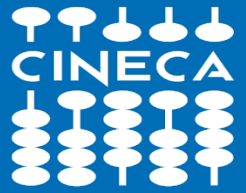
Introduction to the CINECA Marconi100 HPC system

May 29, 2020

Isabella Baccarelli

i.baccarelli@cineca.it

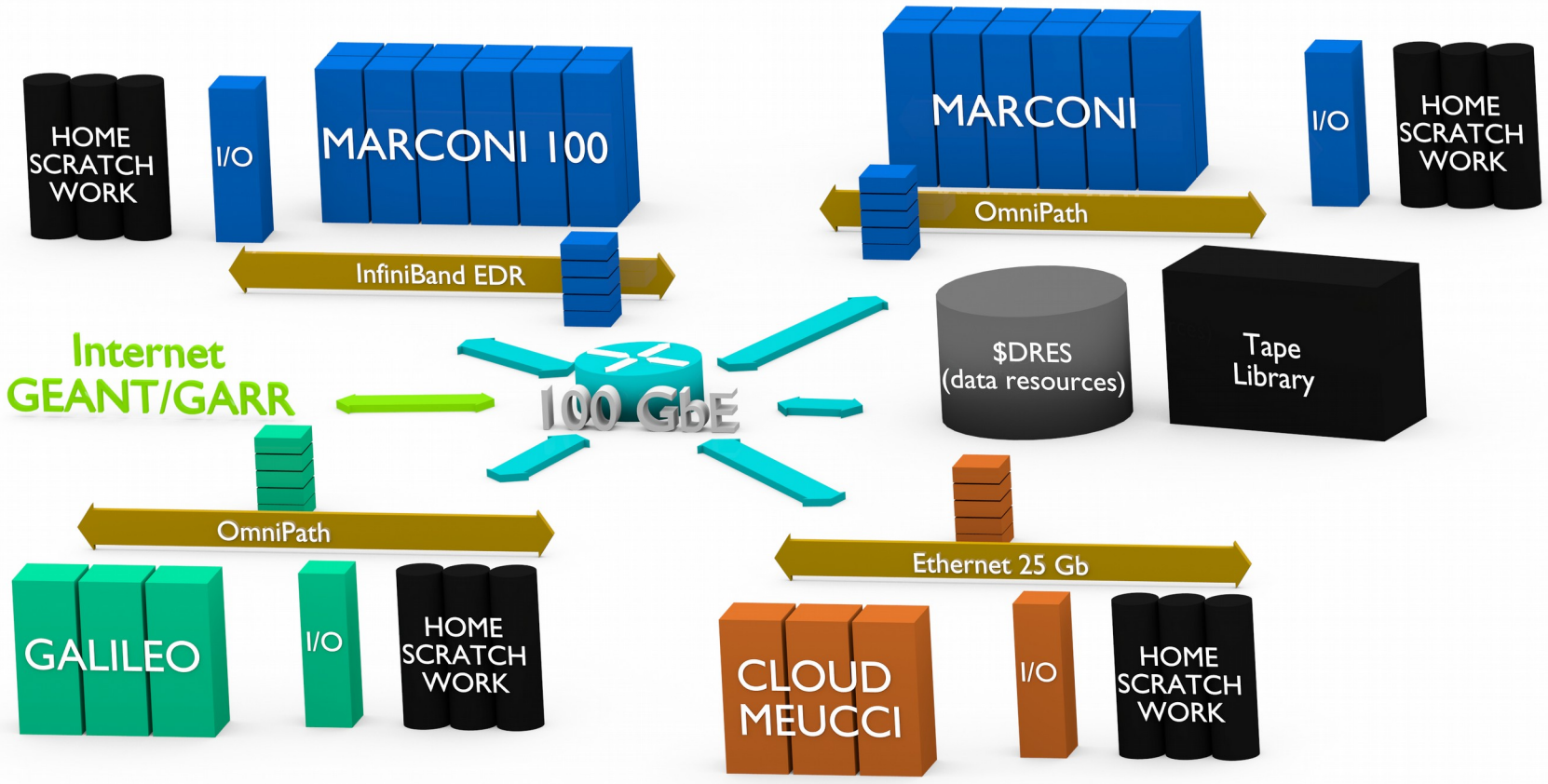
SuperComputing Applications and Innovations (SCAI) – High Performance Computing Dept



Outline

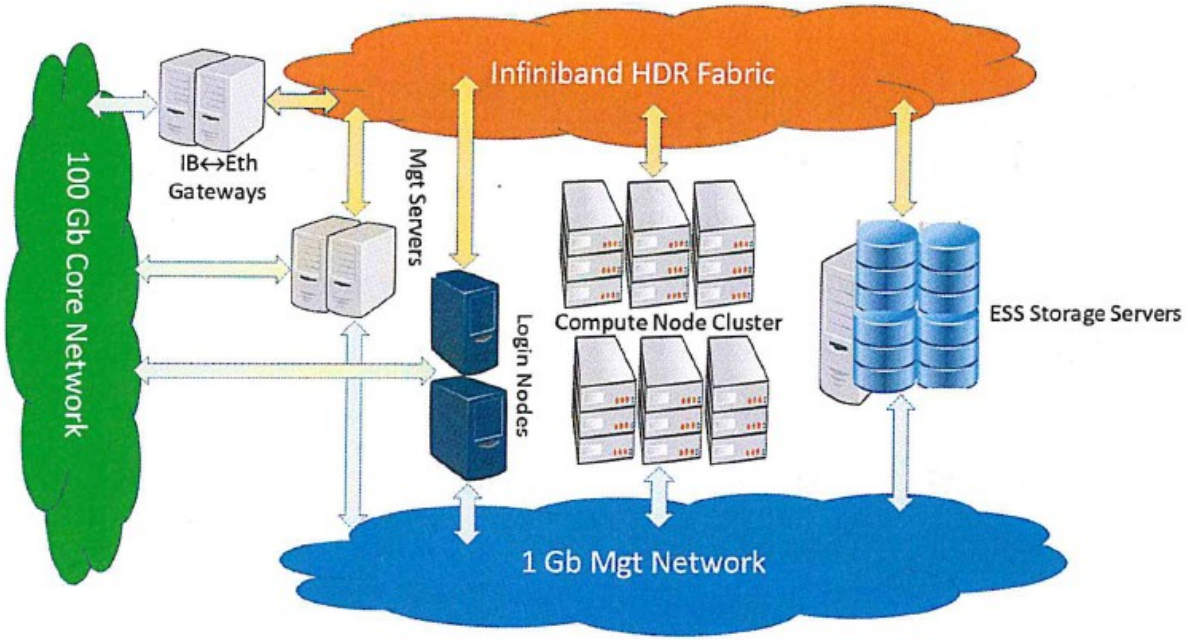
- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnections)
- Software environment
- Programming environment
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- Final remarks

CINECA Infrastructure



M100 Infrastructure: how to access

The access to M100 is granted to users with approved projects for this platform (Eurofusion, Prace, Iscra B and C, European HPC programs,…).
Eurofusion community has **80 dedicated nodes** of M100



- New Users:
  - **Register on the User Portal UserDB**
    userdb.hpc.cineca.it
  - **Get associated to an active project on M100**:
    - Principal Investigators (PIs): automatically associated if registered on UserDB (otherwise inform superc@cineca.it once done)
    - Collaborators: ask your PI to associate you to the project
  - Fill the "**HPC Access**" section on UserDB
  - **Submit your request** for the HPC Access (from UserDB)
  - You will receive your credentials by e-mail in the next few working hours (Note: the way to get access to the machine will change in the near future)

M100 Infrastructure: how to access

```
$ ssh -X username@login.m100.cineca.it
**********************************************************************
**
* Welcome to MARCONI100 Cluster /
*
*           IBM Power AC922 (Whiterspoon) -
*
*           Red Hat Enterprise Linux Server release 7.6 (Maipo)
*
etc. etc.
```

- Short system description
- “In evidence” messages
- “Important messages” (changes of policies, maintenances, etc.)

