021:

- #1 Europe
- #8 World
- #4* Top/Green500



JUWELS Booster – Scaling Higher!

- 936 nodes with AMD EPYC Rome CPUs (2×24 cores)
- Each with 4 NVIDIA A100 Ampere GPUs (each: $\text{FP64TC: } 19.5$ TFLOP/s, 40 GB memory)
 $\text{FP64: } 9.7$
- InfiniBand DragonFly+ HDR-200 network; 4×200 Gbit/s per node

State of the Art

Picture by OLCF at ORNL on Flickr

- Current fastest supercomputer: **Frontier** at Oak Ridge (USA) with 38 000 AMD MI250X GPUs – 1.102 EFLOP/s; also most energy-efficient!



State of the Art

Picture by OLCF at ORNL on Flickr

- Current fastest supercomputer: **Frontier** at Oak Ridge (USA) with 38 000 AMD MI250X GPUs – 1.102 EFLOP/s; also most energy-efficient!
- 2022: Aurora at Argonne with > 60 000 Intel Ponte Vecchio GPUs – > 2 EFLOP/s
- 2023: El Capitan at Lawrence Livermore with AMD MI300 GPUs – > 2 EFLOP/s



State of the Art

Picture by OLCF at ORNL on Flickr

- Current fastest supercomputer: **Frontier** at Oak Ridge (USA) with 38 000 AMD MI250X GPUs – 1.102 EFLOP/s; also most energy-efficient!
- 2022: Aurora at Argonne with > 60 000 Intel Ponte Vecchio GPUs – > 2 EFLOP/s
- 2023: El Capitan at Lawrence Livermore with AMD MI300 GPUs – > 2 EFLOP/s
- 2023: JUPITER at JSC – > 1 EFLOP/s! GPUs, but details TBD



State of the Art

Picture by OLCF at ORNL on Flickr

- Current fastest supercomputer: **Frontier** at Oak Ridge (USA) with 38 000 AMD MI250X **GPUs** – 1.102 EFLOP/s; also most energy-efficient!
- 2022: Aurora at Argonne with > 60 000 Intel Ponte Vecchio **GPUs** – > 2 EFLOP/s
- 2023: El Capitan at Lawrence Livermore with AMD MI300 **GPUs** – > 2 EFLOP/s
- 2023: JUPITER at JSC – > 1 EFLOP/s! **GPUs**, but details TBD

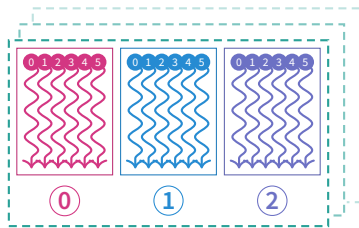


GPUs

- **GPUs:** Exascale Enablers
- Processors efficient at applying same (/similar) instruction on large set of data (image)
- Over last 15 years, extended from rendering to variable computing
- Not good for every task, but **great for some**, which happen to be computing with large amounts of similar data

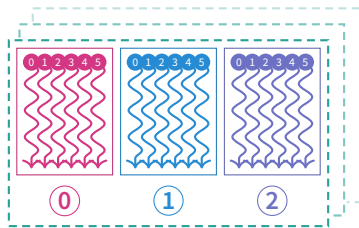
GPUs

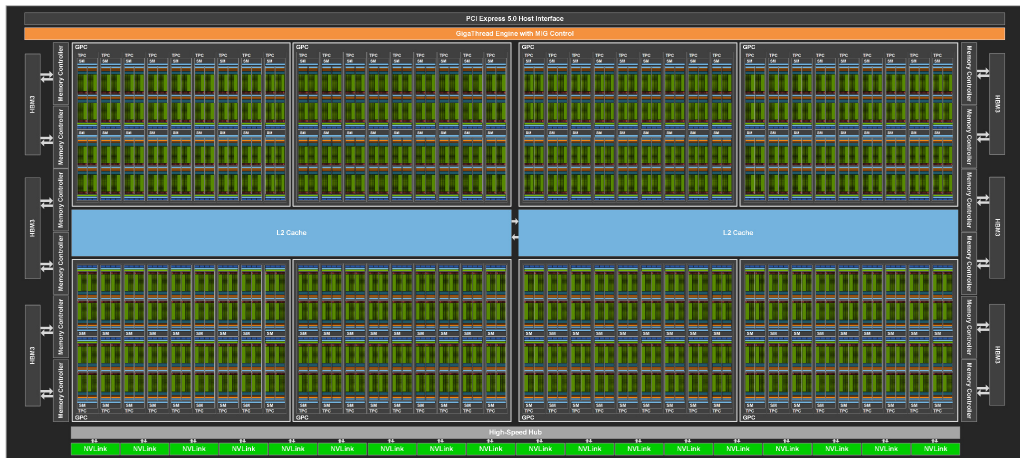
- **GPUs:** Exascale Enablers
- Processors efficient at applying same (/similar) instruction on large set of data (image)
- Over last 15 years, extended from rendering to variable computing
- Not good for every task, but **great for some**, which happen to be computing with large amounts of similar data
- Programming model: SIMT, SIMD $\otimes$ SMT (*vectors* $\otimes$ *threads*)
- JUWELS Booster thread 100 % occupancy: $3744 \text{ GPUs} \times 108 \text{ SMs} \times 2048 \text{ threads/SM} = 828\,112\,896 \text{ threads}$



