CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Developing for Grace-Hopper

User lab day 2025

Sebastian Keller, Fabian Bösch, Jean Favre

Table of contents

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

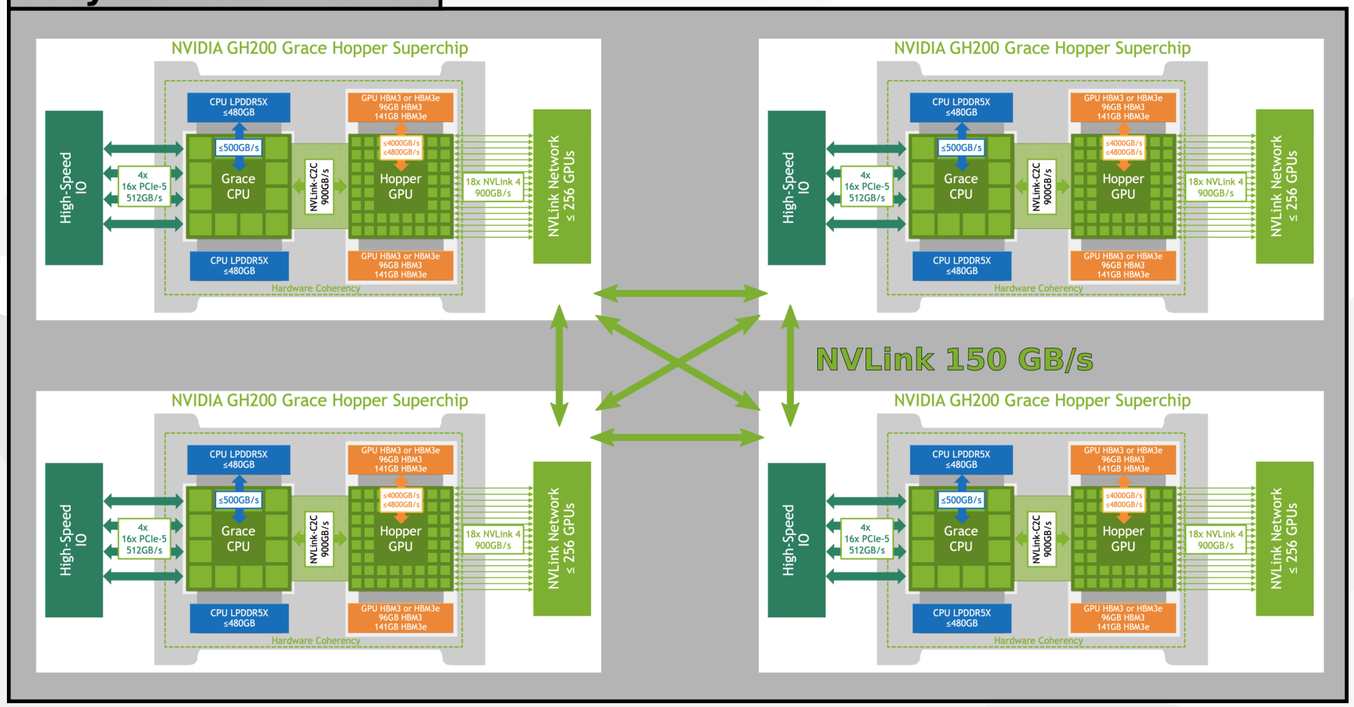
ETH zürich

Hardware overview

Grace-Hopper GH200

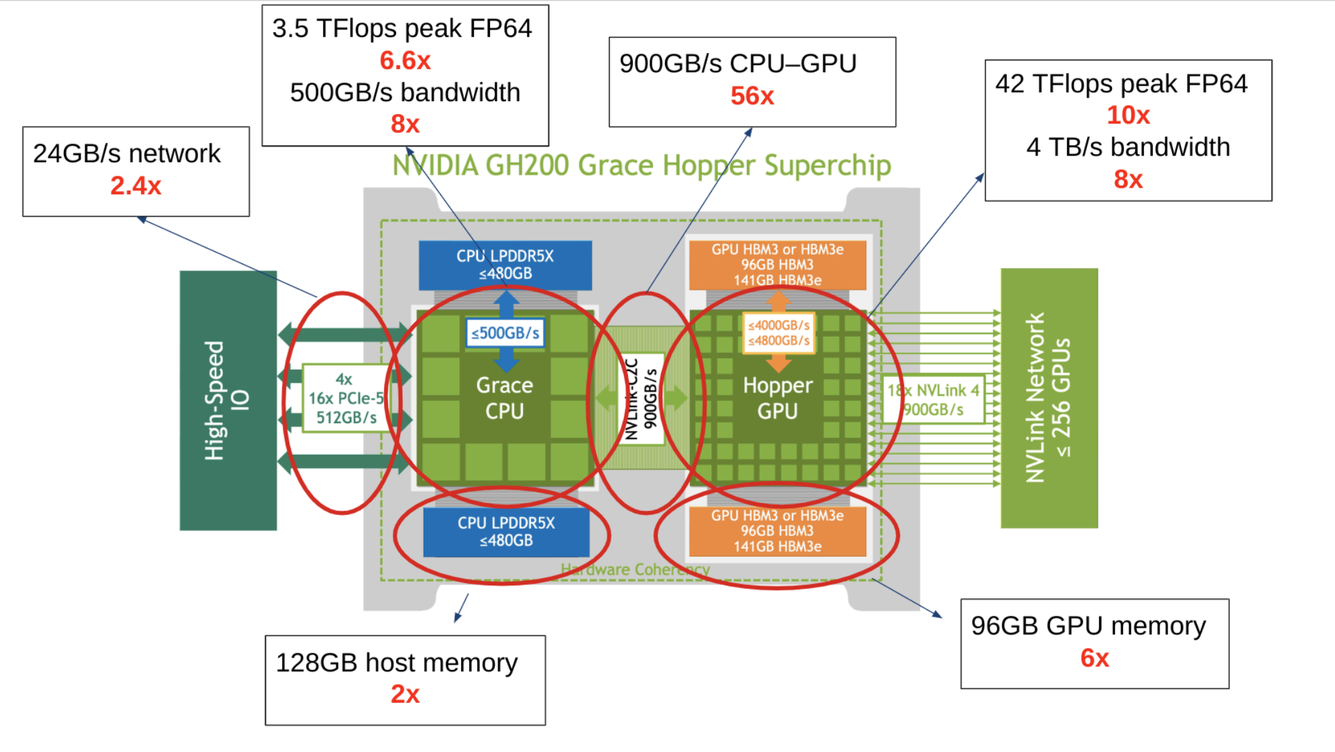
ALPS Grace-Hopper node

Cray EX GH200 node



- 4 modules with a CPU and GPU
- Fast NVLink connections with 150 GB/s between the modules
- 4 network cards with 25 GB/s

Grace-Hopper (GH200) module



- 72 ARM Grace CPU cores and one H100 GPU
- 128GB host memory and 96GB GPU memory
- Fast chip-to-top (C2C) link between CPU and GPU with 900GB/s

Four modules make a node

- All four modules form a single machine with 4x72 = 288 CPU cores
- All memory visible and allocatable by a single process:

```
daint-ln004:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:         877138       25955      837179        4092       21987      851183
```

- CPU and GPU memory is unified into a single address space
- **Both** CPU and GPU memory are allocatable and accessible from the CPU
- Each node has 4x128GB + 4x96GB = 896GB memory
- CPU and GPU memory from different modules are organized into different **NUMA** (non-uniform memory access) domains

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

SLURM best practices

Running on GH200: SLURM and affinity

- Affinity indicates which CPU cores the threads of a process/rank may run on