Access by **public keys** (with the ssh keys generated on a local and **secure** environment, and protected via passphrase) is strongly recommended

Outline

- CINECA infrastructure and Marconi100
- **System architecture (CPUs, GPUs, Memory, Interconnects)**
- Software environment
- Programming environment
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- Final remarks

Marconi100: the Power AC922 model

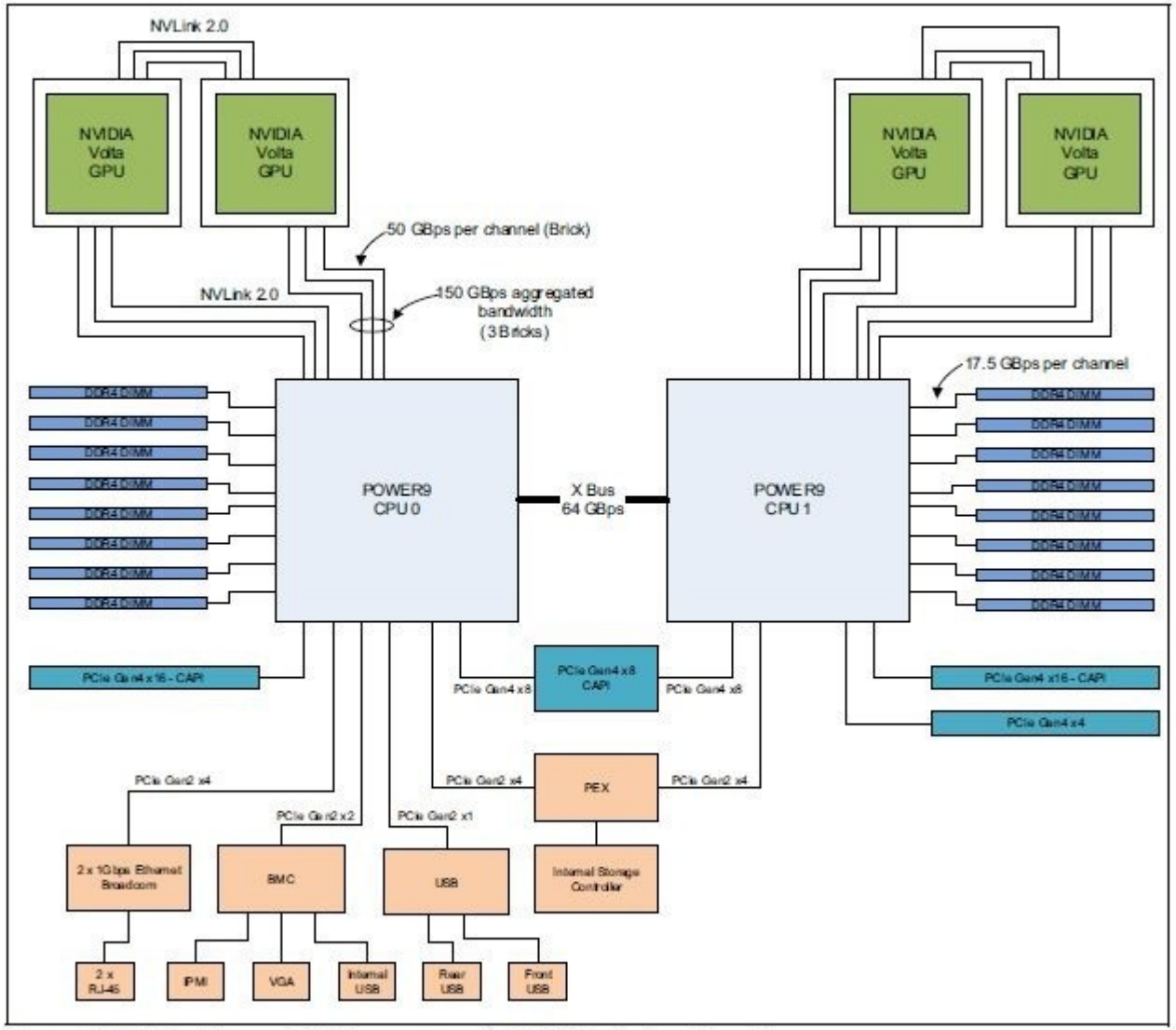


Figure 2-5 The Power AC922 server model GTH logical system diagram

- AC922 “Whiterspoon”
- **32 PFlops peak**
- Nodes: 980 compute + 3 login nodes, 32 TFlops each
- Processors: 2x16 cores IBM 8335-GTG 2.6 (3.1) GHz
- Accelerators: **4xNVIDIA V100 GPUs**, Nvlink 2.0, 16GB
- RAM: 256 GB/node
- Local disk: 1.6TB NVMe
- Internal Network: Mellanox Infiniband EDR DragonFly+
- Disk Space: 8PB storage

Power AC922 CPUs

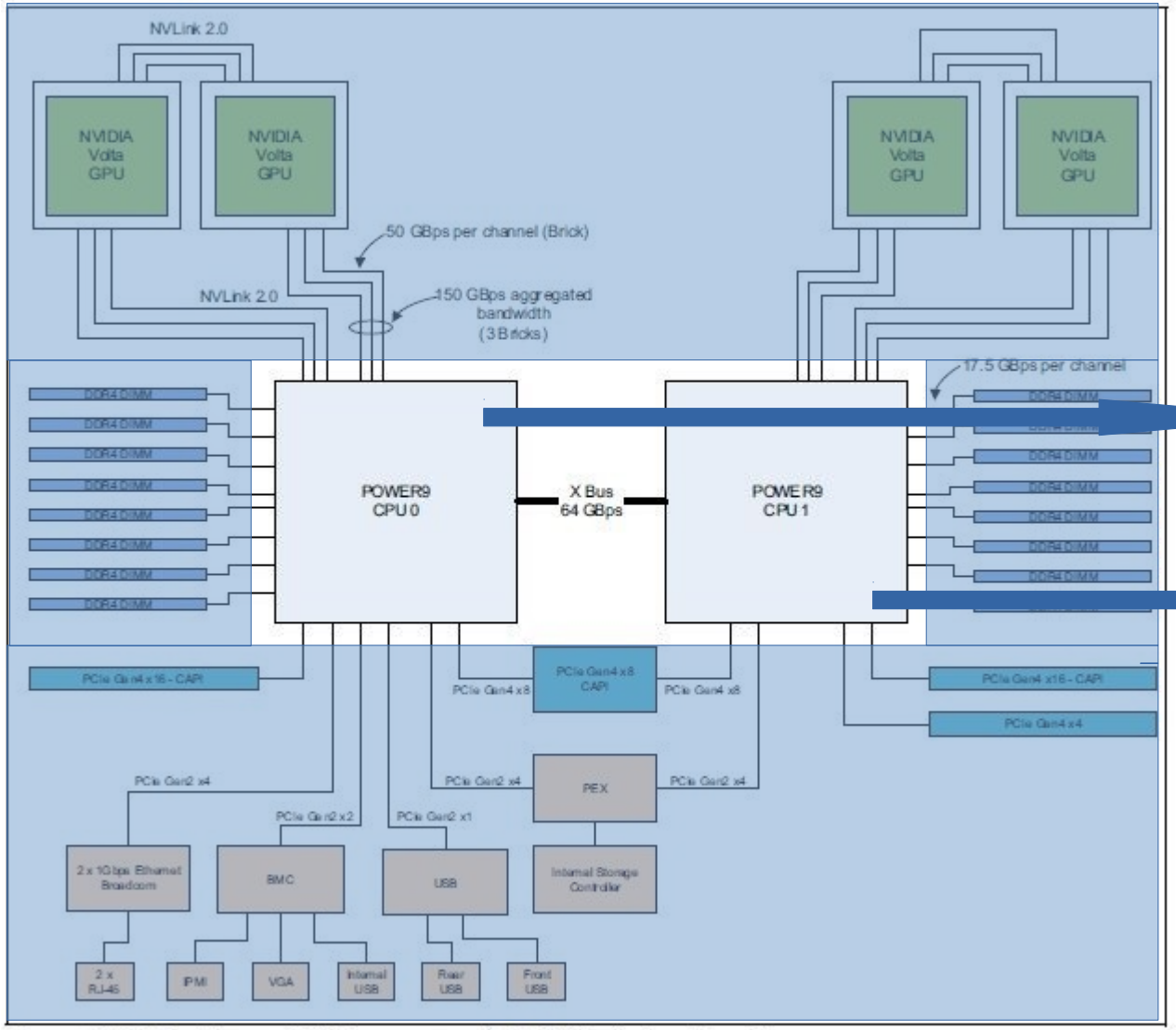


Figure 2-5 The Power AC922 server model GTH logical system diagram

- 2 Power9 sockets
- Each with 16 cores @ 2.6 GHz (3.1 GHz peak)
- SMT (Hardware threads per core): 4

$ ppc64_cpu –info

```
Core   0:    0*   1*   2*   3*
Core   1:    4*   5*   6*   7*
        (Cores from 2 to 13)
Core  14:   56*  57*  58*  59*
Core  15:   60*  61*  62*  63*
```

CPU 0

```
Core  16:   64*  65*  66*  67*
Core  17:   68*  69*  70*  71*
        (Cores from 18 to 29)
Core  30:  120* 121* 122* 123*
Core  31:  124* 125* 126* 127*
```