GPUs

- **GPUs:** Exascale Enablers
- Processors efficient at applying same (/similar) instruction on large set of data (image)
- Over last 15 years, extended from rendering to variable computing
- Not good for every task, but **great for some**, which happen to be computing with large amounts of similar data
- Programming model: SMT, SIMD $\otimes$ SMT (*vectors* $\otimes$ *threads*)
- JUWELS Booster thread 100 % occupancy: $3744 \text{ GPUs} \times 108 \text{ SMs} \times 2048 \text{ threads/SM} = 828\,112\,896 \text{ threads}$
- Important vendors: First **NVIDIA**, then **AMD**; soon also **Intel**





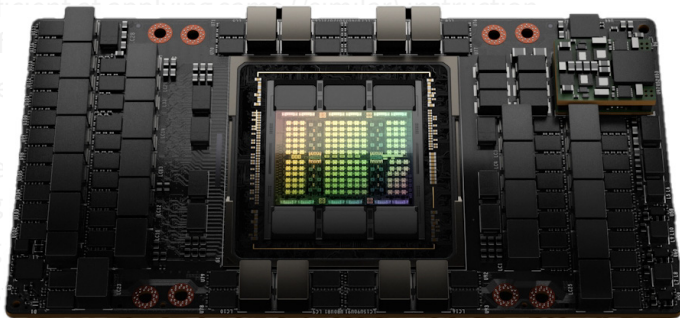
GPUs

- GPUs: Exascale Enabler
- Processors efficient at a on large set of data (ima
- Over last 15 years, exten computing
- Not good for every task, be computing with large
- Programming model: SL
- JUWELS Booster thread SMs $\times$ 2048 threads/SM
- Important vendors: Fir



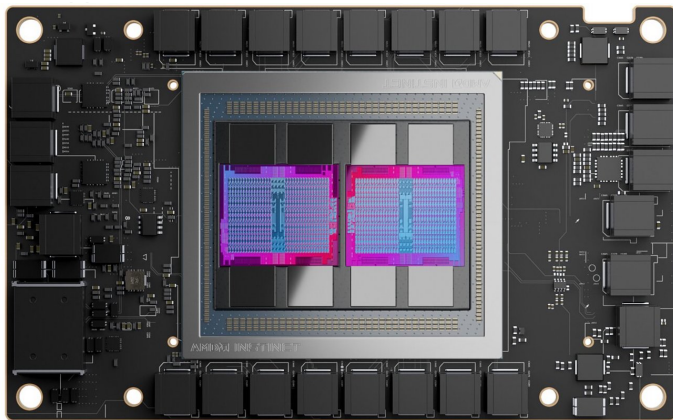
GPUs

- GPUs: Exascale Enablers
- Processors efficient at executing large number of (similar) instructions on large set of data
- Over last 15 years, GPUs have become dominant in high performance computing
- Not good for embedded computing
- Programming
- JUWELS Boost: 1024 SMs $\times$ 2048 threads/SM = 828 112 896 threads
- Important vendors: First NVIDIA, then AMD; soon also Intel



GPUs

- GPUs: Exascale
- Processors efficient on large set of
- Over last 15 years computing
- Not good for embedded computing
- Programming
- JUWELS Boost AI SMs $\times$ 2048 threads
- Important vendors: First NVIDIA, then AMD, soon also Intel



**High Performance Computing is
computing with a powerful machine
using the available resources efficiently.**

***High Performance Computing is
computing with a powerful machine
using the available resources efficiently.***

My interpretation.