- CPU cores and GPUs are both assigned sequentially by SLURM to the 4 modules:

GH200 node	
Module 0: GPU `0` , CPUs `0-71`	**Module 1**: GPU `1` , CPUs `72-143`
Module 2: GPU `2` , CPUs `144-215`	**Module 3**: GPU `3` , CPUs `216-287`

SLURM affinity is configured with sensible defaults: each MPI rank within a node will be placed on a different module.

4 MPI ranks per node with up to 72 cores

This case automatically ends up with ideal affinity placement:

- No NUMA effects
- All threads of a rank bound to the same module
- Each rank gets a different GPU
- Use up to 72 threads per rank. Many applications will be faster with fewer.
- Sensible default programming model for new code on ALPS-GH200

```
#SBATCH --nodes=<numNodes>
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=72
#SBATCH --gpus-per-task=1   # Will set CUDA_VISIBLE_DEVICES, such that each rank gets a different GPU

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun ./the-app
```

Multiple ranks per GPU with MPS

- Executing multiple processes per GPU will work out of the box
- ⚠️ If you distribute ranks unevenly among the 4 GPUs, you will not get any errors (unless you run out of memory), just bad performance
- We also provide a wrapper script for the *CUDA Multi-process Service (MPS)* that needs to be called manually and can lead to better performance compared to distributing ranks to GPUs by setting `CUDA_VISIBLE_DEVICES` only.

