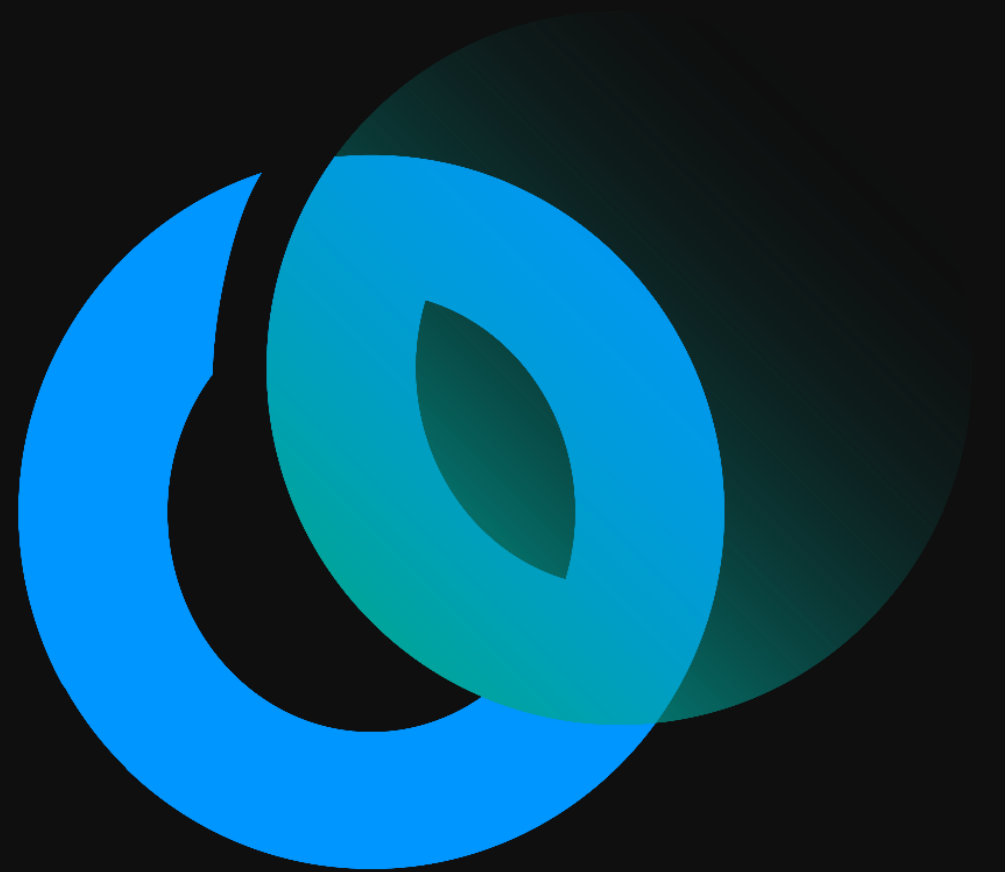


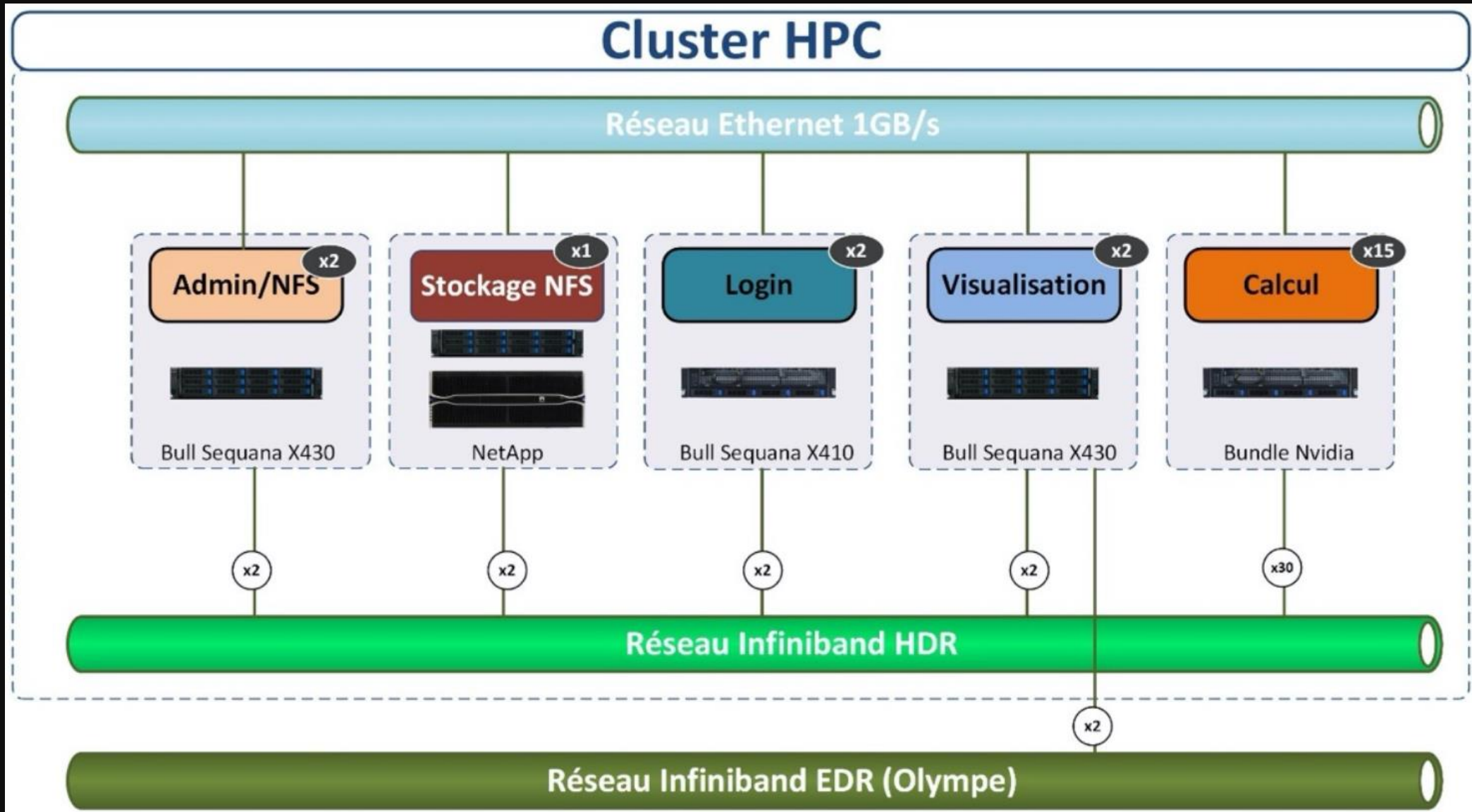
# Turpan Architecture

Okba Hamitou



# Turpan Architecture

## General



# Turpan Architecture

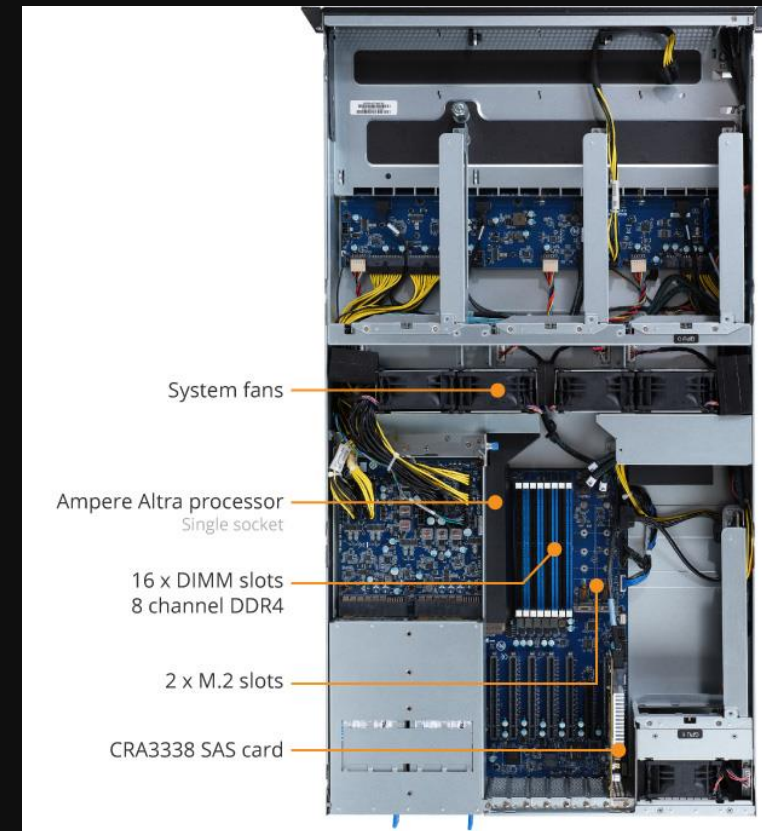
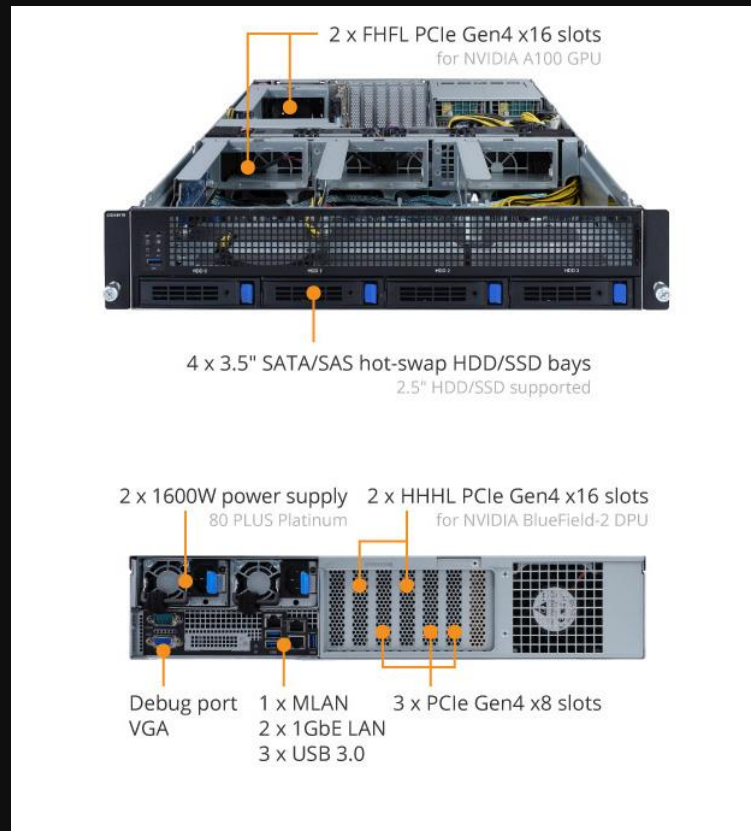
## General