Resource Utilization

1

Exploit all capabilities of processing entity (*core*)

Resource Utilization

1

Exploit all capabilities of processing entity (*core*)

2

Parallelize to all processing entities of node

Resource Utilization

1

Exploit all capabilities of processing entity (*core*)

2

Parallelize to all processing entities of node

3

Distribute to all nodes



1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsic*s), modern compilers, optimized libraries

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsic*s), modern compilers, optimized libraries
- Example: Vectorization/SIMD

A_0	$\times$	B_0	$+$	C_0	$=$	D_0
A_1	$\times$	B_1	$+$	C_1	$=$	D_1
A_2	$\times$	B_2	$+$	C_2	$=$	D_2
A_3	$\times$	B_3	$+$	C_3	$=$	D_3

1 Exploit All Capabilities of Processing Entity

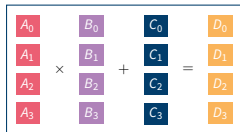
- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsic*s), modern compilers, optimized libraries
- Example: Vectorization/SIMD

A_0	$\times$	B_0	$+$	C_0	$=$	D_0
A_1	$\times$	B_1	$+$	C_1	$=$	D_1
A_2	$\times$	B_2	$+$	C_2	$=$	D_2
A_3	$\times$	B_3	$+$	C_3	$=$	D_3

$\times$ 4 multiplications
 $+$ 4 additions
 $=$ 4 assignments
 $\rightarrow$ 8 instructions

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



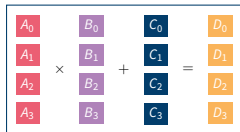
× 4 multiplications
+ 4 additions
= 4 assignments
→ 8 instructions



× 1 multiplication
+ 1 addition
= 1 assignment
→ 2 instructions

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



$\times$ 4 multiplications
 $+$ 4 additions
 $=$ 4 assignments
 $\rightarrow$ 8 instructions



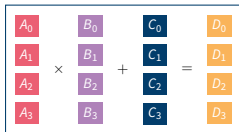
$\times$ 1 multiplication
 $+$ 1 addition
 $=$ 1 assignment
 $\rightarrow$ 2 instructions

CPU Instruction:
VADDPD

C Intrinsic:
`_mm256_add_pd();`

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



× 4 multiplications
+ 4 additions
= 4 assignments
→ 8 instructions

SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 2 instructions

FMA SIMD

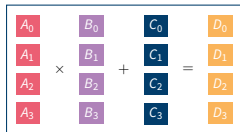
× 1 multiplication
+ 1 addition
= 1 assignment
→ 1 instruction

CPU Instruction:
VADDPD

C Intrinsic:
_mm256_add_pd();

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



× 4 multiplications
+ 4 additions
= 4 assignments
→ 8 instructions

SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 2 instructions

FMA SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 1 instruction

CPU Instruction:
VADDPD

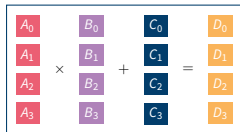
C Intrinsic:
`_mm256_add_pd();`

CPU Instruction:
VFMADD132PD

C Intrinsic:
`_mm256_fmadd_pd;`

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



× 4 multiplications
+ 4 additions
= 4 assignments
→ 8 instructions

SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 2 instructions

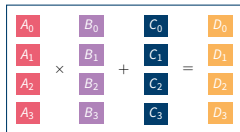
FMA SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 1 instruction

Compiler!

1 Exploit All Capabilities of Processing Entity

- Modern CPUs: many advanced instructions, high clock rate, large caches, high memory bandwidth
- Use via tailored algorithms, specific functions (*intrinsics*), modern compilers, optimized libraries
- Example: Vectorization/SIMD