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=32 # 32 MPI ranks per node
#SBATCH --cpus-per-task=8
#SBATCH --account=<account>

export OMP_NUM_THREADS=8

srun --cpu-bind=socket ./mps-wrapper.sh <code> <args>
```

The `mps-wrapper.sh` script is available on our knowledge base at
https://docs.cscs.ch/running/slurm/#multiple-ranks-per-gpu

Summary: SLURM and affinity

Node configuration	for optimal performance, remember to
4 ranks x 72 cores `--ntasks-per-node=4 -c72`	👉 use up to 72 threads 👉 check if fewer threads are faster
>4 ranks	👉 use the mps-wrapper script
1 rank	👉 write NUMA-aware code 👉 manage multiple GPUs in code

Refer to the CSCS knowledge base at https://docs.cscs.ch/software/sciapps/ for

- example batch submission scripts
- wrapper scripts

for applications like *CP2K, GROMACS, LAMMPS, NAMD, Quantum ESPRESSO, VASP*

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

NUMA domains of GH200

NUMA

What is Non-Uniform Memory Access (NUMA)?

- NUMA provides information about physical distances between domains and associated hardware resources

Example:

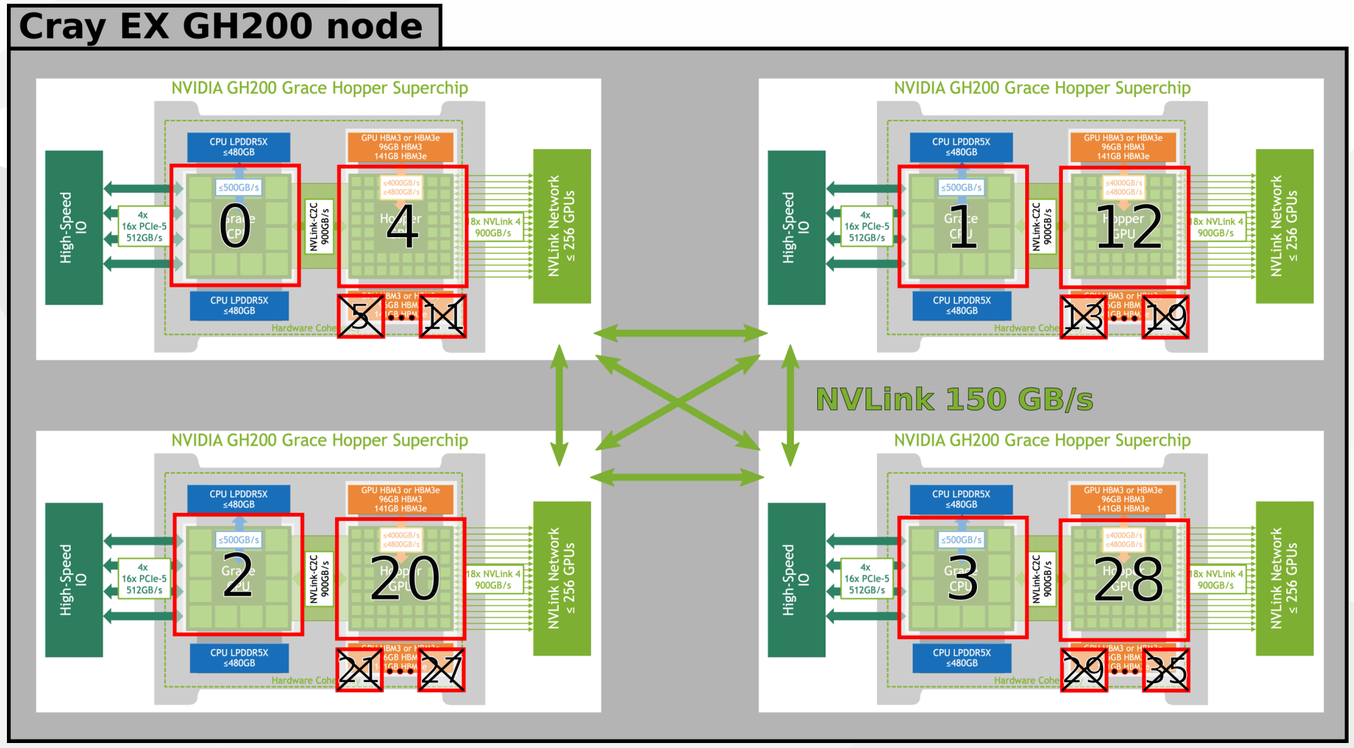
- Systems with **two CPU sockets** such as Piz Daint MC:
  - **two NUMA nodes:** memory associated with either of the sockets
  - accessing memory of the other socket is slower

Impact on developing high-performance multithreaded code:

- Either write NUMA-aware code (observing first touch-policy)
- or launch at least 1 MPI rank for each NUMA node

NUMA nodes of GH200

- **1 node per CPU**
- **1 node per GPU** (+7 empty, non-used nodes for multi-instance GPU, MiG)



NUMA nodes of GH200