CPU 1

- Performance: ~0.8 Tflops per node

Power AC922 V100 (Volta) GPUs

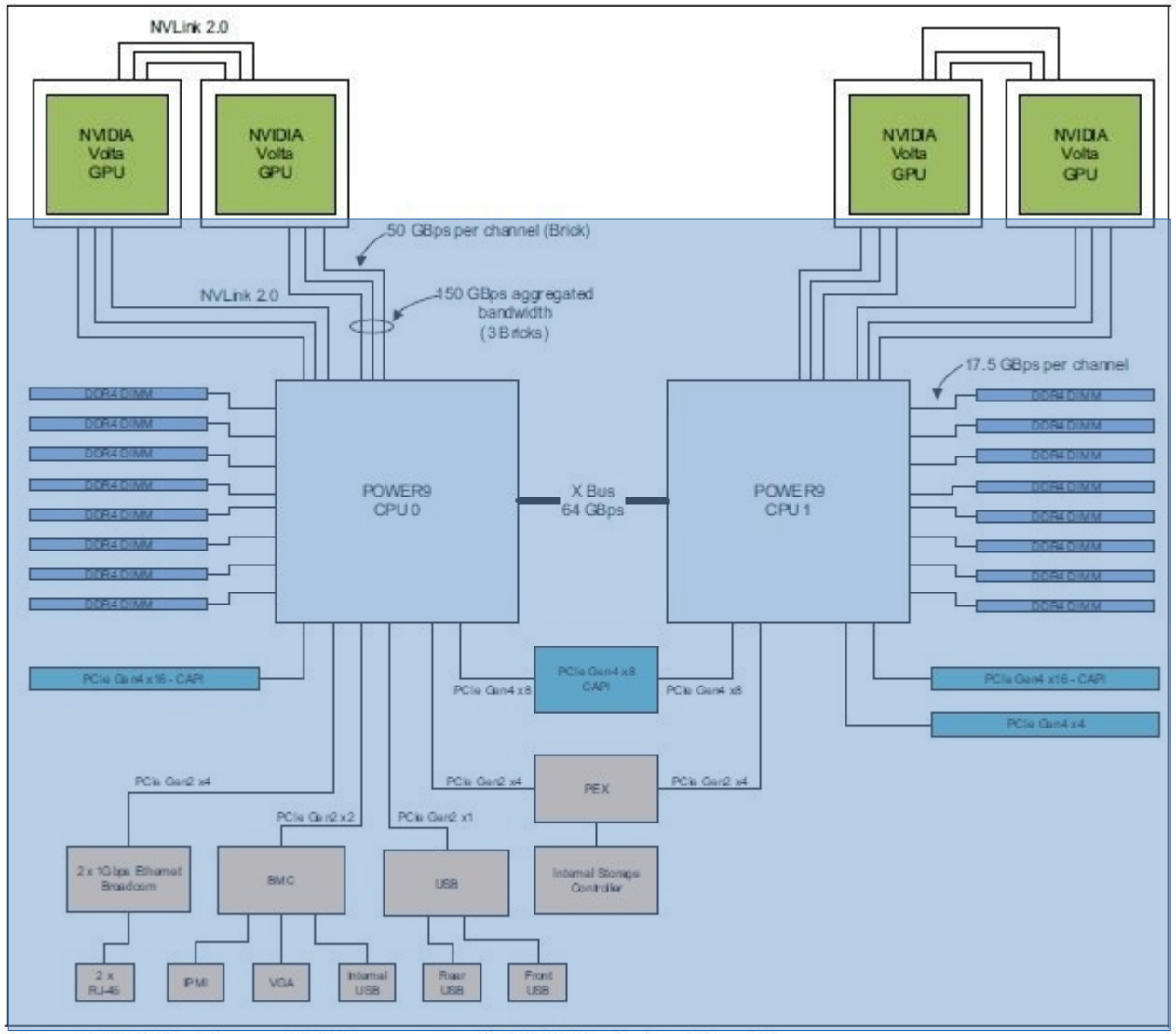


Figure 2-5 The Power AC922 server model GTH logical system diagram

- **Compute acceleration units**
- 4 V100 GPUs per node
- **Performances per GPU:**
  - **7.8 TFLOPS FP64**
  - **15.7 TFLOPS FP32**
  - **125 Tensor TFLOPS**
- nvidia-smi (NVIDIA System Management Interface)
- DeviceQuery (from CUDA samples, load the cuda module to access the command)

A digression: what is a GPU - 1

- **G**raphics **P**rocessing **U**nit

  a device equipped with an highly parallel microprocessor (*many-core*) and a private memory with very high bandwidth

- born in response to the growing demand for high definition 3D rendering graphic applications

A digression: what is a GPU - 2

- GPUs are designed to render complex 3D scenes composed of ***milions of data*** points/vertex in a very fast frame rate (60-120 FPS)
- the rendering process requires a set of transformations based on linear algebra operations and (mostly local) filters
- the ***same set* of operations** are applied on each point/vertex of the scene
- each operation is ***independent*** with respect to data
- all **operations are performed *in parallel*** using a **huge number of *threads*** which process all data independently

GPUs are specialized for parallel intensive communication

A digression: what is a GPU - 3


```
// typical loop over each point with the same set of operation
for each point in collection_of_points:
  output_data = trasformations_on_point(point, input_data)
```

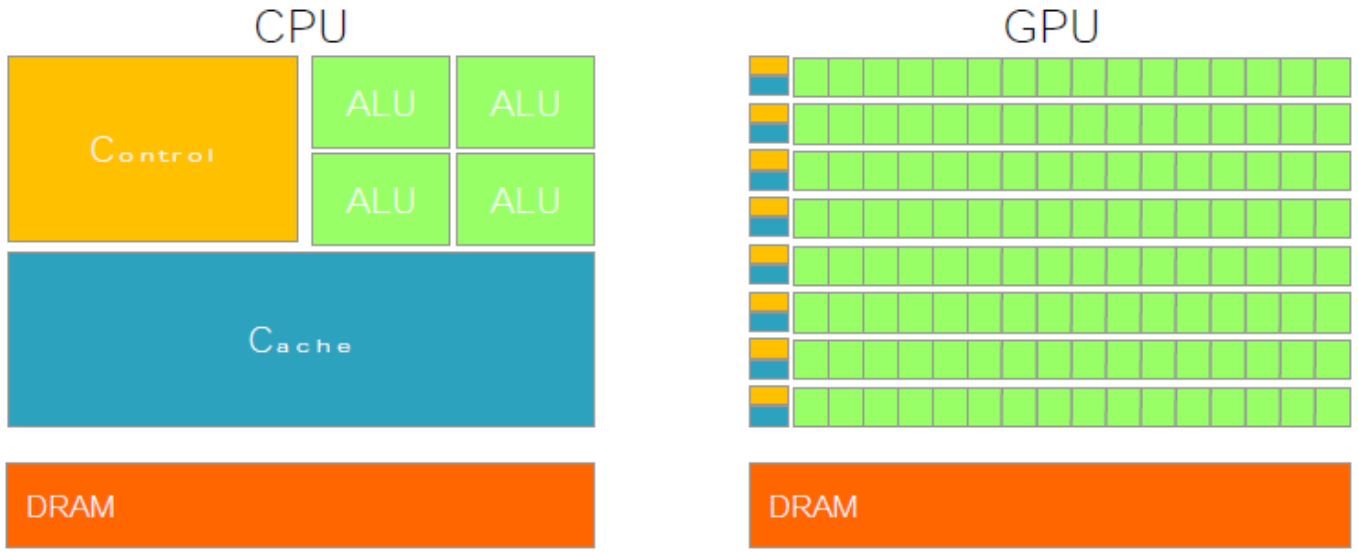
- If the set of tranformations can be applied independently on each point, the output result is independent on the order of point computation
- If transformations are independent, we can speed up the elaboration:
  - apply the same transformation (Single Program) ...
  - ... to each point (Multiple Data)
  - ... using multiple threads concurrently

Parallelism of Single Program Multiple Data (SPMD)

A digression: what is a GPU – 4

CPU vs GPU Architectures

- GPU are specialized for *intense data-parallel computations* where the same set of operations are executed on different data
- GPU hardware is designed so that more transistors are devoted to data processing rather than data caching and flow control

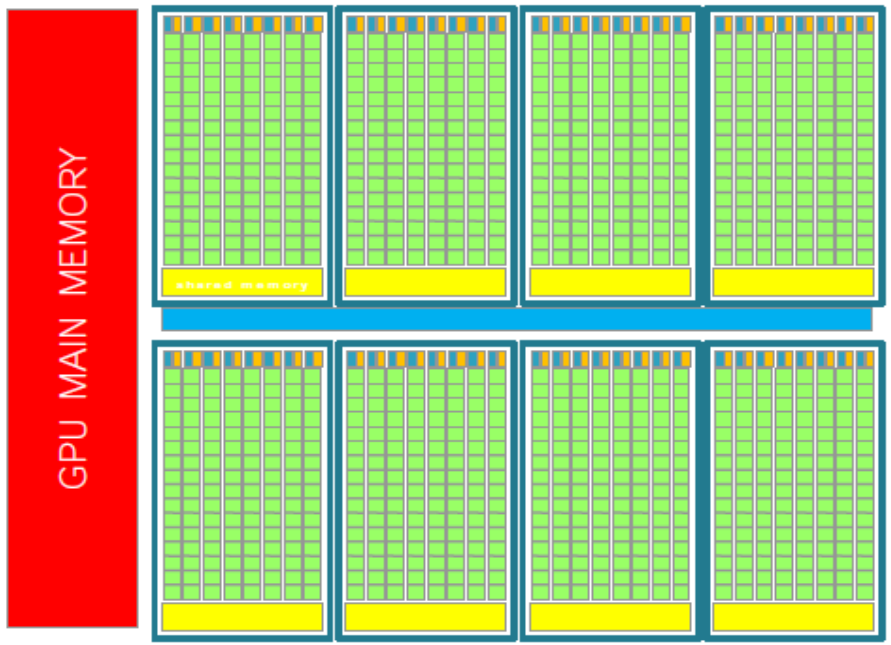


A digression: what is a GPU – 5

GPU Architecture scheme

A tipical GPU architecture consists of

- Main Global Memory
  - medium size (16 - 32 GB)
  - very hgh bandwidth (250-800 GB/s)
- Streaming Processors (SM)
  - grouping independent cores and control units
- each SM unit has
  - many ALU cores (> 100 cores)
  - lots of registers (32K-64K)
  - instruction scheduler dispatchers
  - a shared memory with very fast access to data