× 4 multiplications
+ 4 additions
= 4 assignments
→ 8 instructions

SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 2 instructions

FMA SIMD

× 1 multiplication
+ 1 addition
= 1 assignment
→ 1 instruction

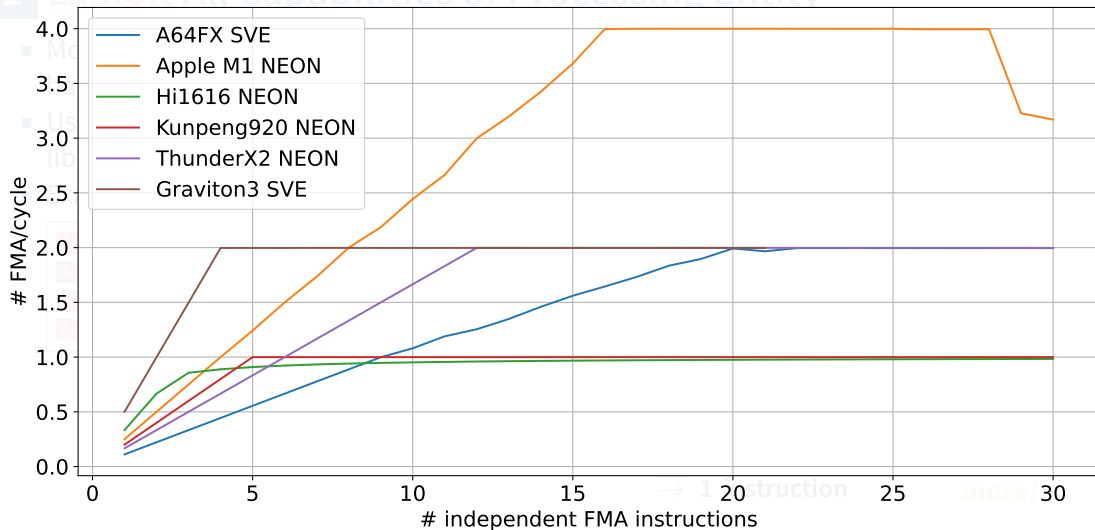
**Improve
throughput!**

Compiler!

**Improve
throughput
more!**

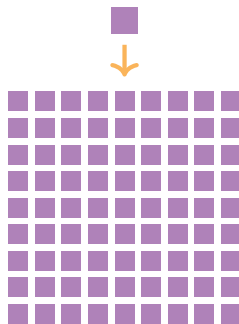
1

Exploit All Capabilities of Processing Entity



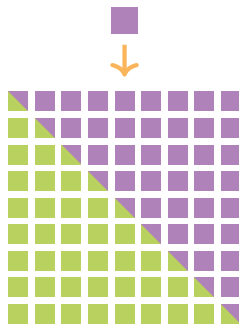
2 Parallelize to All Processing Entities of Node

- From core to cores



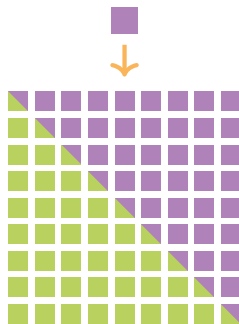
2 Parallelize to All Processing Entities of Node

- From core to cores
- From CPU cores to GPU cores



2 Parallelize to All Processing Entities of Node

- From core to cores
- From CPU cores to GPU cores
- **Parallelization:** Tasks work on portion of full problem using some local shared memory; *fine-grained* split



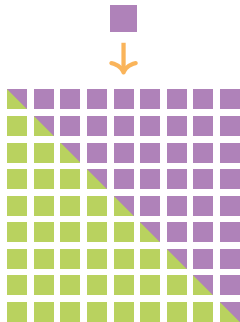
2 Parallelize to All Processing Entities of Node

- From core to cores
- From CPU cores to GPU cores
- **Parallelization:** Tasks work on portion of full problem using some local shared memory; *fine-grained* split

CPU Mostly through operating system capacities

- OS *threads* launched on *cores*
- Easiest threading interface: OpenMP

```
#pragma omp parallel for  
for (int i = 0; i < N; i++) y[i] = x[i] * 3.14 + a[i];
```



2 Parallelize to All Processing Entities of Node

- From core to cores
- From CPU cores to GPU cores
- **Parallelization:** Tasks work on portion of full problem using some local shared memory; *fine-grained* split

CPU Mostly through operating system capacities

- OS *threads* launched on *cores*
- Easiest threading interface: OpenMP

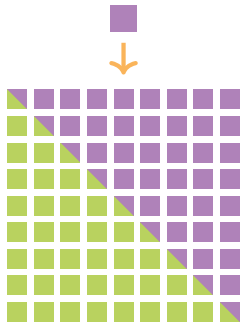
```
#pragma omp parallel for  
for (int i = 0; i < N; i++) y[i] = x[i] * 3.14 + a[i];
```

GPU Through dedicated programming environments

- Mostly, explicit models

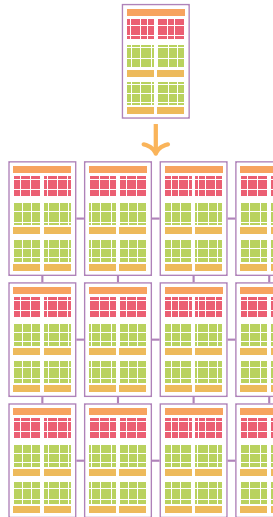
```
int i = threadIdx.x + blockIdx.x * blockDim.x;  
y[i] = x[i] * 3.14 + a[i];
```