```
daint-ln004:~$ numactl -H
available: 36 nodes (0-35)
node 0 cpus: 0-71              # CPU socket of module 0, 72 cores, 128GB RAM
node 0 size: 120162 MB
node 0 free: 105047 MB
node 1 cpus: 72-143            # CPU socket of module 1, 72 cores, 128GB RAM
node 1 size: 122663 MB
node 1 free: 117005 MB
node 2 cpus: 144-215           # CPU socket of module 2, 72 cores, 128GB RAM
node 2 size: 122663 MB
node 2 free: 118055 MB
node 3 cpus: 216-287           # CPU socket of module 3, 72 cores, 128GB RAM
node 3 size: 122529 MB
node 3 free: 108291 MB
node 4 cpus:                   # GPU of module 0, note: no CPU cores
node 4 size: 97280 MB          # 96GB RAM
node 4 free: 97117 MB
...
```

- `numactl -H` lists 36 numa domains (or `nodes`), but only 8 of them are in use:
  - `0,1,2,3` for the 4 CPU sockets
  - `4,12,20,28` for the 4 GPUs

NUMA nodes of GH200

`numactl -H` also shows connectivity information between domains.

```
daint-ln004:~$ numactl -H
available: 36 nodes (0-35)
[...]
node   0    1    2    3    4   12   20   28
  0:  10   40   40   40   80  120  120  120
  1:  40   10   40   40  120   80  120  120
  2:  40   40   10   40  120  120   80  120
  3:  40   40   40   10  120  120   20   80
  4:  80  120  120  120   10  255  255  255
 12: 120   80  120  120  255   10  255  255
 20: 120  120   80  120  255  255   10  255
 28: 120  120  120   80  255  255  255   10
```

There are 4 possible distances:

40: CPU-CPU, different modules
80: CPU-GPU, same module
120: CPU-GPU, different modules
255: GPU-GPU

Wait... NUMA nodes on GPUs?

Yes, that's right!

GH200 is the first hardware at CSCS with shared memory addresses between CPUs and GPUs.

- Memory allocated with host APIs like `malloc`, `new` and `mmap` can be **accessed by all CPUs and GPUs** in the compute node
- Physical memory placement decided by **first touch principle**
- **Automatic page migration**: repeated accesses to memory on a different NUMA node can cause memory to be migrated to the closest NUMA node
  - e.g. memory initialized on CPU, then repeatedly accessed on the GPU may be migrated from LPDDR5 to HBM3
- Memory allocated with `cudaMalloc` still **cannot** be accessed from the CPU and by other GPUs on the compute node

Passing system memory to GPU kernels

Memory allocated through host APIs can be passed to GPU kernels:

```
__global__ void add(const int* x, const int* y, int* z, size_t n)
{
    size_t tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < n) { z[tid] = x[tid] + y[tid]; }
}

size_t n = 1024;
std::vector<int> a(n), b(n), c(n); // uses `new` to allocate memory

// OK, pass system memory to
add<<<n/128, 128>>>(a.data(), b.data(), c.data(), n);
```

⚠️ The constructor of `std::vector` initializes elements with a single CPU thread, memory in this example will be placed in host-LPDDR5

Controlling NUMA memory placement

Placement of a memory page is decided by the NUMA node of the thread that first ***writes*** to it, not by the thread that allocates it.