Power AC922 V100 (Volta) GPUs

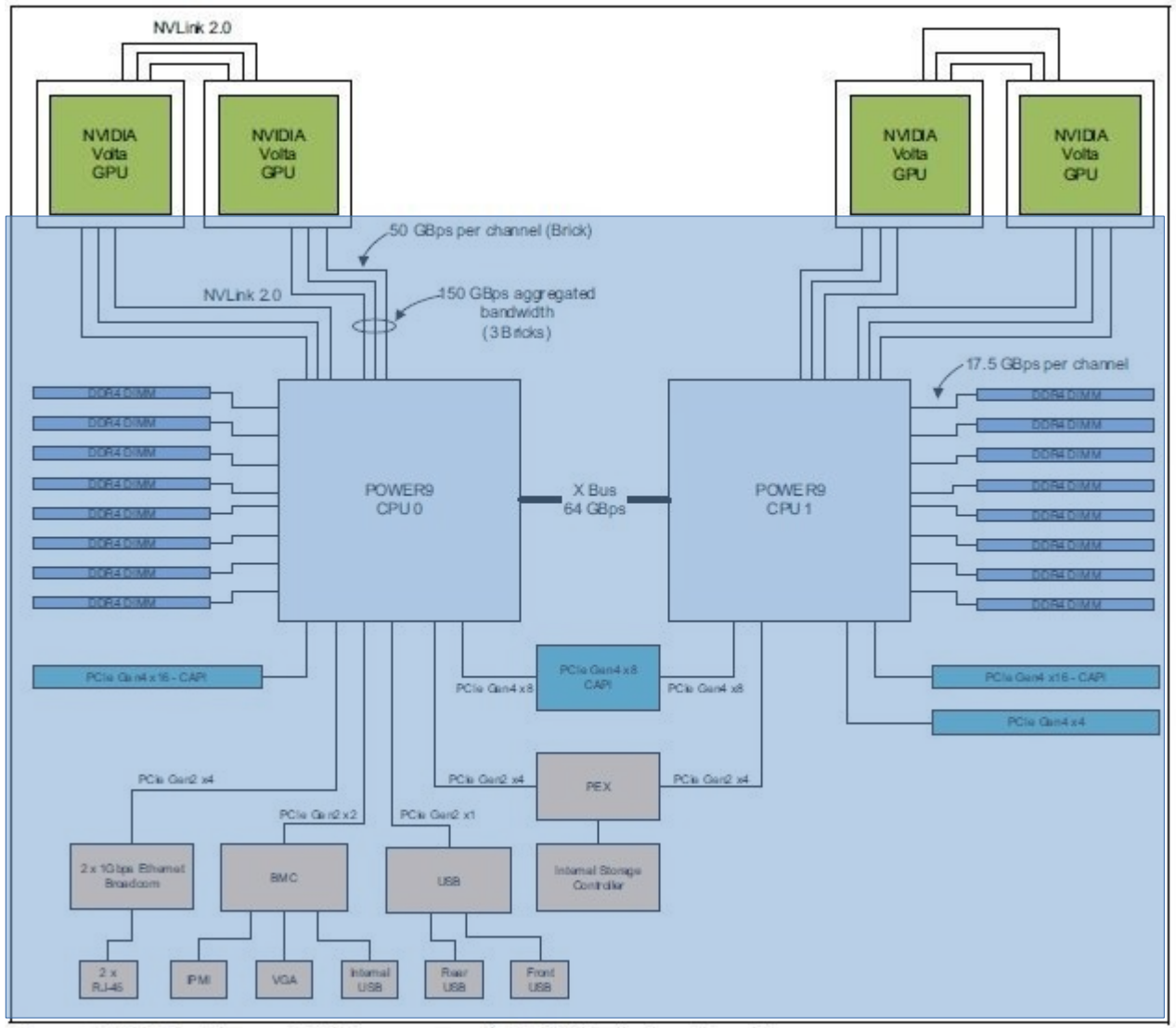


Figure 2-5 The Power AC922 server model GTH logical system diagram

- **Compute acceleration units**
- 4 V100 GPUs per node
- For each GPU:
  - 80 Volta Streaming Multiprocessors (SMs)
    - 64 FP32 cores
    - 64 INT32 cores
    - 32 FP64 cores
    - 640 Tensor cores
  - 16 GiByte High Bandwidth Memory

- **Performances per GPU:**
  - **7.8 TFLOPS FP64**
  - **15.7 TFLOPS FP32**
  - **125 Tensor TFLOPS**

Power AC922 Memory

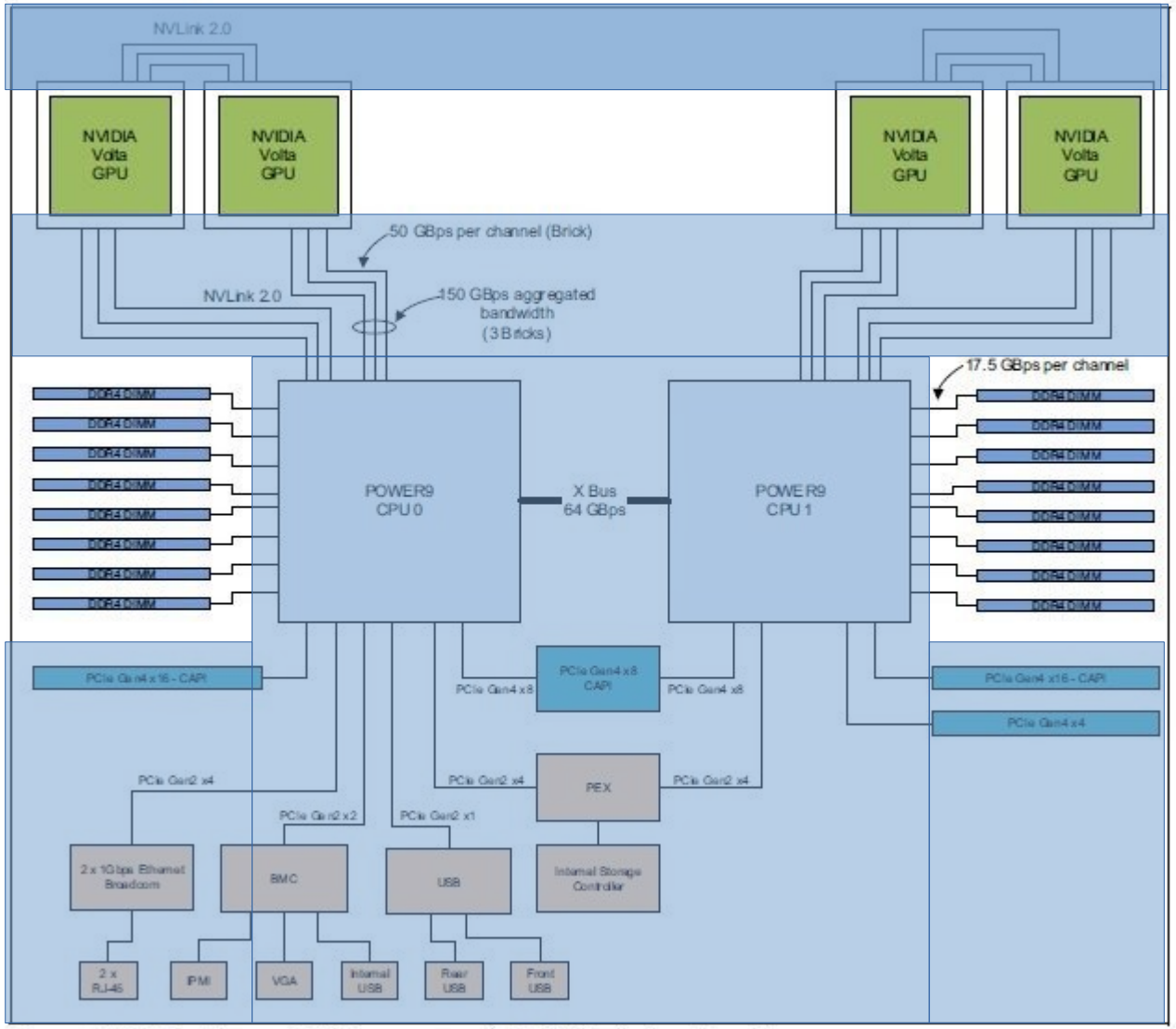


Figure 2-5 The Power AC922 server model GTH logical system diagram

- 16 GiByte High Memory Bandwidth per GPU
- 4096-bit interface (8x 512-bit memory controllers)
- Peak Memory Bandwidth ~900 GB/s

- 256 GB (16xDDR4 DIMM)
- 8 channels @ 17.5 GB/s
- Memory Bandwidth ~ 240 GB/s

UVAs and ATS features

Enable threads running on CPU cores and on GPU SMs to access **ANY** of the AC922 memory Pools (NUMA domains)

Power AC922 Memory NUMA nodes and UVAs

Unified Virtual Address Space (UVAs) and **Address Translation Services (ATS)**

Enable threads running on CPU cores and on GPU SMs to access **ANY** of the AC922 memory pools (NUMA domains)

$ numactl -H
available: 6 nodes (0,8,252-255)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
...size...
node 8 cpus: 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
node 252 cpus:
node 252 size: 16128 MB
node 252 free: 16128 MB
node 253 cpus:
node 254 cpus:
node 255 cpus:

Note: a series of patches have been accepted by the Linux Kernel community providing support for ATS and coherent Memory access to GPU RAM pages by a host task and host RAM pages by a GPU kernel

GPU device memory visibile as NUMA memory to the host

Power AC922 Intra-node interconnect

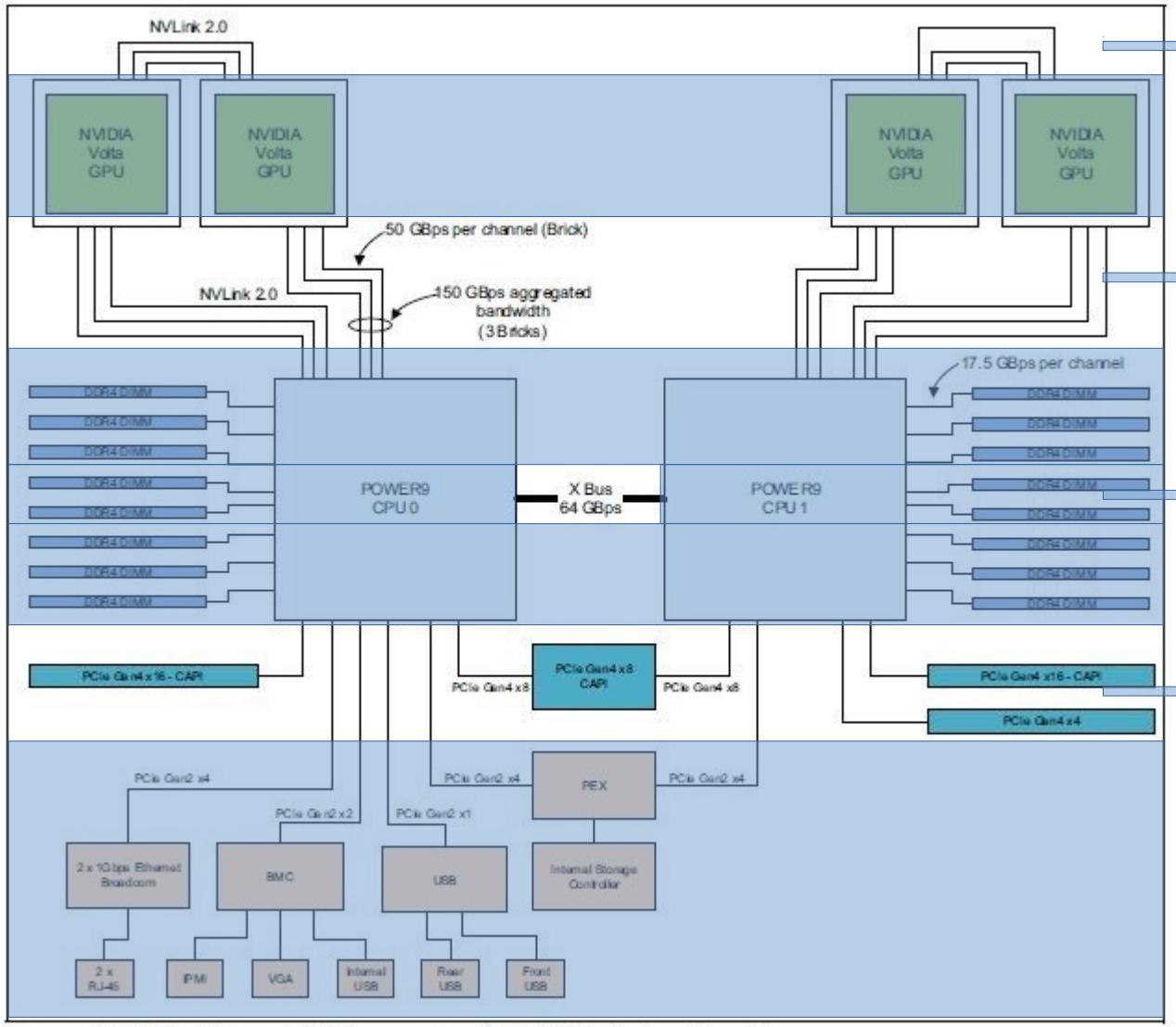


Figure 2-5 The Power AC922 server model GTH logical system diagram

NVLink 2.0

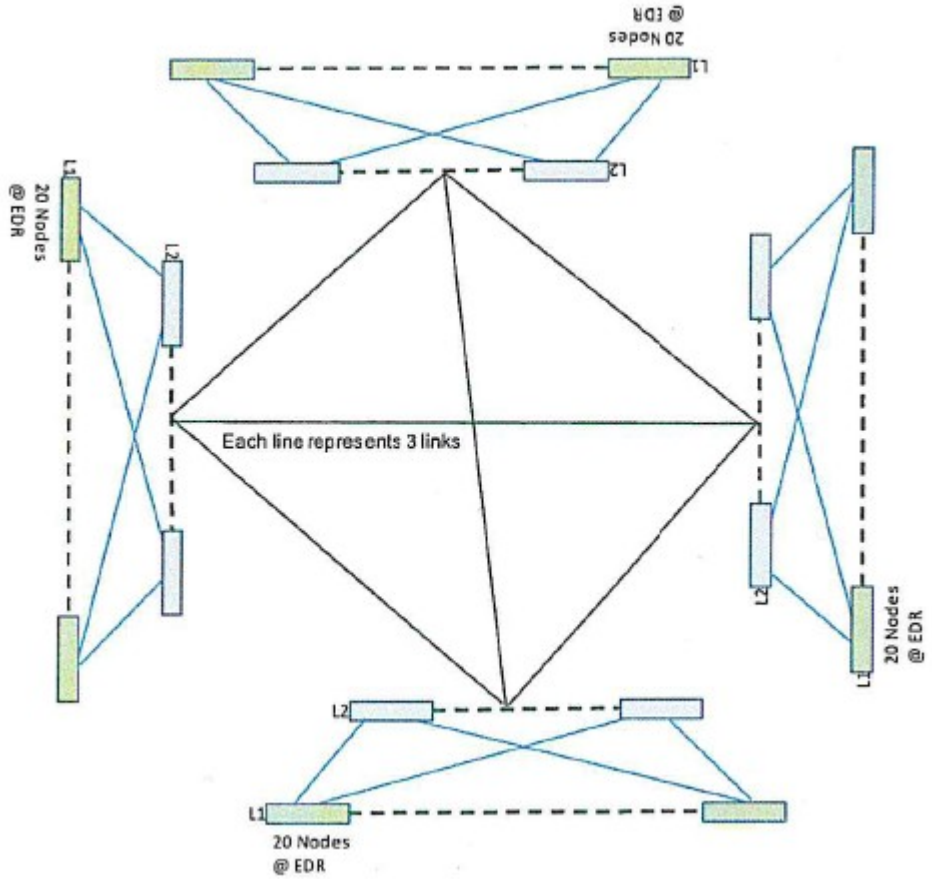
- NVIDIA **high-speed coherent interconnect** Technology **GPU-to-GPU and GPU-to-CPU**
- 3 NVLink bricks GPU-GPU and GPU-CPU Per socket
- **150 GiByte/s** peak bidirectional bandwidth between each of the three components