- Also, higher-level models (OpenMP, OpenACC)



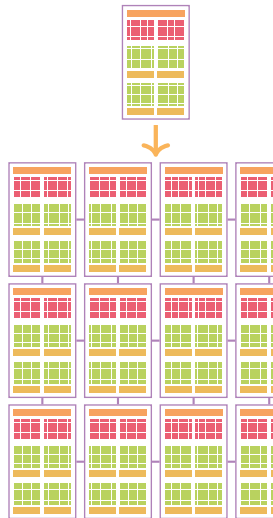
3 Distribute to All Nodes

- From node to nodes



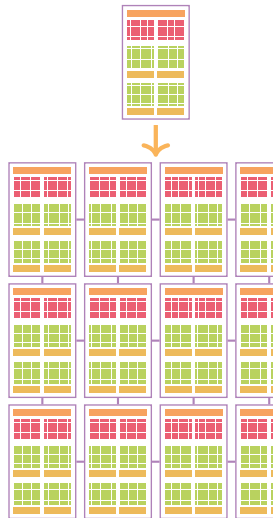
3 Distribute to All Nodes

- From node to nodes
- **Distribution:** Tasks work on portion of full problem using distributed memory; *coarse-grained* split



3 Distribute to All Nodes

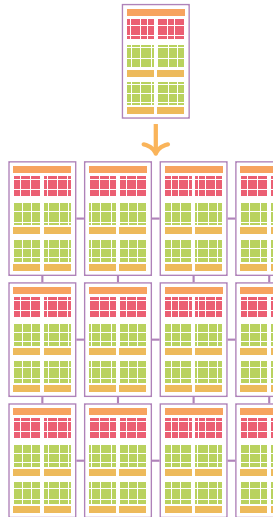
- From node to nodes
- **Distribution:** Tasks work on portion of full problem using distributed memory; *coarse-grained* split
- Every task runs on own node with copy of program, defined exchange functions
- High-speed network important! GPUs directly attached to network



3 Distribute to All Nodes

- From node to nodes
- **Distribution:** Tasks work on portion of full problem using distributed memory; *coarse-grained* split
- Every task runs on own node with copy of program, defined exchange functions
- High-speed network important! GPUs directly attached to network
- Classical programming model: **MPI**

```
MPI_Comm_size(MPI_COMM_WORLD, &size);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
MPI_Send(buffer, 10, MPI_INT, 1, 555, MPI_COMM_WORLD);  
MPI_Allreduce(local_sum, global_sum, 1, MPI_FLOAT, MPI_SUM,  
↪ MPI_COMM_WORLD);
```



The Gateway Heroes of HPC

Compilers **Translate** high-level code to low-level machine code, with general *and* very architecture-specific optimizations

The Gateway Heroes of HPC

Compilers

Translate high-level code to low-level machine code, with general *and* very architecture-specific optimizations

Frameworks

Offer pre-programmed **function primitives** to build a program upon



The Gateway Heroes of HPC

Compilers

Translate high-level code to low-level machine code, with general *and* very architecture-specific optimizations

Frameworks

Offer pre-programmed **function primitives** to build a program upon

Libraries

Back-end, low-level functions, usually optimized extensively, sometimes by vendors themselves

The Gateway Heroes of HPC

Compilers **Translate** high-level code to low-level machine code, with general *and* very architecture-specific optimizations

Frameworks Offer pre-programmed **function primitives** to build a program upon

Libraries **Back-end**, low-level functions, usually optimized extensively, sometimes by vendors themselves

Compilers

CPU GCC, LLVM, Intel, Cray

GPU + NVIDIA CUDA,
NVHPC, AMD

- Long history,
constantly evolving

Frameworks

MPI OpenMPI, MPICH

Threads pthreads, OpenMP

GPU CUDA, HIP, SYCL, pSTL,
Kokkos

Libraries

CPU MKL, BLIS, FFTW

GPU cuBLAS, rocBLAS,
cuDNN

→ TensorFlow, PyTorch,
ELPA

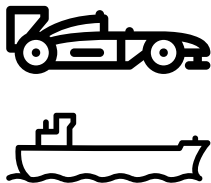
***High Performance Computing is
computing with a powerful machine
using the available resources efficiently.***

My interpretation.

Conclusion

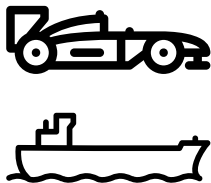
Conclusion

- HPC is intensive computing with largest machines
- Sometimes like Formula 1, sometimes like a tanker
- Sophisticated hardware is underlying everything, delivering up to 1.1 EFLOP/s
- Advanced software holds everything together and enables science at the frontiers



Conclusion

- HPC is intensive computing with largest machines
- Sometimes like Formula 1, sometimes like a tanker
- Sophisticated hardware is underlying everything, delivering up to 1.1 EFLOP/s
- Advanced software holds everything together and enables science at the frontiers, like
 - Plasma physics simulations
 - Drug discovery
 - Material design
 - Weather and climate modelling
 - Precise Artificial Intelligence

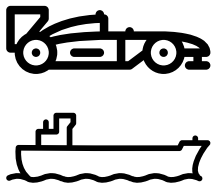


Images by lastspark and Dmitry Podluzny from Noun Project.

Conclusion

- HPC is intensive computing with largest machines
- Sometimes like Formula 1, sometimes like a tanker
- Sophisticated hardware is underlying everything, delivering up to 1.1 EFLOP/s
- Advanced software holds everything together and enables science at the frontiers, like
 - Plasma physics simulations
 - Drug discovery
 - Material design
 - Weather and climate modelling
 - Precise Artificial Intelligence

... or both!

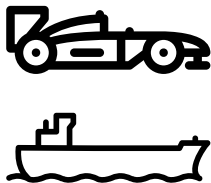


Images by lastspark and Dmitry Podluzny from Noun Project.

Conclusion

- HPC is intensive computing with largest machines
- Sometimes like Formula 1, sometimes like a tanker
- Sophisticated hardware is underlying everything, delivering up to 1.1 EFLOP/s
- Advanced software holds everything together and enables **science at the frontiers**, like
 - Plasma physics simulations
 - Drug discovery
 - Material design
 - Weather and climate modelling
 - Precise Artificial Intelligence
- We are hiring!
go.fzj.de/jsc-jobs

... or both!



Images by lastspark and Dmitry Podluzny from Noun Project.

Conclusion

- HPC is intensive computing with largest machines
- Sometimes like Formula 1, sometimes like a tanker
- Sophisticated hardware is underlying everything, delivering up to 1.1 EFLOP/s
- Advanced software holds everything together and enables **science at the frontiers**, like
 - Plasma physics simulations
 - Drug discovery
 - Material design
 - Weather and climate modelling
 - Precise Artificial Intelligence
- We are hiring!
go.fzj.de/jsc-jobs



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

... or both!



Appendix

text3

Appendix
License
References

License

This slide deck is published under the following license:
CC BY-SA 4.0



References: Images, Graphics I

- [1] Forschungszentrum Jülich GmbH (Ralf-Uwe Limbach). *JUWELS Booster*.
- [2] Control Data Corporation. *Picture: CDC 6600*. Computer History Museum. URL: <https://www.computerhistory.org/revolution/supercomputers/10/33> (pages 33, 34).
- [3] Sandia National Lab. *Picture: Intel XP/S 140*. Top500.org. URL: <https://www.top500.org/resources/top-systems/intel-xps-140-paragon-sandia-national-labs/> (page 41).
- [4] Forschungszentrum Jülich. *Picture: JUGENE*. JUGENE Press Release. URL: https://www.fz-juelich.de/de/aktuelles/news/pressemitteilungen/2007/index4763_hm (page 42).

References: Images, Graphics II

- [5] OLCF at ORNL. *Picture: Summit*. Flickr. URL: <https://www.flickr.com/photos/olcf/42659222181/> (page 43).
- [6] RIKEN. *Picture: Fugaku*. Fujitsu.com. URL: <https://blog.de.fujitsu.com/data-driven/fugaku-der-aktuell-weltweit-leistungsstaerkste-supercomputer/> (page 44).
- [7] OLCF at ORNL. *Picture: Frontier*. Flickr. URL: <https://www.flickr.com/photos/olcf/52117623843/>.
- [8] Nvidia Corporation. *Pictures: Ampere GPU*. Ampere Architecture Whitepaper. URL: <http://www.nvidia.com/nvidia-ampere-architecture-whitepaper> (pages 55–57).

References: Images, Graphics III

- [9] AMD Inc. *Pictures: Instinct MI250 GPU*. Promotional Material. URL: <https://videocardz.net/amd-instinct-mi250> (page 58).