- **Compute nodes**
  - 15 Arm and Nvidia compute nodes, 1200 CPU cores, 30 GPU units
  - 1 Arm Ampere Q80-30 processor, 80 CPU cores, 3,0 GHz, RAM (8 x 64GB) 512GB
  - 2 NVIDIA A100-80GB GPU cards, 6912 CUDA cores, 80GB
- **Login nodes**
  - 2 Arm login nodes
  - 1 Arm Ampere Q80-30 processor, 80 CPU cores, 3,0GHz, RAM (8 x 32GB) 256GB
- **Visualization nodes**
  - 2 Intel x86 Visual Nodes
  - 2 Intel Icelake 6326 processors, 16 cores, 2.8GHz, RAM (16 x 32GB) 512GB
  - 2 NVIDIA A40-48GB GPU cards
- **Storage**
  - 1 NetApp E5760 + DE224C storage, 25TB with SSD, 350TB with HDD
  - 10 x 3,8 TB SSD, 3GB/s write, 8GB/s read
  - 60 x 8 TB HDD, 8GB/s write, 10GB/s read
- **Network**
  - Mellanox Infiniband HDR, 2 x 200 Gb/s

# Turpan Architecture

## Processor : NVIDIA Arm HPC Developer kit

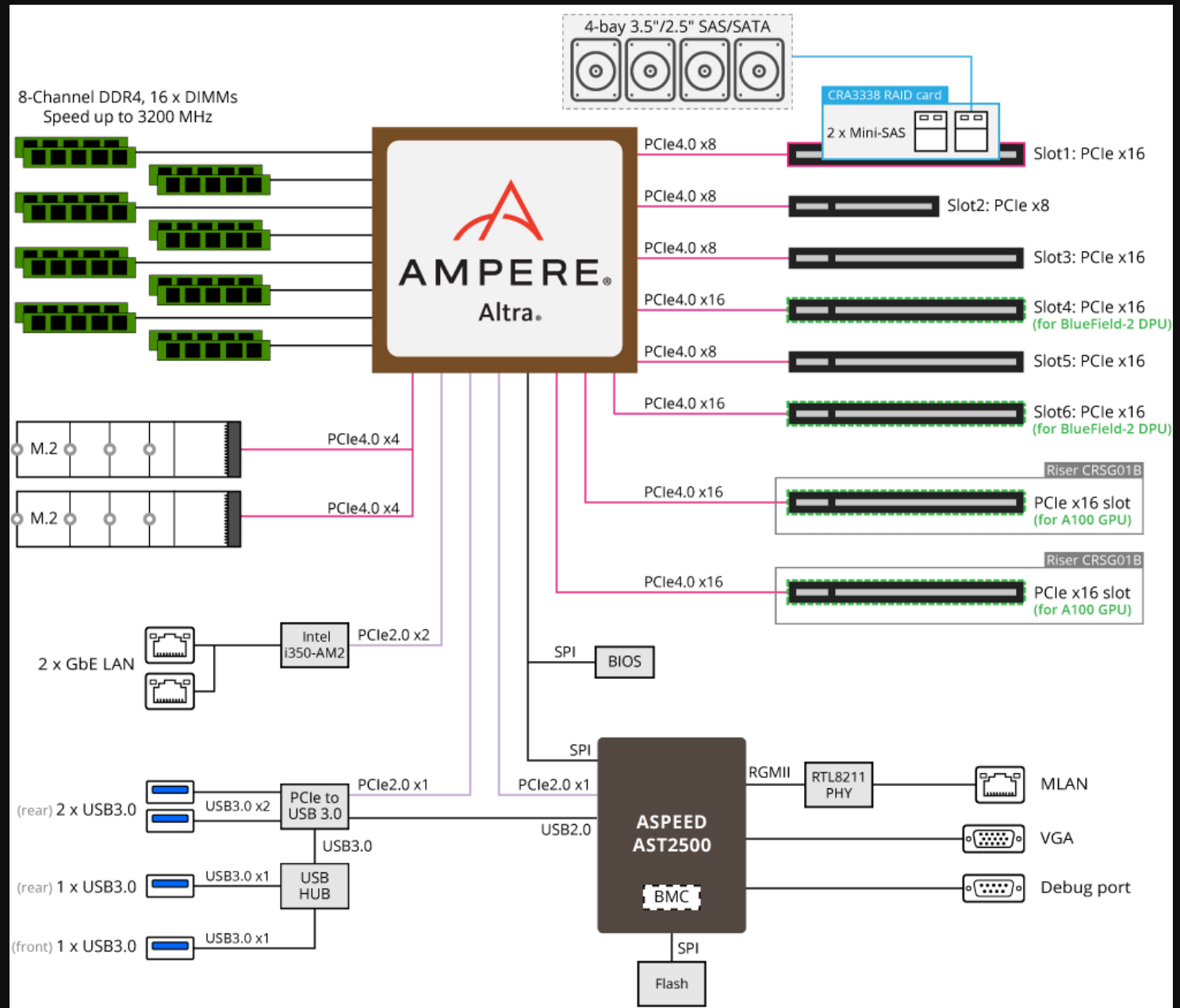
- Development platform for HPC and AI
- Applications porting from x86 to Arm with NVIDIA HPC and AI SDKs
- Enables experimentation and characterization of high-performance, NVIDIA-accelerated, Arm server-based system architectures



# Turpan Architecture

## Processor design

- 1 Arm Ampere Q80-30 80c
- 8 channels DDR4 with 3200 MHz
  - 512 GB (8 x 64 GB)
- 2 Infiniband cards HDR
- 2 NVIDIA A100-80GB GPUs
- 1 local storage
  - HDD or SSD – 6000 GB



# Turpan Architecture

## Core description & features

- The Altra Processor Complex consists of 80 Processor Modules (PMDs)
- Each PMD contains a high-performance Altra core, each of which has
  - Neoverse N1
  - 3,0 GHz
  - Arm v8,2 architecture
  - Private 64 KB L1 I-cache,
  - Private 64 KB L1 D-cache,
  - Private 1 MB L2 cache.
  - Shared 32 MB System Level Cache (SLC) by all 80 PMDs,
- **The Altra Processor Complex features include:**
  - 128-bit double precision FPU
  - 2x full-width (128-bit wide) SIMD execution pipes
  - Coherent Mesh Interconnect (CMI)

# Turpan Architecture

## NVIDIA GPU A100

- Since years ~2000, frequency scaling is over... We are now scaling cores !
- GPU is a massively multi-threaded many-cores architecture
- Thousands of threads executed in parallel
  - Kepler (K80) able to run  $2048 * 13 = 26\ 624$  threads
  - Pascal (P100)  $2048 * 56 = 114\ 688$  threads
  - Volta (V100)  $2048 * 80 = 163\ 840$  threads
  - Ampere (A100)  $2048 * 108 = 221\ 184$  threads



GPU Features	NVIDIA Tesla P100	NVIDIA Tesla V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
GPU Board Form Factor	SXM	SXM2	SXM4
SMs	56	80	108
TPCs	28	40	54
FP32 Cores/ SM	64	64	64
FP32 Cores/ GPU	3584	5120	6912
FP64 Cores/ SM (excl. Tensor)	32	32	32
FP64 Cores/ GPU (excl. Tensor)	1792	2560	3456
INT32 Cores/ SM	NA	64	64
INT32 Cores/ GPU	NA	5120	6912
Tensor Cores/ SM	NA	8	4 <sup>2</sup>
Tensor Cores/ GPU	NA	640	432
GPU Boost Clock	1480 MHz	1530 MHz	1410 MHz

# Turpan Architecture

## Network

- Infiniband HDR based technology
- All equipments (login, visu, storage, service, etc...) are connected by one switch
- Compute nodes are connected to each other using two HDR links
- The topology is equivalent to Fat-Tree without blocking factor



### QM8790

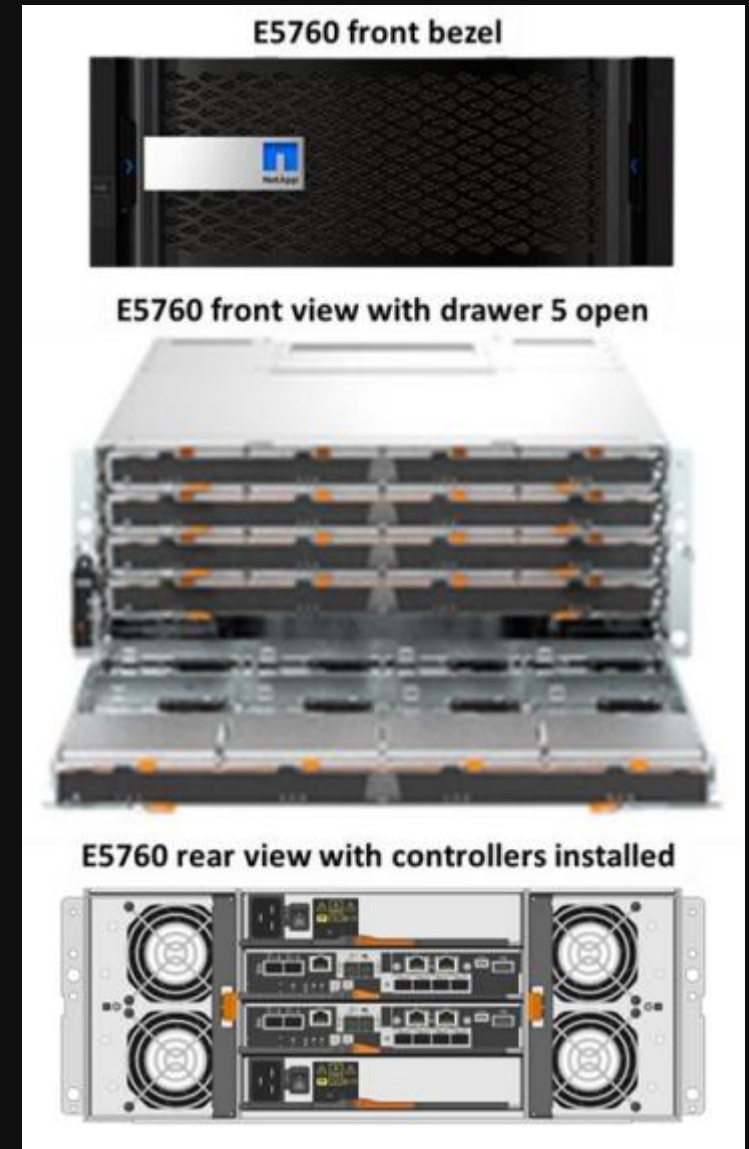
A 1U form factor InfiniBand switch system with 40 HDR 200Gb/s ports; 80 HDR 100Gb/s pots



# Turpan Architecture

## Storage

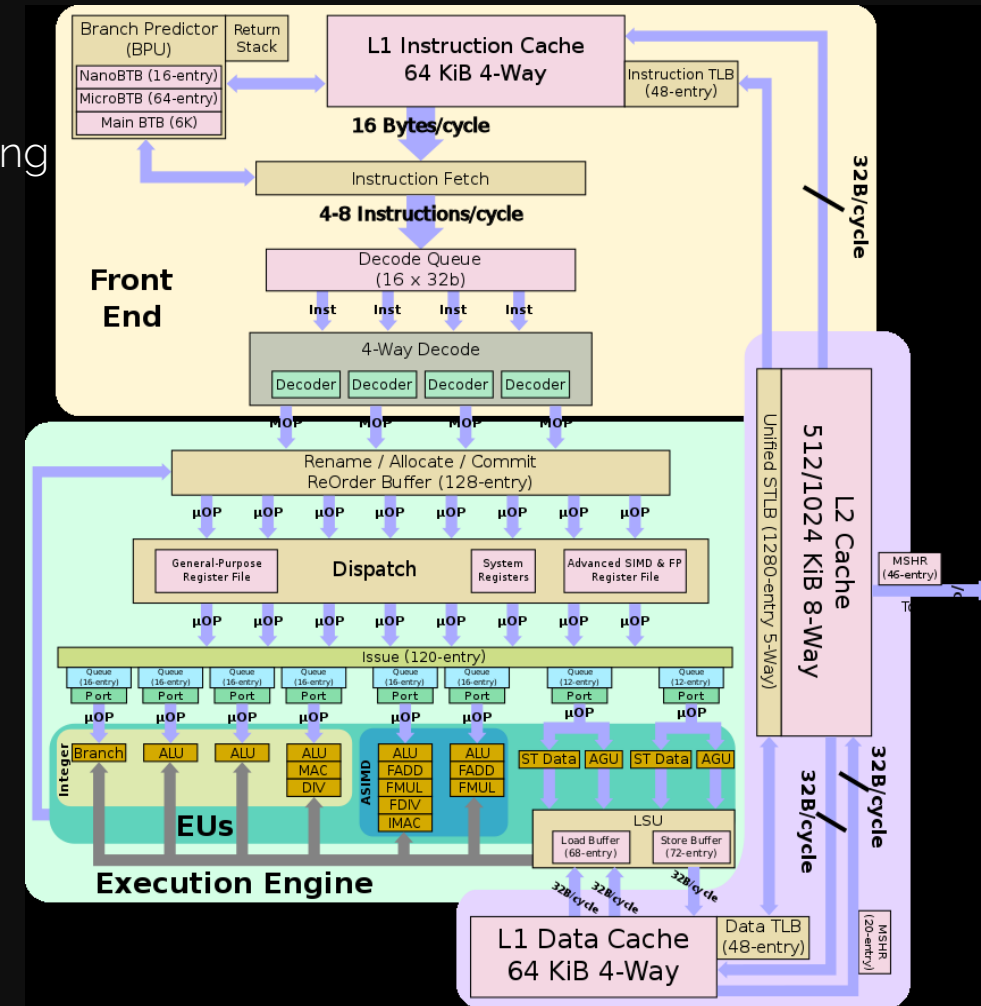
- NFS type storage
  - 25 TB SSD type
  - 350 TB HDD type
- Hybrid storage type which enables fast response time with the SSD
  - SSD disks are used as cache to accelerate I/O operations



# Turpan Architecture

## Core diagram

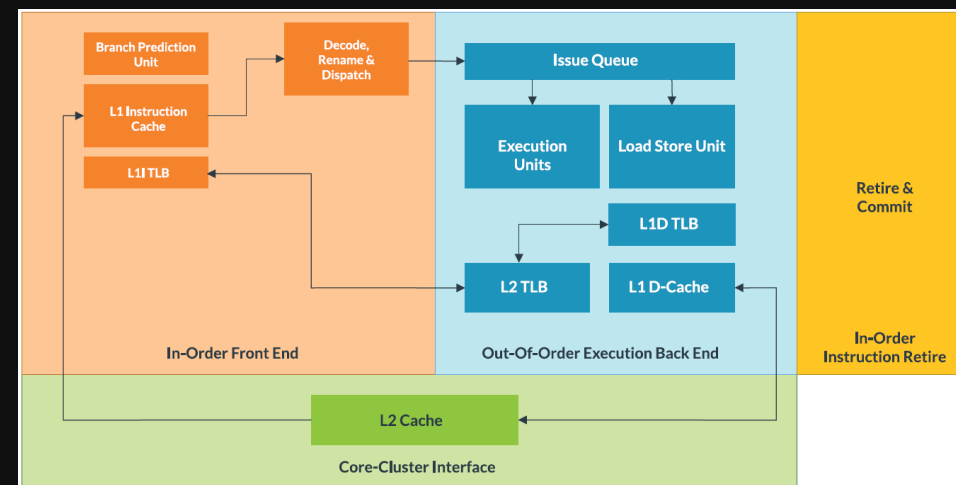
- The execution pipeline includes:
  - Integer execute unit that performs arithmetic and logical data processing operations.
  - Vector execute unit that performs Advanced SIMD and floating-point operations.



# Turpan Architecture

## Core diagram

- The Back End of the CPU handles the execution of the micro-operations which are dispatched to the relevant execution units for processing.
- Neoverse N1 supports multiple execution units including the Branch unit, Load/Store unit and Arithmetic units including the advanced vector engines.
- The number of execution units and cycles taken for the execution of an instruction can vary per micro-architecture; therefore, instruction execution latency and throughput are implementation dependent.
- Once instruction execution is complete, the results are stored and committed in-order when dependencies are resolved.
- Neoverse N1 has
  - 4 integer execution units,
  - 2 Floating point/SIMD pipelines,
  - 2 load/store pipelines, which allows up to 8 micro-operations to be dispatched into the execution pipeline every cycle.



# Turpan Supercomputer Software environment

Okba Hamitou



# Access and data transfers

## How to access, transfert, view data

- You must have a Mesonet account

- Please refer to CALMIP Université de Toulouse for request

- Connect to Turpan

- ssh [your\\_login@turpanlogin\[1,2\].calmip.univ-toulouse.fr](mailto:your_login@turpanlogin[1,2].calmip.univ-toulouse.fr)

- Data transfers (rsync [options] ORIGIN DESTINATION)

- Always from outside the cluster
- Get data from Turpan:

```
rsync -avP -e "ssh -p XXX" user@turpanlogin[1,2].calmip.univ-toulouse.fr:/path/to/data.tgz local_copy.tgz
```

- Get data to Turpan:

```
rsync -avP -e "ssh -p XXX" /path/to/data.tgz user@turpanlogin[1,2].calmip.univ-toulouse.fr:/path/to/directory/  
local_copy.tgz
```

- Data Visualization

- No need to transfer large data to your computer
- Use the visualization node (refer to Turpan webpage for more information about it)

# Disk spaces

home/, work/, tmpdir/

- **Home directory:**
  - Dedicated to the installation of software and binaries,
  - Located in /users/\${GROUPE}/\${USER}" directory
  - Shortcut in environment variable \$HOME
- **Work directory:**
  - Dedicated to inputs and outputs,
  - Located in /work/\${PROJET}/\${USER}" directory
  - Shortcut in environment variable \$WORK
- **Temporary space**
  - Dedicated to temporary files
  - Located in /tmpdir/\${USER}" directory
  - Shortcut in environment variable \$SCRATCHI
  - 200TB shared by all users and accelerated by an 8.5TB SSD cache.
  - Files not accessed for more than 100 days are automatically deleted.
- **Quotas on the different spaces**

Quota	Groupe	User	File group	File user
home		10 Go		100 K
work	1 To		5 M	
tmpdir				5 M

# Partitions

## Full, big, small, shared, visu

- **small:**
  - exclusive,
  - 2 jobs max,
  - no more than 6 nodes per job,
  - max walltime per job 4 hours
- **big:**
  - exclusive,
  - 1 job max,
  - no more than 13 nodes per job,
  - max walltime per job 2 hours
- **full:**
  - exclusive,
  - 1 jobs max,
  - at least 14 nodes per job,
  - max walltime per job 20 hours
- **shared:**
  - non-exclusive,
  - 2 jobs max,
  - no more than 1 GPU, 40 cpu and 256G ram per job,
  - max walltime per job 4 hours
- **visu:**
  - non-exclusive,
  - 1 job max,
  - max 50GB RAM
  - max 8 cpu per job,
  - max walltime per job 4 hours

# Software environment

## Modules

- What is a module ?
  - a convenient way to use software/library/compiler which are available on the cluster

- Usage

- display all available software installed on the cluster

```
hamitouo@turpanlogin1:~> module avail
```

- load/unload a module

```
hamitouo@turpanlogin1:~> module load nvhpc/22.9  
hamitouo@turpanlogin1:~> module unload nvhpc/22.9
```

- switch between 2 modules

```
hamitouo@turpanlogin1:~> module swap nvhpc/22.9 nvhpc/22.11
```

- unload all modules

```
hamitouo@turpanlogin1:~> module purge
```

- list all loaded modules

```
hamitouo@turpanlogin1:~> module list  
Currently Loaded Modulefiles:  
 1) binutils/11.2.0   2) gnu/11.2.0   3) openmpi/gnu/4.1.4-gpu
```

- Strong recommendation to use module show to have details



# Software environment

## NVIDIA compiler collection

- Compilers

- nvc pour C
- nvc++ pour C++
- nvfortran pour Fortran

- Without OpenMPI, with cuda

```
hamitouo@turpanlogin1:~> module load nvidia  
hamitouo@turpanlogin1:~> module load nvhpc-nompi/22.9
```

- With OpenMPI, for CPU computing only

```
hamitouo@turpanlogin1:~> module load openmpi/nvidia/4.1.4-cpu
```

- With OpenMPI for GPU

```
hamitouo@turpanlogin1:~> module load openmpi/nvidia/4.1.4-gpu
```

# Software environment

## Arm compiler collection

- Compilers

- armclang pour C
- armclang++ pour C++
- armflang pour Fortran

- Without OpenMPI

```
hamitouo@turpanlogin1:~> module load arm  
hamitouo@turpanlogin1:~> module load acfl/22.1
```

- With OpenMPI, for CPU computing only

```
hamitouo@turpanlogin1:~> module load openmpi/arm/4.1.4-cpu
```

- With OpenMPI for GPU

```
hamitouo@turpanlogin1:~> module load openmpi/arm/4.1.4-gpu
```

# Software environment

## GNU compiler collection

- Compilers

- gcc pour C
- g++ pour C++
- gfortran pour Fortran

- Without OpenMPI

```
hamitouo@turpanlogin1:~> module load gnu/11.2.0
```

- With OpenMPI, for CPU computing only

```
hamitouo@turpanlogin1:~> module load openmpi/gnu/4.1.4-cpu
```

- With OpenMPI for GPU

```
hamitouo@turpanlogin1:~> module load openmpi/gnu/4.1.4-gpu
```

# Software environment

## Available software

- **Communication libraries :**
  - OpenMPI
  - HPC-X (can be found in nvhpc)
- **Math librairies/application :**
  - Julia
  - Blas, cuBlas, Lapack, Scalapack, cuSparse, AMGX,
  - FFT, CuFFT,
  - PetSC, AMGX,
- **I/O libraries**
  - HDF5
  - NetCDF
  - pNetCDF
- **Scientific applications :**
  - OpenFoam,
  - Magma,
  - CGAL
- **Debug & profile :**
  - *Arm MAP* (CPU / GPU),
  - *NVIDIA Nsight* (GPU)

# Software environment

## Available software

```
----- /usr/local/nvidia/nvhpc/modulefiles -----
nvhpc-byo-compiler/22.9  nvhpc-byo-compiler/22.11  nvhpc-nompi/22.9  nvhpc-nompi/22.11  nvhpc/22.9  nvhpc/22.9-openmpi.4.0.5  nvhpc/22.11
----- /usr/local/arm/modulefiles -----
acfl/22.1  binutils/11.2.0  gnu/11.2.0
----- /usr/local/modules/modulefiles/compilers_and_libraries -----
amgx/nvidia/2.3.0      boost/arm/1.81.0      fftw/arm/3.3.10      gnu/11.2.0      mpfr/arm/4.1.1      openmpi/gnu/4.1.4-cpu      petsc/nvidia/3.18.2
amgx/nvidia/2.3.0-test  boost/gnu/1.81.0      fftw/gnu/3.3.10      magma/gnu/2.7.0      mpfr/gnu/4.1.1      openmpi/gnu/4.1.4-gpu      scalapack/arm/2.2.0
amgx/nvidia/2.4.0      boost/nvidia/1.81.0   fftw/nvidia/3.3.10   magma/nvidia/2.7.0  mpfr/nvidia/4.1.1   openmpi/nvidia/4.1.4-cpu   scalapack/gnu/2.2.0
amgx/nvidia/2.4.0-test  cgal/arm/5.5.1        gmp/arm/6.2.1        mpc/arm/1.3.1      nvidia             openmpi/nvidia/4.1.4-gpu   scalapack/nvidia/2.2.0
arm                    cgal/gnu/5.5.1        gmp/gnu/6.2.1        mpc/gnu/1.3.1      openmpi/arm/4.1.4-cpu  petsc/arm/3.18.2
binutils/11.2.0        cgal/nvidia/5.5.1    gmp/nvidia/6.2.1    mpc/nvidia/1.3.1   openmpi/arm/4.1.4-gpu  petsc/gnu/3.18.2
----- /usr/local/modules/modulefiles/scientific_applications -----
 hdf5/arm/1.12.2-parallel  hdf5/gnu/1.12.2-seq      julia/gnu/1.8.5  netcdf/nvidia/4.8.1  openfoam/gnu/10      openfoam/nvidia/v2212  pnetcdf/nvidia/1.12.3
 hdf5/arm/1.12.2-seq      hdf5/nvidia/1.12.2-parallel  netcdf/arm/4.8.1  openfoam/arm/10      openfoam/gnu/v2212  pnetcdf/arm/1.12.3
 hdf5/gnu/1.12.2-parallel  hdf5/nvidia/1.12.2-seq    netcdf/gnu/4.8.1  openfoam/arm/v2212  openfoam/nvidia/10  pnetcdf/gnu/1.12.3
----- /usr/local/modules/modulefiles/tools -----
arm-forge/22.1.2  cmake/3.25.1  dcgm/3.1.3-1
```

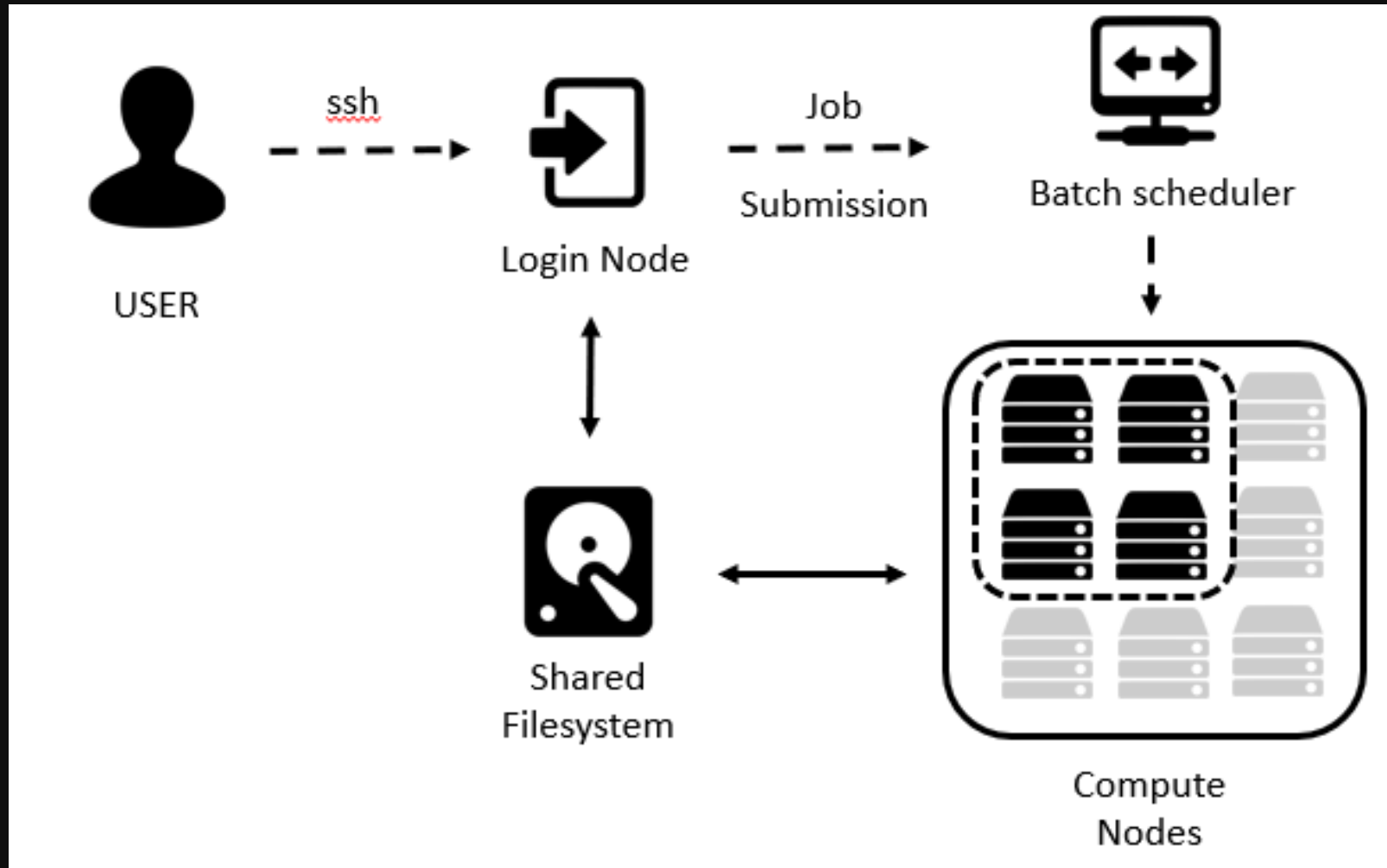
# Turpan Supercomputer Fast Start

Okba Hamitou



# Slurm

## Batch system



# Slurm

## In a nutshell

- The Simple Linux Utility for Resource Management (SLURM) is used for submission, scheduling, execution, and monitoring of jobs
- Main Slurm commands :
  - *sinfo* show information about all partitions and nodes managed by SLURM as well as about general system state
  - *squeue* query the list of pending and running jobs
  - *sbatch* submit a batch script
  - *scancel* cancel a pending or running job or job step
  - *salloc* request interactive jobs/allocations
  - *srun* initiate parallel job steps within a job or start an interactive job
  - *scontrol* provides some functionality for the users to manage jobs or get some information about the system configuration such as nodes, partitions, jobs, and configurations
  - *sacct* retrieve accounting information about jobs and job steps. For completed jobs *sacct* queries the accounting database



# Slurm

## Batch system

Advantages of a batch system are:

- It allows to run MANY jobs at the same time.

The system takes care that they are run efficiently on the available resources.

- Multiusers, queue system.

A batch system allows users to always submit jobs, even if a lot of people are using the system at the same time. In addition, take care of budgeting and fair resource usage.

- System load balance.

The system takes care of balancing the load across nodes and during time. In a batch system, most jobs may be submitted during office hours, but the scheduler will continue to start jobs at night as nodes become available.

# Slurm

## Partitions

- There are 5 partitions :
  - Small (default)
  - Big
  - Full
  - Shared
  - Visu
- A partition request is done using the flag
  - -p
  - --partition=

# Slurm

## Resource allocation

- Every job submission starts with a resource allocation (nodes, cores, memory).
- An allocation is valid for a specific amount of time, and can be created using the salloc, sbatch or srun commands.
- salloc and sbatch only create resource allocations, srun launches parallel tasks within such a resource allocation, or implicitly creates an allocation if not started within one.
- The usual procedure is to combine resource requests and task execution (job steps) in a single batch script (job script) and then submit the script using the sbatch command.

# Slurm

## sbatch

### Syntax

- `sbatch [options] script [args...]`

### Job Script

- Usually, a job script consists of two parts. The first part is optional but highly recommended:
  - Slurm-specific options used by the scheduler to manage the resources (e.g. memory) and configure the job environment
  - Job-specific shell commands

The job script acts as a wrapper for your actual job. Command-line options can still be used to overwrite embedded options.

I strongly recommend to get familiar with sbatch options on Slurm website or using the man command.

# Slurm

## Submitting jobs

- Allocating Resources with SLURM : the *salloc* and *sbatch* commands make resource allocations only. The *srun/mpirun* command launches parallel tasks.
- The usual way to allocate resources and execute a job is to write a batch script and submit them to SLURM with the *sbatch* command.

```
#!/bin/bash
#SBATCH --job-name=test_mpi
#SBATCH --partition=compute
#SBATCH --ntasks=96
#SBATCH --time=01:30:00
#SBATCH --output=job_%x_%j.out
#SBATCH --err=job_%x_%j.err
#SBATCH --exclusive

# setup environment
module load applications

#launch executable with or without inputs
srun ./mybinary input1 input2
```

Specify the name of the partition

Specify a reasonable time limit. It may allow your job to start quickly

Specify the names of your output files.  
%x : environment variable referencing the job name  
%j : environment variable referencing the job ID

# Slurm

## Submitting jobs

- Running serial, OpenMP, MPI jobs

Sequential job	OMP job	MPI job
<pre>#!/bin/bash #SBATCH --job-name=test_serial #SBATCH --partition=compute #SBATCH --ntasks=1 #SBATCH --time=00:30:00 #SBATCH --output=job_%x_%j.out #SBATCH --err=job_%x_%j.err #SBATCH --exclusive  # setup environment module load app1 app2 ..appN  #launch executable with or without inputs ./mybinary input1 input2</pre>	<pre>#!/bin/bash #SBATCH --job-name=test_omp #SBATCH --partition=compute #SBATCH --ntasks=1 #SBATCH --cpus-per-task=28 #SBATCH --time=00:30:00 #SBATCH --output=job_%x_%j.out #SBATCH --err=job_%x_%j.err #SBATCH --exclusive  # setup environment module load app1 app2 ..appN export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK  #launch executable with or without inputs ./mybinary input1 input2</pre>	<pre>#!/bin/bash #SBATCH --job-name=test_mpi #SBATCH --partition=compute #SBATCH --ntasks=240 # or #SBATCH --nodes=3 #SBATCH --time=00:30:00 #SBATCH --output=job_%x_%j.out #SBATCH --err=job_%x_%j.err #SBATCH --exclusive  # setup environment module load app1 app2 ..appN  #launch executable with or without inputs srun ./mybinary input1 input2</pre>

# Slurm

## Submitting jobs

- Running hybrid jobs (OpenMP + MPI)

### Hybrid job

```
#!/bin/bash
#SBATCH --job-name=test_hybride
#SBATCH --partition=compute
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=40
#SBATCH --tasks-per-node=2
#SBATCH --gres=gpu:2
#SBATCH --time=00:30:00
#SBATCH --output=job_%x_%j.out
#SBATCH --err=job_%x_%j.err
#SBATCH --exclusive

# setup environment
module load app1 app2 ..appN
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

#launch executable with or without inputs
srun ./mybinary input1 input2
```

# Slurm

## Submitting jobs

- Environment Variables

- Slurm sets various environment variables available in the context of the job script. Some are set based on the requested resources for the job.
- For the full list, see man sbatch

Environment variables	Correspondence
SLURM_JOB_ID	Job id
SLURM_NTASKS	#SBATCH -n
SLURM_JOB_NUM_NODES	#SBATCH -N
SLURM_CPUS_PER_TASK	#SBATCH -c
SLURM_ACCOUNT	#SBATCH -A
SLURM_JOB_NODLIST	Nodelist which is allocated

```
#!/bin/bash
#SBATCH -A <account name>
#SBATCH -J test
#SBATCH -N 2
#SBATCH -n 32
#SBATCH -t 00:10:00
#SBATCH -o test.%j.out
#SBATCH -e test.%j.err
NP=$SLURM_NTASKS
echo "jobid:$SLURM_JOB_ID"
echo "numnodes:$SLURM_JOB_NUM_NODES"
echo "nodelist:$SLURM_JOB_NODLIST"
mpirun -np $NP ./a.out
```



# Slurm

## Monitoring jobs

- Different Slurm commands provide information about jobs/job steps on different levels.
- The command `squeue` provides high-level information about jobs in the Slurm scheduling queue (state information, allocated resources, runtime, ...).
- The command `sstat` provides detailed usage information about running jobs, and `sacct` provides accounting information about active and completed (past) jobs.
- The command `scontrol` provides even more detailed information about jobs and job steps.

# Slurm

## Monitoring jobs

- `squeue`
  - Use the `squeue` command to get a high-level overview of all active (running and pending) jobs in the cluster.
- The default output format is as follows:
  - JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
- Job States
  - During its lifetime, a job passes through several states.
  - The most common states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED.
  - PD Pending. Job is waiting for resource allocation
  - R Running. Job has an allocation and is running
  - S Suspended. Execution has been suspended and resources have been released for other jobs
  - CA Cancelled. Job was explicitly cancelled by the user or the system administrator
  - CG Completing. Job is in the process of completing. Some processes on some nodes may still be active
  - CD Completed. Job has terminated all processes on all nodes with an exit code of zero
  - F Failed. Job has terminated with non-zero exit code or other failure condition

# Slurm

## Monitoring jobs

- List all currently running jobs of user foo:
  - `squeue --user=foo --states=PD,R`
- List all currently running jobs of user foo in partition bar:
  - `squeue --user=foo --partition=bar --states=R`
- **scontrol**
  - Use the `scontrol` command to show more detailed information about a job
  - Syntax : `scontrol [options] [command]`
- **Examples**
  - Show detailed information about job with ID 354684: `scontrol show jobid 354684`
  - Show even more detailed information about job with ID 35464 (including the jobscript): `scontrol -dd show jobid 3546`
- **sacct**
  - Use the `sacct` command to query information about past jobs
  - Syntax: `sacct [options]`
  - See `man sacct`

# Slurm

## Deleting jobs

- **scancel**
  - Use the scancel command to delete active jobs
- **Syntax** : `scancel [options] <jobid> ...`
- **scancel can be restricted to a subset of jobs, using the following options with the related value, e.g:**
  - `-u, --user $USER` jobs of current user
  - `-A, --account` jobs under this charge account
  - `-n, --jobname` jobs with this job name
  - `-p, --partition` jobs in this partition
  - `-t, --state` jobs in this state
- **Delete specific job:** `scancel 12345678`
- **Delete all running jobs:** `scancel --state=R`
- **Delete all of your jobs:** `scancel --user $USER`

# Turpan Supercomputer Fast Start

Okba Hamitou



# Compiling on Arm Ampere Altra

## Compilers and Build Systems

- All popular build systems are supported and performant on Arm
- GCC, Arm, NVIDIA are excellent Arm compilers
  - Auto-vectorizing, auto-parallelizing, tested, in production
  - Arm & partners are the majority of GCC contributors
- All major build systems and tools work on Arm
  - CMake, Make, GNUMake, EasyBuild, Spack etc...