X bus

- Intersocket memory access
- **64 GB/s** peak bidirectional bandwidth

PCIe Gen 4

M100 System Interconnects
DragonFly+ Mellanox Infiniband EDR



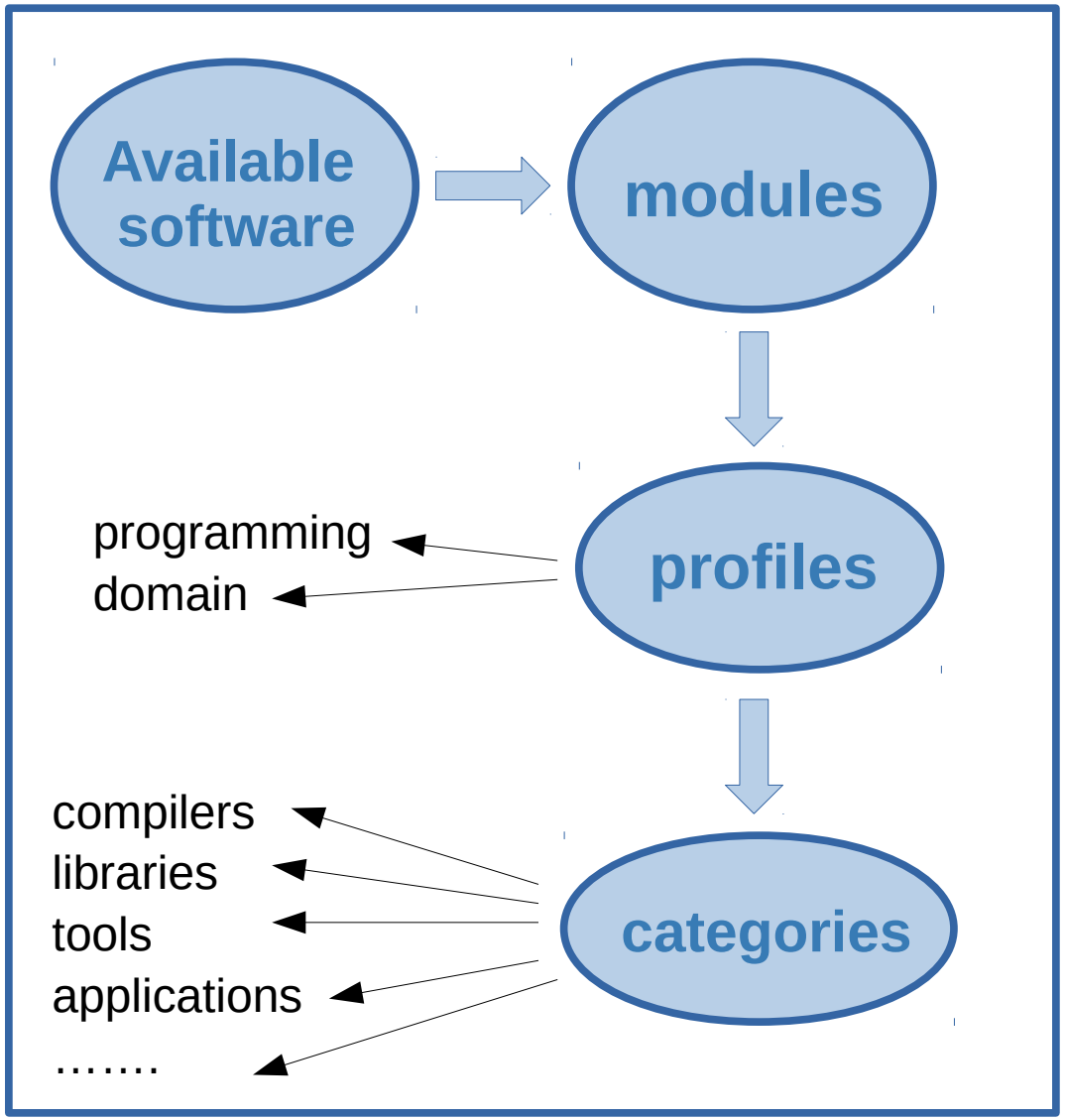
- 4 islands of 13 compute racks (up to 260 total) grouped using 20 downlinks and 16 uplinks
- Linked together by multiple EDR links
- **Fully nonblocking communication between groups**
- Adaptive routing algorithms (prevent contention and provide failover or fast re-routing in case of hardware failures) – WORK IN PROGRESS (next M100 maint?)

Each compute node is connected to the IB fabric with a single EDR adapter placed in the **shared PCIe x16 socket**. the shared socket provides one x8 connections to each processor Power9 socket => symmetric network connection to all MPI ranks on both POWER9 processors.

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- **Software environment**
- Programming environment
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- Final remarks

M100 Module Software Environment



The available software is offered in a module environment

The modules are collected in different profiles and organized in functional categories

Profile types:

- Programming (**base**, advanced): compilation, debugging, profiling, libraries
- Domain (chem-phys, lifesc, …): production activities

The **base** profile is the default:

- automatically loaded after login
- contains basic modules for the programming activities

M100 Module Software Env: base

$ module av

---------- /cineca/prod/opt/modulefiles/profiles ----------
profile/advanced **profile/base** profile/chem-phys profile/global profile/archive profile/candidate profile/deeplrn profile/lifesc

---------- /cineca/prod/opt/modulefiles/**base**/environment ----------
autoload

---------- /cineca/prod/opt/modulefiles/**base**/libraries ----------
blas/3.8.0--gnu--8.4.0 szip/2.1.1--gnu--8.4.0 boost/1.72.0--spectrum_mpi--10.3.1--binary zlib/1.2.11--gnu--8.4.0
elsi/2.5.0--gnu--8.4.0 essl/6.2.1--binary ……..

---------- /cineca/prod/opt/modulefiles/**base**/compilers ----------
cuda/10.1 gnu/8.4.0 pgi/19.10--binary python/3.7.7 python/3.8.2 spectrum_mpi/10.3.1--binary xl/16.1.1--binary

---------- /cineca/prod/opt/modulefiles/**base**/tools ----------
anaconda/2020.02 cmake/3.17.1 singularity/3.5.3 spack/0.14.2-prod superc/2.0

---------- /cineca/prod/opt/modulefiles/**base**/tools ----------
anaconda/2020.02 cmake/3.17.1 singularity/3.5.3 spack/0.14.2-prod superc/2.0

M100 Module Software Env: domains

“Domain” profiles:

```
---------------------------------------------------------- /cineca/prod/opt/modulefiles/profiles ----------------------------------------------------------
profile/advanced  profile/archive  profile/base  profile/candidate  profile/chem-phys  profile/deeplrn  profile/global  profile/lifesc
```

To access a “domain” application, *e.g.* in the chemical physics scientific domain, you need to load the profile/chem-phys first:

```
$ module load profile/chem-phys
```

The profile chem-phys is **added** to the base profile

The domain profiles are all “additive”: you can load them together, adding them to the base profile

M100 Module Software Env: domains

$ module av

---------- /cineca/prod/opt/modulefiles/profiles ----------

profile/advanced profile/archive profile/base profile/candidate profile/chem-phys profile/deeplrn profile/global profile/lifesc

---------- /cineca/prod/opt/modulefiles/**base**/environment----------

autoload

---------- /cineca/prod/opt/modulefiles/**base**/libraries ----------

……..

---------- /cineca/prod/opt/modulefiles/**base**/compilers ----------

……..

---------- /cineca/prod/opt/modulefiles/**base**/tools ----------

……..

---------- /cineca/prod/opt/modulefiles/**chem-phys**/libraries ----------

blas/3.8.0--pgi--19.10--binary libxc/4.3.4--gnu--8.4.0 scalapack/2.1.0--pgi--19.10--binary
elpa/2020.05.001--pgi--19.10--binary netcdf/4.4.1--spectrum_mpi--10.3.1--binary xmlf90/1.5.4--gnu--8.4.0
elpa/2020.05.001--spectrum_mpi--10.3.1--binary netcdff/4.4.4--spectrum_mpi--10.3.1--binary
elsi/2.5.0--gnu--8.4.0 openblas/0.3.9--gnu--8.4.0

---------- /cineca/prod/opt/modulefiles/**chem-phys**/applications ----------

amber/18 cp2k/7.1 cpmd/4.3 gromacs/2020.2 lammps/3mar2020 namd/2.13 plumed/2.7.0 qe-gpu/6.5 siesta/4.1-b4 vasp/5.4.4
vasp/6.1.0 yambo/4.5

M100 Module Software Env:
autoload, modmap

Needing, *e.g.*, lammps?

```
$ module load profile/chem-phys
$ module load autoload lammps/3mar2020
$ module list
Currently Loaded Modulefiles:
 1) profile/base          4) spectrum_mpi/10.3.1--binary   7) cuda/10.1                10) plumed/2.7.0
 2) profile/chem-phys     5) gnu/8.4.0                     8) lapack/3.9.0--gnu--8.4.0 11) openblas/0.3.9--gnu--8.4.0
 3) autoload              6) blas/3.8.0--gnu--8.4.0        9) fftw/3.3.8--gnu--8.4.0   12) lammps/3mar2020
```

The **autoload** module takes care to load all the lammps dependencies

A better, easier way to know if an application is available on M100?

The **modmap** command!

```
$ modmap -m lammps
Profile: advanced
Profile: archive
Profile: base
Profile: chem-phys
                lammps
                 3mar2020
Profile: deeplrn
Profile: lifesc
```

modmap detects all the available profiles, categories, and modules
=> “map” of the available modules

modmap -h # command help

M100 Module Software Env:
spack

$ module load spack/0.14.2-prod

- setup-env.sh file is sourced, $SPACK_ROOT is initialized to /cineca/prod/opt/tools/spack/<vers>/none, spack command is added to your PATH, and some nice command line integration tools too.

- A folder is created into your default $WORK space ($USER/spack-<vers>) with the subfolders created and used by spack during the phase of a package installation:

  - sources cache: $WORK/$USER/spack-<vers>/cache
  - software installation root: $WORK/$USER/spack-<vers>/install
  - module files location: $WORK/$USER/spack-<vers>/modulefiles

- You can define different paths for cache, installation and modules directories (please refer to the spack guide to find out how to customize these paths)

- Some softwares installed with spack are already available as modules or as spack packages:

$ module load spack/0.14.2-prod	or	$ module load spack/0.14.2-prod
$ module av		$ spack find
$ module av <module _name>		$ spack find <package_name>

Can't you find the software you need? Use the "**spack**" environment

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- Software environment
- **Programming environment**
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- Final remarks

M100 Programming Environment

Available compilers in BASE profile:

IBM XL C/C++ and Fortran	16.1.1
gnu	8.4.0
PGI	19.10
Cuda	10.1

IBM XL: enable OpenMP parallelization by the -qsmp option (-qsmp=omp)

Note: newer versions are/will be available in profile/candidate

hwloc provides details about NUMA memory nodes, sockets, shared caches, cores and SMT, etc.