```
__global__ void add(const int* x, const int* y, int* z, size_t n)
{
    size_t tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < n) { z[tid] = x[tid] + y[tid]; }
}

size_t n = 1024;
int* a = new int[n];
int* b = new int[n];
int* c = new int[n];
// or, see [1]:
// std::vector<int, DefaultInitAdaptor<int>> a(n), b(n), c(n);

cudaMemset(a, 0, n * sizeof(int)); // first touch on GPU, memory will be placed in H100-HBM3
cudaMemset(b, 0, n * sizeof(int));
cudaMemset(c, 0, n * sizeof(int));

add<<<n/128, 128>>>(a.data(), b.data(), c.data(), n);
```

[1] : https://github.com/sekelle/cornerstone-octree/blob/master/include/cstone/util/noinit_alloc.hpp

Summary: NUMA on GH200

- GPU memory allocated with `cudaMalloc` behaves the same as on previous machines; the **CPU and other GPUs can't access it.**
- System memory (`malloc` , `new` , ...) now **also accessible by GPUs**
- NUMA memory placement decided by the **first touch policy**, which can be tricky to get right
- Designing code to run with 1 rank per module still a reasonable choice in most cases
  - can still take advantage of unified memory between CPU and GPU
  - no need to manage multiple GPUs per process...
  - ... or deal with NUMA effects on CPUs

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Dealing with hardware and system issues

Dealing with hardware and system issues

- Unfortunately, compute nodes sometimes fail
- Problems will often manifest as MPI errors
- If you run jobs on hundreds of nodes, sooner or later you'll encounter some kind of failure

```
MPICH ERROR [Rank 305] [job id 352547.0] [Sat Aug 24 09:25:20 2024] [nid005606] -
Abort(5811599) (rank 305 in comm 0): Fatal error in PMPI_Init: Other MPI error, error stack:
MPIDI_CRAY_post_init(2315).................:
MPIDI_CRAY_ofi_check_nic_symmetry(3675).....:
MPIR_CRAY_Allreduce(577)....................:
MPIR_Allreduce_impl(363)....................:
MPIR_Allreduce_intra_auto(275)..............:
MPIR_Allreduce_intra_recursive_doubling(192):
MPIC_Sendrecv(341)..........................:
MPIC_Wait(71)...............................:
MPIR_Wait_impl(41)..........................:
MPID_Progress_wait(201).....................:
MPIDI_Progress_test(97).....................:
MPIDI_OFI_handle_cq_error(1075).............: OFI poll failed (ofi_events.c:1077:MPIDI_OFI_handle_cq_error:Input/output error - UNDELIVERABLE)
```

What to do after a failure

```
MPICH ERROR [Rank 305] [job id 352547.0] [Sat Aug 24 09:25:20 2024] [nid005606] -
Abort(5811599) (rank 305 in comm 0): Fatal error in PMPI_Init: Other MPI error, error stack:
MPIDI_CRAY_post_init(2315)..................:
MPIDI_CRAY_ofi_check_nic_symmetry(3675).....:
MPIR_CRAY_Allreduce(577)....................:
MPIR_Allreduce_impl(363)....................:
MPIR_Allreduce_intra_auto(275)..............:
MPIR_Allreduce_intra_recursive_doubling(192):
MPIC_Sendrecv(341)..........................:
MPIC_Wait(71)...............................:
MPIR_Wait_impl(41)..........................:
MPID_Progress_wait(201).....................:
MPIDI_Progress_test(97).....................:
MPIDI_OFI_handle_cq_error(1075).............: OFI poll failed (ofi_events.c:1077:MPIDI_OFI_handle_cq_error:Input/output error - UNDELIVERABLE)
```

1. Identify the failed compute node 👉 `nid005606`

What to do after a failure

2. Check the status of that node with `scontrol show nodes`:

```
NodeName=nid005606 CoresPerSocket=72
   CPUAlloc=0 CPUEfctv=288 CPUTot=288 CPULoad=0.00
   AvailableFeatures=gh,gpu
   ActiveFeatures=gh,gpu
   Gres=gpu:4
   NodeAddr=nid005606 NodeHostName=nid005606
   RealMemory=460000 AllocMem=0 FreeMem=N/A Sockets=4 Boards=1
   State=DOWN+NOT_RESPONDING ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   ...
```

3. What's the `State` of the failed node?

- `DOWN` or `DRAIN`: The queueing system recognized the failure and removed the node
  👉 Resubmit job, no further action required
- `IDLE` or `ALLOCATED`: Broken node did not get removed
  👉 Resubmit job and exclude the node in SLURM with `#SBATCH -x nid005606`

  Job running successfully with one or several nodes exluded?
  👉 Notify CSCS by opening a ticket

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich