
# Regular access call PLG/2022/03 user training

# LUMI, the Queen of the North

*LUMI is a Tier-0* ***GPU-accelerated supercomputer*** *that enables the convergence of* ***high-performance computing, artificial intelligence,*** *and* ***high-performance data analytics.***

Tier-0 GPU partition:

over **550** Pflop/s
powered by AMD Instinct GPUs

Supplementary CPU partition

**~200,000**
AMD EPYC CPU cores

Interactive partition with

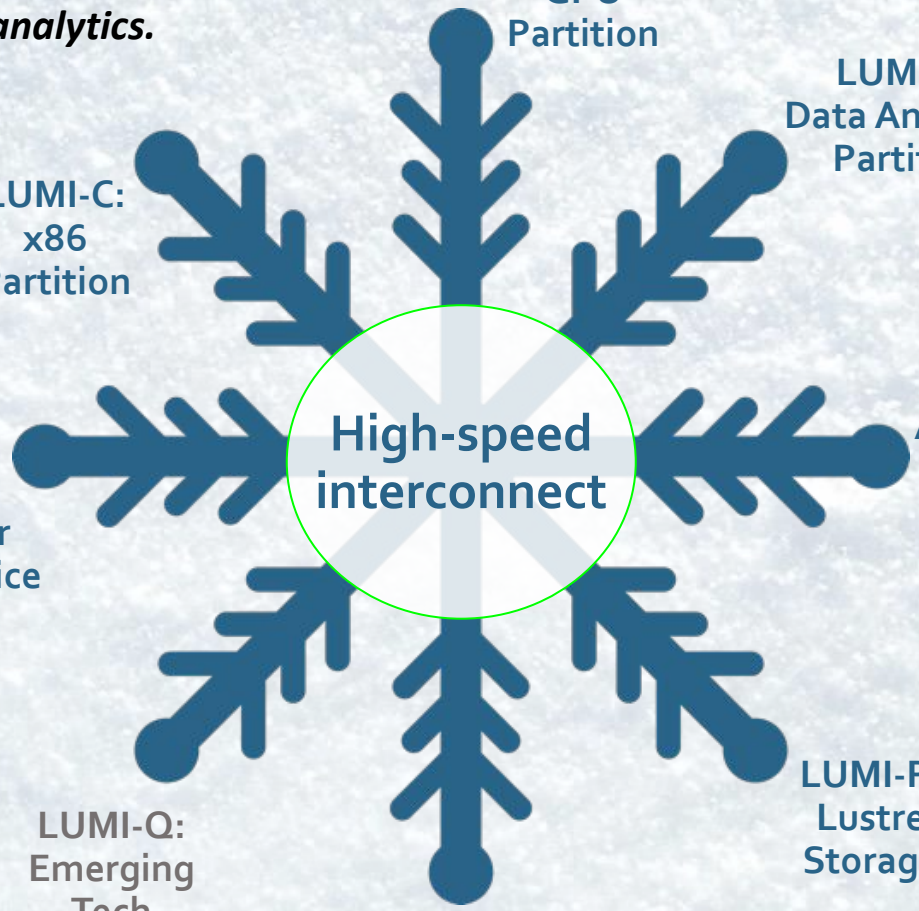
**32** TB of memory and graphics GPUs for data analytics and visualization.

Possibility for combining different resources within a single run. HPE Slingshot technology.