Note: a gnu compiled **Open MPI 4.0.3** installation will be made available in profile/advanced (work in progress)

Available MPI environment in BASE profile:

IBM Spectrum MPI 10.3.1

- Based on Open MPI version 4.0.1, full MPI 3.2 standard
- FCA (hcoll) support (Mellanox Fabric Collective Accelerator on InfiniBand interconnect)
- Relies on hwloc to navigate the server hardware topology
- GPU support
  - NVIDIA GPUDirect RDMA
  - CUDA-aware MPI

Use **mpirun** (not srun) to execute your MPI program

By default, GPU support is disabled. Run the “**mpirun -gpu**” command to enable it.

Use the --report-bindings option for an abbreviated image of the server’s hardware and the binding of processes

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- Software environment
- Programming environment
- **Production environment (SLURM)**
- Considerations and tips on the use of Marconi100
- Final remarks

M100 Production Environment

M100 is a general purpose system used by hundreds of users.

Compute nodes

- Production jobs must be submitted to M100 queueing system: batch jobs
- **SLURM** scheduler and resource manager
- **Node sharing** (but the allocated resources – cores, gpus, memory – are assigned in an exclusive way)

Login nodes

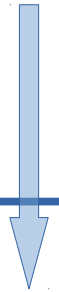
A responsible use of the login nodes is crucial to ensure the effective use of the infrastructure and the access to the computing resources.

- Protect your credentials and access from “safe” posts; opt for **ssh keys with passphrase**
- Interactive runs on login nodes are strongly discouraged and should be limited to **short test runs**
  - **per user limits** on cpu-time (10 minutes) and memory (1 GB)
  - **avoid running large parallel applications** on the front-ends.
- Some apps and tools, in the absence of an explicit setting of the TMPDIR variable, use the **/tmp** area (10 GB) and can saturate it => **critical!**
  Check the apps and tools you use and set the TMPDIR variable to your $CINECA_SCRATCH

M100 Production Environment
SLURM specs and Accounting

Each node “exposes” itself as having

- 128 (virtual) cpus [32 physical cores with 4 HTs each]
- 4 GPUs
- 246000 MB of memory



It is possible to ask **up to**

- 128 ntasks-per-node (1 cpus-per-task)
- 1 ntask-per-node (128 cpus-per-task)
- or any combination of ntasks-per-node * cpus-per-task ≤ 128

The accounting considers:

- the requested number of **physical** cpus
- the requested number of GPUs
- the amount of memory

and calculates the number of equivalent cores taking the maximum among

N physical cpus
N GPUs * 8
Memory / Memory-per-core

BUT:

SLURM has been configured so to assign a physical core with its 4 Hts

Asking for --ntasks-per-node=1 and --cpus-per-task=1 corresponds to ask for --cpus-per-task=4

M100 Production Environment
EuroFusion resources

80 dedicated nodes

production partition: **m100_fua_prod** (68 nodes)

- max 16 nodes per job
- max walltime: 24 h

debug QOS: **m100_qos_fuadbg** (12 nodes)

- max 2 nodes, 256 logical cpus, 8 gpus per job
- max time: 2 h

Serial partition: **m100_all_serial** (2 nodes of the login group, open to the external network)

- max 1 physical core
- max walltime: 4 h
- open and free for all M100 users

lowprio production: **qos_lowprio**

- Non-expired projects with exhausted budget are automatically associated the qos_lowprio
- For active projects: you can ask to be associated to the FUAC4_LOWPRIO Account (write to superc@cineca.it)

Special production: **qos_special**

- For jobs needing more than 16 nodes or 24 h walltime
- Ask superc@cineca.it; your request will be evaluated by the EF Operation Committee

M100 Production Environment

Interactive batch jobs

- In case you need to “interact” with your running job (tuning of input parameters, debugging etc.)
- and it needs more than 10 minutes, or many processes (not suitable on the login nodes)
- **“Interactive” SLURM batch job**

- Ask for the needed resources (cores, gpus, memory, time) with srun or salloc.
- The job is queued and scheduled as any other job but, when, executed, the standard input, output, and error streams are connected to the terminal session from which srun or salloc were launched.
- You can then run your application from that terminal

Non MPI programs (single process or multi-threaded programs using one or more GPUs)

$ **srun** <options> **--pty bash**

The session starts on the compute node (look at the prompt)

MPI programs using one or more GPUs

$ **salloc** <options>

A new session is started on the login node

M100 Production Environment

Interactive batch jobs: non MPI examples

[username@login03 ~]$ srun -N 1 --ntasks-per-node=1 --cpus-per-task=4 --gres=gpu:1 --time=01:00:00 -p <partition_name> -A <account_name> **--pty bash**

[username@r240n17 ~]$ nvidia-smi

=> only one GPU is detected

[username@r240n17 ~]$ numactl -s → show NUMA policy settings of the current process

policy: default
preferred node: current
physcpubind: 0 1 2 3
cpubind: 0
nodebind: 0

Membind: 0 8 252 253 254 255 → all six NUMA nodes CPUs: 0, 8 GPUs: 252-255

[username@r240n17 ~]$ module load cuda/10.1

[username@r240n17 ~]$ matrixMul → one of the CUDA samples (not meant for performance measurements!)

[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla V100-SXM2-16GB" with compute capability 7.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 2719.55 GFlop/s, Time= 0.048 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

M100 Production Environment

Interactive batch jobs: non MPI examples

[username@login03 ~]$ srun -N 1 --ntasks-per-node=1 --cpus-per-task=4--gres=gpu:1 --time=01:00:00 **--mem-bind=local**
-p <partition_name> -A <account_name> --pty bash
[username@r240n17 ~]$ nvidia-smi
=> only one GPU is detected
[username@r240n17 ~]$ numactl -s
policy: default
preferred node: current
physcpubind: 0 1 2 3
cpubind: 0
nodebind: 0
membind: 0 → the task is bound only to the local NUMA node 0
[username@r240n17 ~]$ module load cuda/10.1
[username@r240n17 ~]$ matrixMul
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla V100-SXM2-16GB" with compute capability 7.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 2719.55 GFlop/s, Time= 0.048 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

M100 Production Environment

Interactive batch jobs: non MPI examples

[username@login03 ~]$ srun -N 1 --ntasks-per-node=1 --cpus-per-task=128 –gres=gpu:4 --time=01:00:00 --mem-bind=local -p <partition_name> -A <account_name> --pty bash
[username@r240n17 ~]$ nvidia-smi
=> four GPUs are detected
[username@r240n17 ~]$ numactl -s
policy: default
preferred node: 0
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
cpubind: 0 8
nodebind: 0 8
membind: 0 8
[username@r240n17 ~]$ module load cuda/10.1
[username@r240n17 ~]$ simpleMultiGPU
Starting simpleMultiGPU
CUDA-capable device count: 4
Generating input data…

Computing with 4 GPUs…
  GPU Processing time: 8.011000 (ms)

Computing with Host CPU...

Comparing GPU and Host CPU results...
GPU sum: 16777304.000000
 CPU sum: 16777294.395033
 Relative difference: 5.724980E-07

M100 Production Environment

Interactive batch jobs: MPI examples

The same principles apply for the **request of resources** and **memory binding**, but you need to "**salloc**" your request (the GPUs are non-overallocatable resources, and would get stuck with the *bash* process of srun <options> --pty bash. Each subsequent step corresponding to the mpirun invocation will hang forever)

Hybrid application MPI/OpenMP, 1 node, 4 GPUs, 4 processes (one per GPU) and 8 threads per task => one thread per physical core

```
[username@login03 ~]$ salloc -N 1 --ntasks-per-node=4  --cpus-per-task=32 --gres=gpu:4 --time=01:00:00 --mem-bind=local
-p <partition_name> -A <account_name>
[username@login03 ~]$ export OMP_NUM_THREADS=8
[username@login03 ~]$ mpirun -n 4 --map-by socket:PE=8 --rank-by core <exe>
```

It's easy to forget that you are "inside" a job, and launching a second salloc in the job sessions can lead to a very confusing and not properly working environment.

REMEMBER to close the session (exit or Ctrl-D) at the end of your activity.

If you don't want to exploit the Power9 SMT feature (128 "slots"):

- ntasks * OMP_NUM_THREADS ≤ 32
- choose --cpus-per-task=OMP_NUM_THREADS*4
- define a proper map for the task binding (and ranking)

M100 Production Environment

Process binding and task affinity in jobs

Hybrid application MPI/OpenMP, 1 node, 4 GPUs, 4 processes (one per GPU) and 8 threads per task => one thread per physical core

```
[username@login03 ~]$ salloc -N 1 --ntasks-per-node=4  --cpus-per-task=32 --gres=gpu:4 --time=01:00:00 --mem-bind=local
-p <partition_name> -A <account_name>
[username@login03 ~]$ export OMP_NUM_THREADS=8
[username@login03 ~]$ mpirun -n 4 --map-by socket:PE=8 --rank-by core <exe>
```

Process binding:

OpenMPI option **--map-by <object>:PE=n**

- **Objects**: slot, hwthread, core, L1cache, L2cache, L3cache, socket, numa, etc. (see “man mpirun”). Default: socket
- PE=n : bind n “processing elements” (constituting the object) to each task.