# Compiling on Arm Ampere Altra Neon for Armv8-A

- Arm Neon technology implements SIMD
- Enabled through :
  - Arm Performance Libraries, FFTW
  - Auto-vectorization features of compilers
  - *Neon intrinsic functions*
  - *Hand-coded Neon assembler (for experienced programmers)*

# Compiling on Arm Ampere Altra

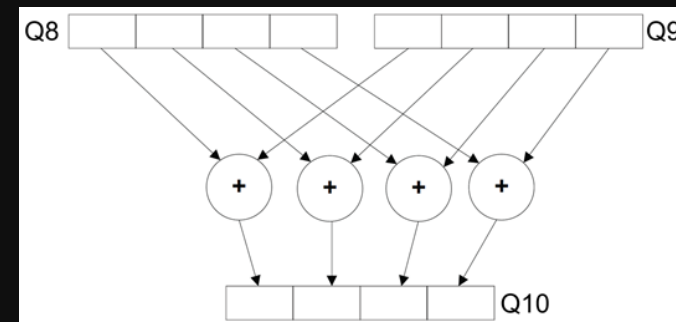
## Neon for Armv8-A

- Single Instruction Single Data (SISD)
  - Each instruction performs its specified operation on a single piece of data
  - Performing four addition operations requires four instructions to add values from four pairs of registers

```
ADD x0, x0, x5
ADD x1, x1, x6
ADD x2, x2, x7
ADD x3, x3, x8
```

- Single Instruction Multiple Data
  - The same operation is executed simultaneously for multiple items
  - The following instruction adds four pairs of single-precision (32-bit) values together
  - This operation adds two 128-bit (quadword) registers, Q8 and Q9, and stores the result in Q10. Each of the four 32-bit lanes in each register is added separately

```
ADD Q10.4S, Q8.4S, Q9.4S
```

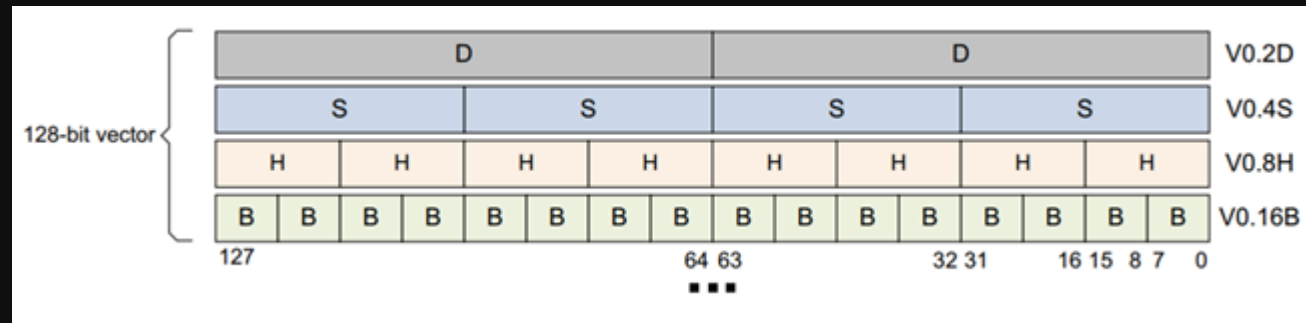




# Compiling on Arm Ampere Altra

## Neon for Armv8-A

- A 128-bit Neon vector can contain the following element sizes:
  - Sixteen 8-bit elements (operand suffix `.16B`, where B indicates byte)
  - Eight 16-bit elements (operand suffix `.8H`, where H indicates halfword)
  - Four 32-bit elements (operand suffix `.4S`, where S indicates word)
  - Two 64-bit elements (operand suffix `.2D`, where D indicates doubleword)

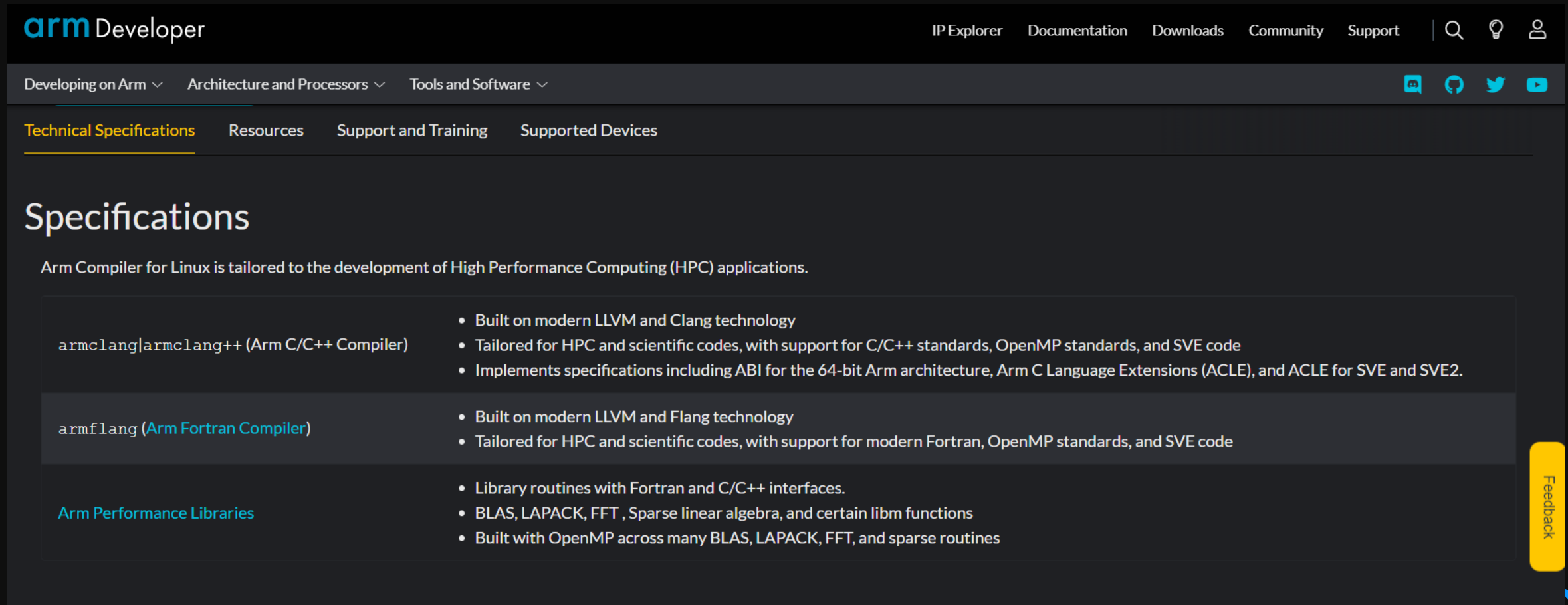


# Compiling on Arm Ampere Altra

Arm compiler & Arm performance libraries Now free to use – No license required

- Free to use, no licence required

<https://developer.arm.com/Tools%20and%20Software/Arm%20Compiler%20for%20Linux>



The screenshot shows the Arm Developer website interface. At the top, the 'arm Developer' logo is on the left, and navigation links for 'IP Explorer', 'Documentation', 'Downloads', 'Community', and 'Support' are on the right. Below the logo, there are dropdown menus for 'Developing on Arm', 'Architecture and Processors', and 'Tools and Software'. A secondary navigation bar includes 'Technical Specifications' (highlighted), 'Resources', 'Support and Training', and 'Supported Devices'. The main content area is titled 'Specifications' and contains a paragraph: 'Arm Compiler for Linux is tailored to the development of High Performance Computing (HPC) applications.' Below this, there are three sections, each with a list of features:

- armclang|armclang++ (Arm C/C++ Compiler)**
  - Built on modern LLVM and Clang technology
  - Tailored for HPC and scientific codes, with support for C/C++ standards, OpenMP standards, and SVE code
  - Implements specifications including ABI for the 64-bit Arm architecture, Arm C Language Extensions (ACLE), and ACLE for SVE and SVE2.
- armflang (Arm Fortran Compiler)**
  - Built on modern LLVM and Flang technology
  - Tailored for HPC and scientific codes, with support for modern Fortran, OpenMP standards, and SVE code
- Arm Performance Libraries**
  - Library routines with Fortran and C/C++ interfaces.
  - BLAS, LAPACK, FFT, Sparse linear algebra, and certain libm functions
  - Built with OpenMP across many BLAS, LAPACK, FFT, and sparse routines

A yellow 'Feedback' button is visible on the right side of the page.

# Compiling on Arm Ampere Altra

## Arm compiler with Neon for Armv8-A

- **Optimization options**

- `-Ox` with `x` in `{0, 1, 2, 3, fast}`
- Auto-vectorization is enabled by default at optimization level `-O2` and higher. The `-fno-vectorize` option lets you disable auto-vectorization.
- At optimization level `-O1`, auto-vectorization is disabled by default. The `-fvectorize` option lets you enable auto-vectorization.
- At optimization level `-O0`, auto-vectorization is always disabled. If you specify the `-fvectorize` option, the compiler ignores it.
- Specifying the `-Rpass=loop` and `-Rpass-analysis=loop` compiler options displays useful diagnostic and analysis information.

- **Specify Neon target**

- `-mcpu=native` option is a combination of `-march` and `-mtune`. It simultaneously specifies the target architecture and optimizes for a given microarchitecture.

- More information here [Porting and Optimizing HPC Applications for Arm Documentation](#)

# Compiling on Arm Ampere Altra

## NVHPC compiler with Neon for Armv8-A

- Optimization options

- `-Ox` with `x` in `{0, 1, 2, 3, 4}` and `-fast`
- SIMD code generation is enabled by default at optimization level `-O2` and higher.
- At optimization level `-O1`, local optimizations are performed (register allocations)
- At optimization level `-O0`, no optimizations are performed.
- Specifying the `-Minfo` compiler options displays useful diagnostic and analysis information.

- Specify Neon target

- `-tp=native|neoverse-n1` is used to specify a CPU target.

- More information here : [NVIDIA HPC SDK Version 22.11 Documentation](#)

# Compiling on Arm Ampere Altra

## GNU compiler with Neon for Armv8-A

- Optimization options

- -Ox with x in {0, 1, 2, 3, fast}
- SIMD code generation is enabled by default at optimization level -O2 and higher.
- At optimization level -O1, reduce code size and execution time.
- At optimization level -O0, no optimizations are performed. Reduce compilation time and make debugging produce the expected results. This is the default.
- -fopt-info-optimized -fopt-info-missed -fopt-info-all -fopt-info-vec-optimized -fopt-info-vec-missed to print information about performed or missed optimizations.

- Specify Neon target

- -mcpu=native|neoverse-n1 to target Arm Ampere Altra.

- More information here [Optimize Options \(Using the GNU Compiler Collection \(GCC\)\)](#)

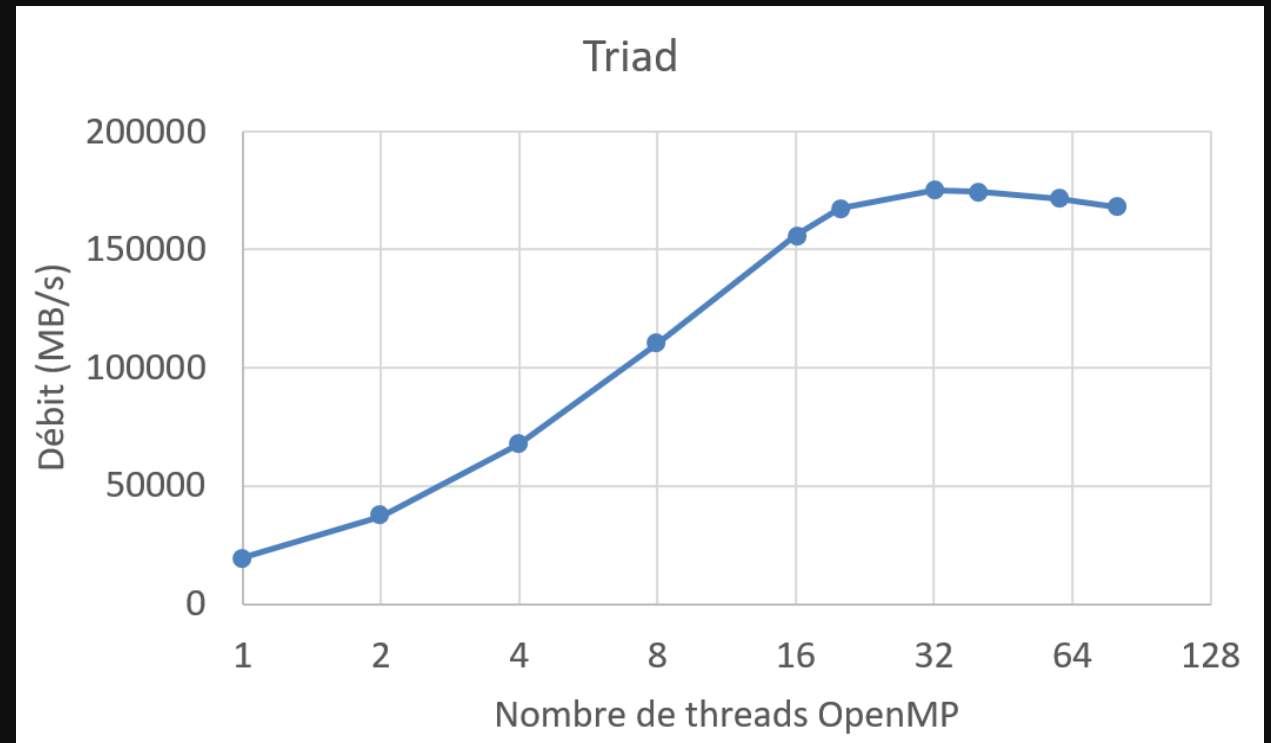
# Turpan Supercomputer Fast Start

Okba Hamitou



# Stream

- `armclang -Ofast -mcpu=native -mcmmodel=large -fopenmp -ffast-math -ffp-contract=on -DSTREAM_ARRAY_SIZE=1000000000 -DNTIMES=20 stream.c -o stream.armclang`
- Theoretical peak performance : 204 GB/s
- Measured peak performance : 175 GB/s



# HPL & HPCG

## With Arm Performance Libraries

- HPL

- CCFLAGS = \$(HPL\_DEFS) -Ofast -mcpu=neoverse-n1
- Theoretical peak performance : 1,92 TFlops
- Measured peak performance : 1,18 TFlops (61,4% efficiency)

- HPCG

- CXXFLAGS = \$(HPCG\_DEFS) -O3 -mcpu=native -ffp-contract=fast -fvectorize -funroll-loops -std=c++11 -ffast-math
- Measured peak performance : 24,22 GFlops

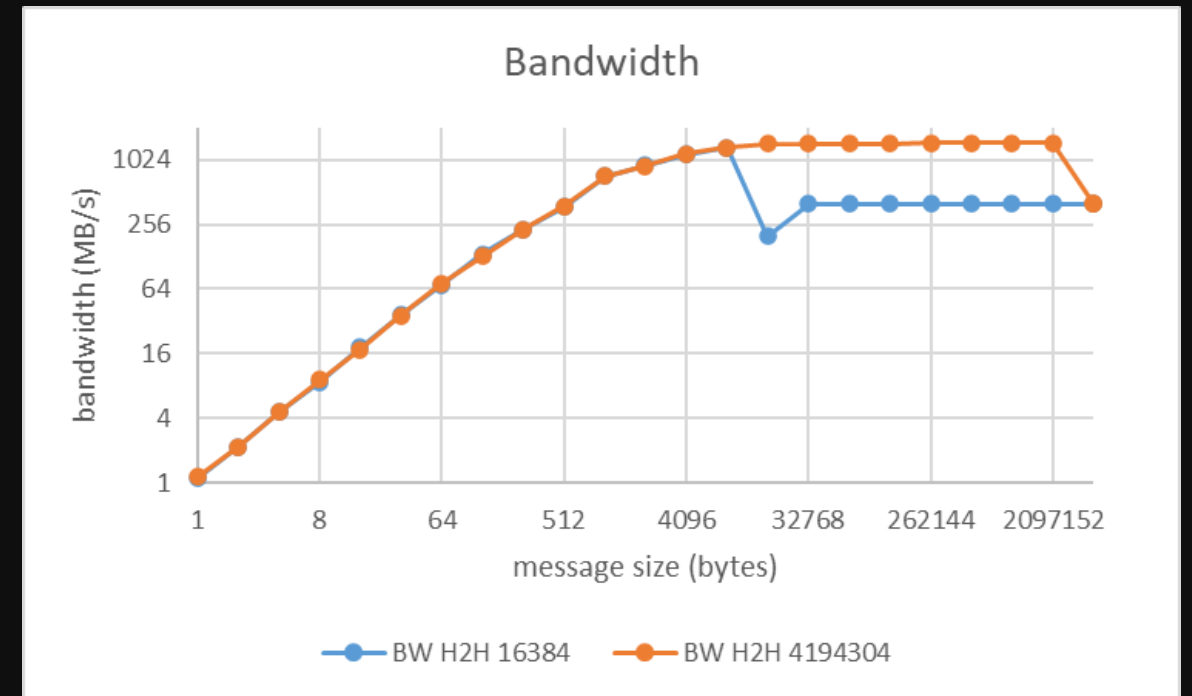
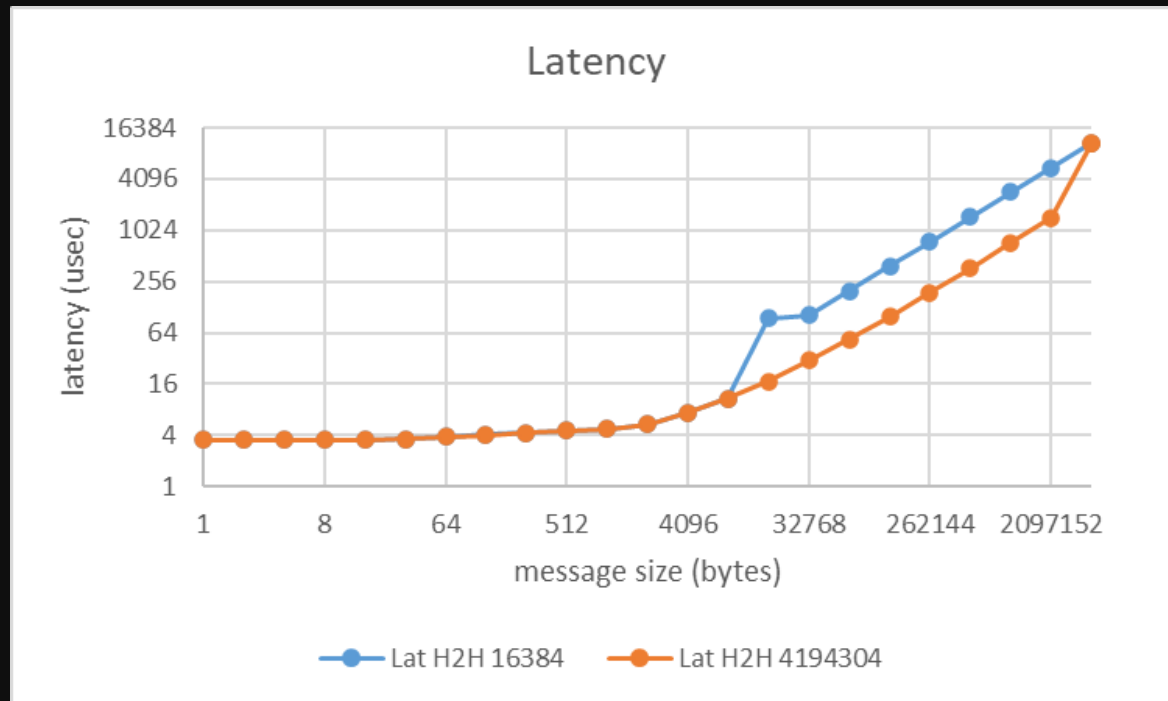


# OSU bandwidth & latency

- Compilation with GNU (CPU) and NVCC (GPU)
- Libraries : OpenMPI and UCX with CUDA and GDRCopy support
- Important environment variables :
  - `export UCX_TLS=cma,rc,mm,cuda_copy,cuda_ipc,gdr_copy`
  - `export UCX_RNDV_THRESH=16384`
  - `export UCX_RNDV_SCHEME=put_zcopy`
  - `export UCX_IB_GPU_DIRECT_RDMA=yes`
  - `export UCX_NET_DEVICES=all`
- Important environment variables :
  - `export UCX_RNDV_THRESH=16384`
  - This option set the message size rendezvous threshold (the UCX will switch from eager algorithm to rendezvous).