In one M100 socket there are 16 physical cpus (16 PEs)
2 tasks per socket => 8 phys cpus per task

Thread binding:

Note: for XL compilers by default OMP_PROC_BIND=true

The execution environment doesn’t move OpenMP threads between OpenMP places, thread affinity is enabled, and the initial thread is bound to the first place in the OpenMP place list

Node => socket => physcpu => hwthread

M100 Production Environment

Process binding and task affinity in jobs

More on process binding:

- Explicit binding (handy if you have few processes on a single node)

--cpu-set=phys_cpu_id1, phys_cpu_id2,….

- --bind-to option

--bind-to <socket,core,hwthread,etc.>
--bind-to core : 32 MPI processes, 1 to 4 OMP threads

More on thread binding:

Other possible values for OMP_PROC_BIND

OMP_PROC_BIND=master,close,spread

OMP_PLACES=threads,cores,sockets

Explicit placement:

OMP_PLACES={log_cpu_id1},{log_cpu_id2}

Do you have unusual configuration in terms of tasks/threads?

- Experiment with the HybridHello_smpi code (module load superc)
- Ask superc@cineca.it

M100 Production Environment

Non interactive batch jobs

As usual on HPC systems, the large production runs are executed in batch mode. The user writes a list of the needed #SBATCH directives (resources, walltime, mail, jobname, etc. etc.) followed by the needed loading of modules, setting of variables, and launch of the Executable.

The same principles apply concerning memory binding, process and thread affinity, as previously discussed

```
#!/bin/bash
#SBATCH --nodes=16                # Number of nodes
#SBATCH --ntasks-per-node=4       # Number of MPI ranks per node
#SBATCH --ntasks-per-socket=2     # Number of MPI ranks per socket
#SBATCH --cpus-per-task=32        # number of HW threads per task
#SBATCH --gres=gpu:4              # Number of requested gpus per node, can vary between 1 and 4
#SBATCH --mem=230000MB            # Memory per node
#SBATCH --time 01:00:00           # Walltime, format: HH:MM:SS
#SBATCH --mem-bind=local
#SBATCH -A <your account_no>
#SBATCH -p m100_usr_prod

module load profile/chem-phys
module load autoload yambo/4.5

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

mpirun --map-by socket:PE=8 --rank-by core  -np ${SLURM_NTASKS} yambo -F yambo.in -J yambo.out
```

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- Software environment
- Programming environment
- Production environment (SLURM)
- **Considerations and tips on the use of Marconi100**
- Final remarks

General considerations and tips
on the use of M100

Let’s sum up:

- The recommended programming stacks are at present provided by **XL/pgi/gnu with IBM Spectrum MPI**
- To **enable the GPU Support** (CUDA-aware MPI and GPUDirect RDMA) use the mpirun -gpu command
- Rely on the already available software, optimized for M100 architecture, on the modmap tool to map the module environment, and on spack for freely installing the software you need
- The UVAs and ATS provide much help in code development and performance improvement, but bind the task to the local NUMA node with the **--mem-bind=local** to optimize the host memory access and spare GPU Numa nodes for kernels
- Take care of the **binding and affinity of processes and tasks** in order to avoid oversubscription (strong impact on performance)

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- Software environment
- Programming environment
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- **Final remarks**

Final remarks

- Marconi100 is still a very “young” platform, but it already showed impressive capabilities
- Further assessment, configuration tuning, optimization of the interconnect network are required
- Further study and analyses on our side are requested as well to fully exploit M100
- More softwares will become available
- More documentation will become available

Thank you (HLST and all EF users) for your feedback and work on Marconi100, and for your contribution to the pre-production phase!

Outline

- CINECA infrastructure and Marconi100
- System architecture (CPUs, GPUs, Memory, Interconnects)
- Software environment
- Programming environment
- Production environment (SLURM)
- Considerations and tips on the use of Marconi100
- Final remarks
- **Questions and Answers**

Questions and Answers

From Simppa Äkäslompolo

1. *The diagram of the nodes looks like there is no direct (DMA) like access b/w main memory and the GPU. No common bus. Thus, does all data transfer really need to go through the CPU?*

   On each socket the two GPUs are directly connected the one to the other, and both to the socket CPU, via NVLinks (150 GB bidirectional bandwidth peak). Intersocket communications relies on the Xbus, but in any case no copy from the GPU memory to the CPU memory is needed. Each process on the GPUs and the CPUs can access the total memory space (UVA and TSM, see slides n. 17/18; the proper drivers are needed to allow host mpi CUDA awareness and GPUDirect RDMA)

2. *Is there a fusion Domain as a Module Software Environment?*

   Not yet. If Eurofusion users consider it useful to have a “fusion” domain profile it can be easily added, please write to superc@cineca.it for the request (and the list of softwares that you consider proper for the profile)

Questions and Answers

From Francesco Iannone:

1. *Why isn't the PGI/OPENMPI installed ?*

   The OpenMPI software shipped by PGI is not built by us, and it has not been configured to fully integrate itself with slurm and the mellanox acceleration libraries (it is instead CUDA-aware). This can cause (and it did) node failures triggering a kernel panic and simulating a hardware problem. We are investigating with the IBM support regarding the logged hardware problems caused by the PGI-OpenMPI orted process, but we however opted for preventing its use until a proper configuration/compilation will be carried out, and to recommend the use of Spectrum MPI libraries (with the proper PGI wrappers).

Questions and Answers

From Serhiy Mochalskyy

1. *With default MPI pinning (scatter) using 4 mpi tasks and 4 GPUs:*

   task0-socket0; task1-socket1; task2-socket0; task3-socket1.

   I had a problem that task1 assigned to socket1 gets GPU which is connected to socket0 as numbering of GPUs. are GPU0-1 - socket0, GPU2-3 - socket1. The same with task 2 that is by default is assigned to socket0 gets GPU2 from socket1. Is there are any reason to make this scatter pinning as default?

   Usually the set of GPU devices is done in the code with cudaSetDevice (what code are you using?). You can remap the GPUs ID by redefining CUDA_VISIBLE_DEVICES according to the scatter pinning, and then set the device with the ID equal to the MPI process rank. As an alternative, you can override the default OpenMPI behaviour in the process pinning (round-robin w.r.t. sockets; it does not depend by our choice) by adding the –rank-by core option of mpirun.

   Please contact us at superc@cineca.it to tell us what code you are using, and we can run some tests to ensure that desired behaviour is followed.

Questions and Answers

From Tiago Ribeiro:

1. *is Likwid available on M100? I guess this could help on process pinning*

   Likwid is not available on M100 (it implies some security issues which are under consideration on Marconi)

Questions and Answers

From Thomas Hayward-Schneider:

1. *regarding the network topology. I found poor performance on MPI jobs running on more than 1 island. a) Is this expected? b) Is this expected to improve after the "adaptive routing" maintenance?*

   All Eurofusion nodes belong to a single island. The application of slurm topology feature in the choice of the assigned nodes, and the adaptive routing algorithm – both changes were applied at the last maintenance of June 11 – showed a significant improvement in the MPI performances.

2. *What is the recommended (host) BLAS/LAPACK library when using PGI compilers (I think ESSL is only for XL compilers?)?*

   OpenBLAS libraries (available in the base profiles) are recommended in the place of blas/lapack libraries, and in some cases (e.g., quantum-espresso on GPU) they show a better performance even respect to the ESSL. You can however use the ESSL as well with the PGI compilers, but they require runtime XL libraries (hence, you need to load the xl module, and the pgi after that, or to properly set your LD_LIBRARY_PATH adding the path of XL libs).

Questions and Answers

From Thomas Hayward-Schneider:

3. *Is it possible to oversubscribe the batch nodes? It can be useful for job-monitoring. In other words, is there a way to do: srun -n 1 --jobid=12345 --pty bash to get an interactive shell on the node where another allocation is running.*

   Yes, it is possible (did you have issues in doing it?). An alternative way is relying on the pam adopt slurm module, which allows users to ssh to the nodes assigned to the job, with all the processes launched in this session being adopted in the step.extern step

4. *When will the new nvidia HPC SDK be available on m100?*

   We are installing the tool with a try license and the process for acquiring the permanent license is ongoing. We'll inform all users via newsletter, but approximately it should be online in the next few days

Questions and Answers

From Thomas Hayward-Schneider:

5. *Are there special settings required for ROMIO etc for IO with Spectrum MPI on m100 ?*

For parallel I/O, IBM Spectrum MPI supports ROMIO version 3.2.1 and OMPIO, as an unsupported technical preview. To understand how either ROMIO or OMPIO was built, use the ompi_info command, with the highest level of verbosity. To request OMPIO, users must specify -mca io romio321. OMPIO include improved nonblocking MPI-IO implementations.

For improved MPI-IO performance using ROMIO on Spectrum Scale parallel filesystem on POWER9 architecture, users should specify the following hints by creating a my_romio321_hints.txt file with the following content:

```
romio_cb_write enable
romio_ds_write enable
cb_buffer_size 16777216
**cb_nodes **<#nodes>
```

After creating the my_romio321_hints.txt file, users may pass the hints file path to SMPI by using the ROMIO_HINTS environment variable. For example:

ROMIO_HINTS=/path/to/my_romio321_hints.txt

Please refer to the official Spectrum MPI guide the the IBM Knowledge Center for additional hints or limitations.