# OSU bandwidth & latency

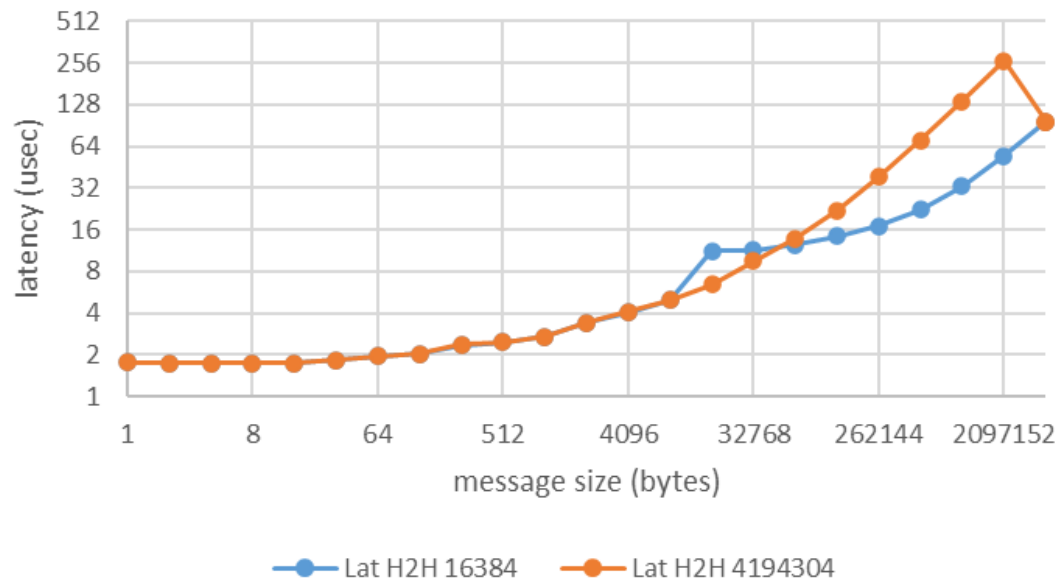
- Device to Device latency and bandwidth
- Influence of the UCX\_RNDV\_THRESH variable
  - UCX\_RNDV\_THRESH = 16 KB
  - UCX\_RNDV\_THRESH = 4 MB



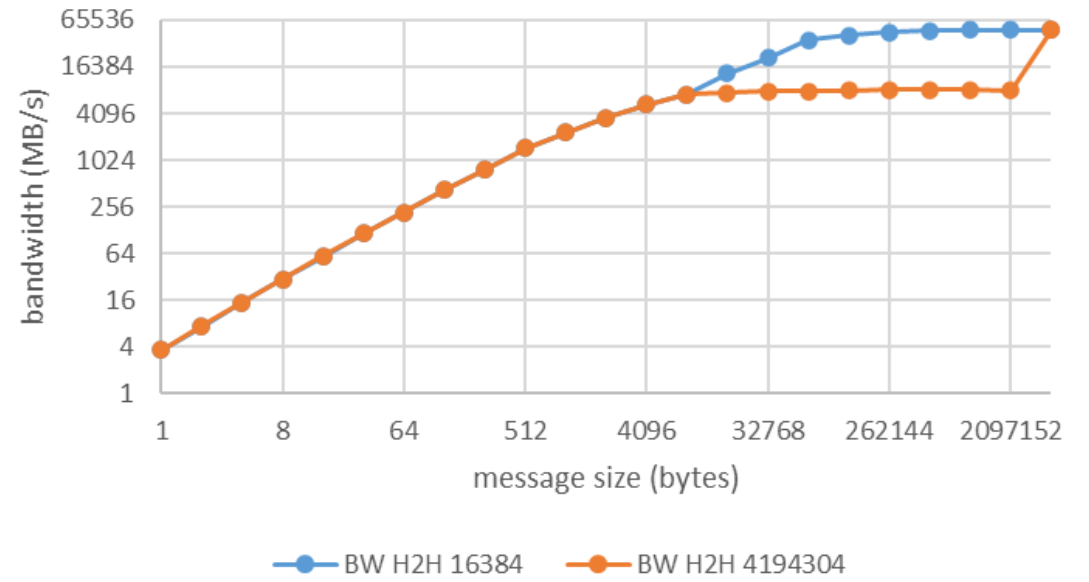
# OSU bandwidth & latency

- Host to Host latency and bandwidth
- Influence of the UCX\_RNDV\_THRESH variable
  - UCX\_RNDV\_THRESH = 16 KB
  - UCX\_RNDV\_THRESH = 4 MB

### Latency

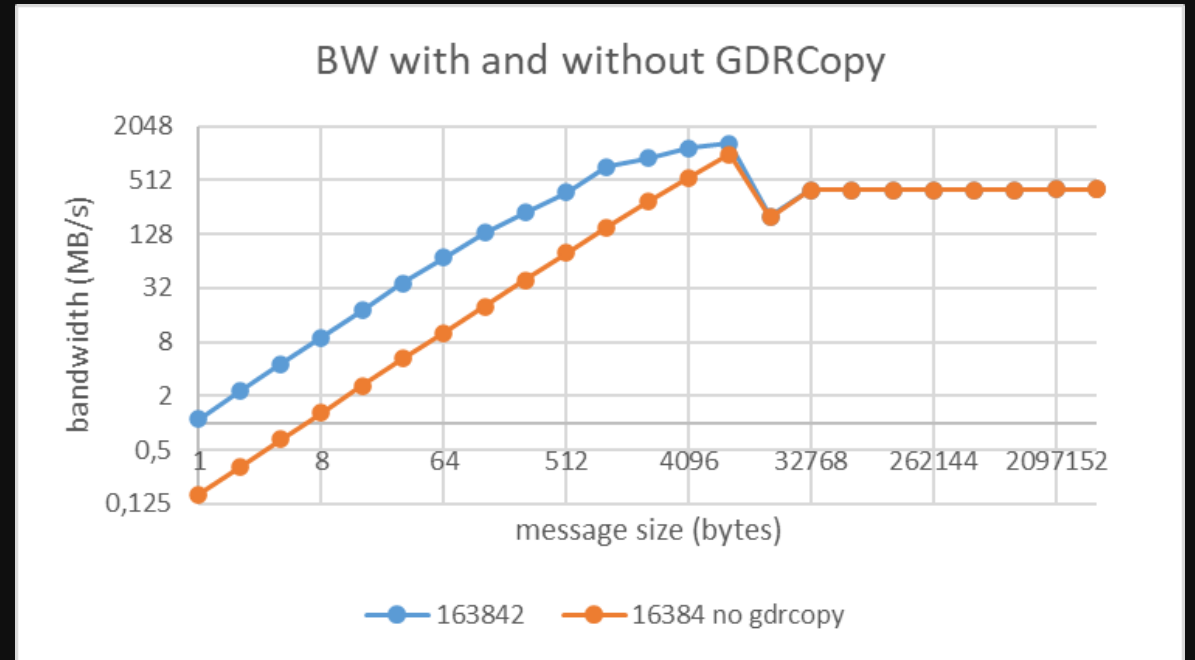
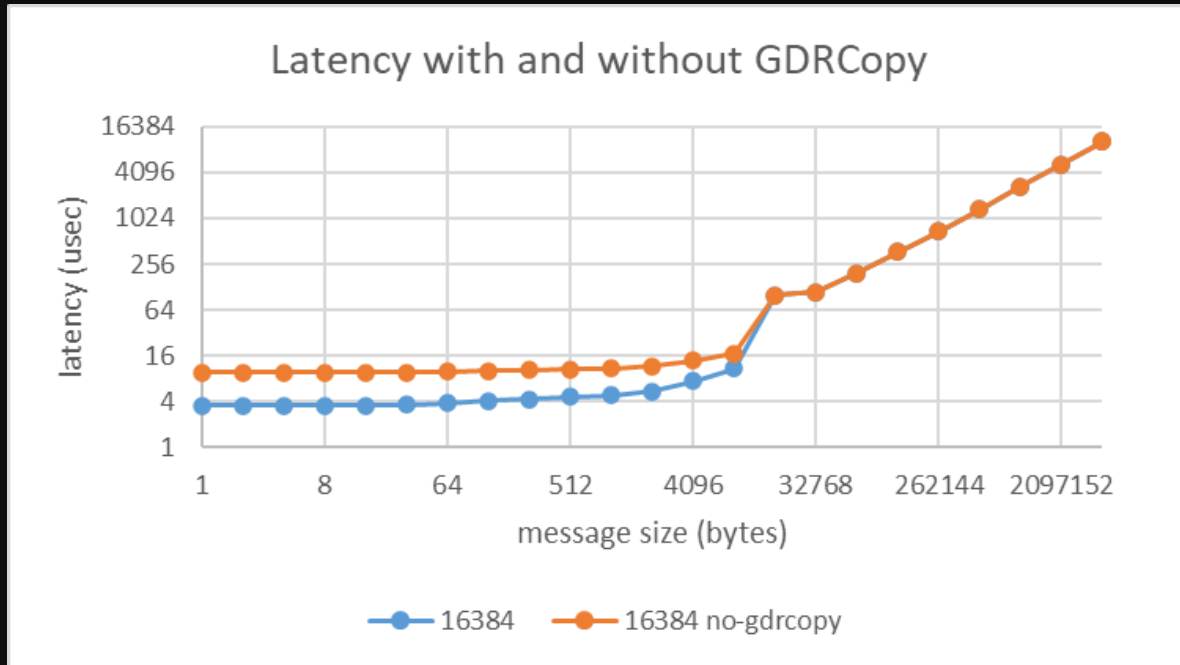


### Bandwidth



# OSU bandwidth & latency

- Device to Device latency and bandwidth
- Influence of GDRCopy
  - Bandwidth x7
  - Latency x3



[Okba-nafie.hamitou@atos.net](mailto:Okba-nafie.hamitou@atos.net)

Atos is a registered trademark of Atos SE. February 2022. © 2022 Atos. Confidential information owned by Atos, to be used by the recipient only. This document, or any part of it, may not be reproduced, copied, circulated and/or distributed nor quoted without prior written approval from Atos.