**7** PB Flash-based storage layer with extreme I/O bandwidth of 2 TB/s and IOPS capability.

**30** PB encrypted object storage (Ceph) for storing, sharing and staging data.

**80** PB parallel file system

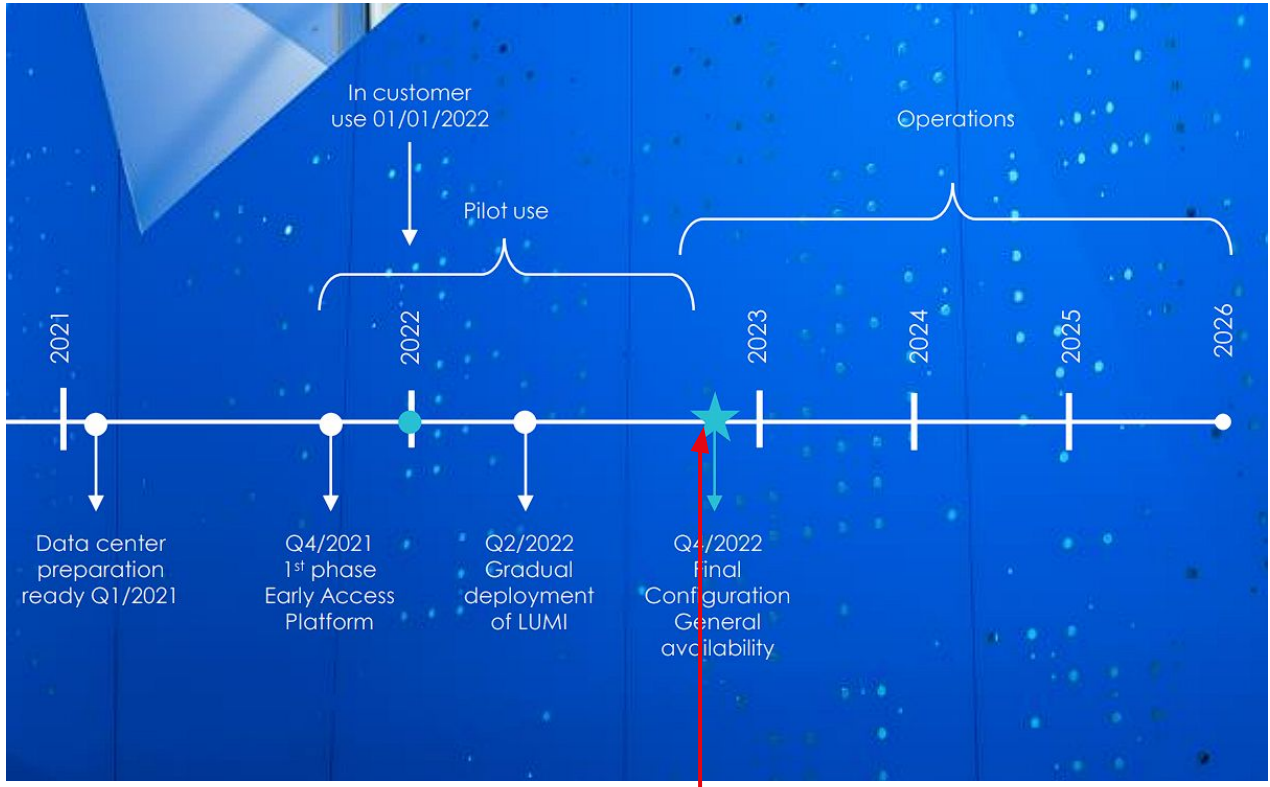


# System Overview

- LUMI is HPE Cray EX massively parallel processor system
- CPU Partition **LUMI-C**
  - 1536 nodes
  - 2 × AMD EPYC CPU
  - HPE Slingshot interconnect
- GPU Partition **LUMI-G**
  - 2560 nodes
  - 1 × AMD EPYC CPU
  - 4 × AMD MI250X GPU module
  - HPE Slingshot interconnect
- High Speed Interconnect **SlingShot**

# Current Status (Nov 2022)

- Status for November 2022
- **Production**
  - LUMI-C in full operation
  - GPU nodes only in EAP
- **Pre-production**
  - LUMI-G in acceptance
  - Pilot use
- Network and Filesystem in integration stage



# LUMI System

## LUMI-C nodes (1536 in total)

- 2 x AMD EPYC 7763 64 core 2.45GHz (128 cores per node)
- 256/512/1024 GB per (1372/128/32) nodes
- 8 NUMA regions per node
- 1 x 200Gb/s SlingShot NIC

# LUMI System

## LUMI-G nodes (2560)

- 1 x AMD EPYC 7A53 64 core
- 4 x AMD MI250X GPUs
  - 2 Graphics Compute Dies (GCDs) per GPU
  - 128 GB HBM2e per GPU
- 512 GB DDR4 main memory per node
- Each GPU connected to a Slingshot 200Gb/s NIC

# LUMI System

**SlingShot network**

- Dragonfly topology
- Compatible with Ethernet (Supports RDMA over Converged Ethernet)
- 200Gbps signaling
- GPU Nodes connected with 4 x 200Gbps NICs
- MPI Features support
  - Hardware tag matching
  - Progress engine
  - One-sided operations
  - Collectives
- Exposed in the user environment via optimized MPI library and/or Libfabric
  - Supported with RCCL

# LUMI Compute Capabilities

- LUMI-C can be considered as complementary, standard CPU resource
- LUMI-G provides most of the system's performance
  - Can be seen as 8-GPU nodes system
  - Optimized for remote GPU-GPU communication

# Paths to run on LUMI(-G)

**Code already supporting AMD GPUs**

- Use ROCm software stack

**Code already supporting other GPUs**

- CUDA: translate to HIP
- OpenCL: use ROCm OpenCL runtime
- OpenACC: use Cray Programming Environment
- SYCL: use HipSYCL

**Code without GPU support**

- C: go for OpenMP Offloading or HIP
- C++: go for High-level libraries, HIP or extensions like SYCL
- Fortran: go for OpenMP offloading or HIPFort plus ISO C Bindings (or GPUFORT)

# Paths to run on LUMI(-G)

**What if I use software framework?**

- PyTorch and TensorFlow are ported
- There is RCCL equivalent of NCCL
- Available as a container (AMD InfinityHub)
- Install using EasyBuild (recommended) or Spack

# ROCm Software Stack

```
> module av rocm
------------------------------------------------------------- HPE-Cray PE modules --------------------------------------------------------------
   rocm/5.0.2 (D)
  Where:
   D:  Default Module
```

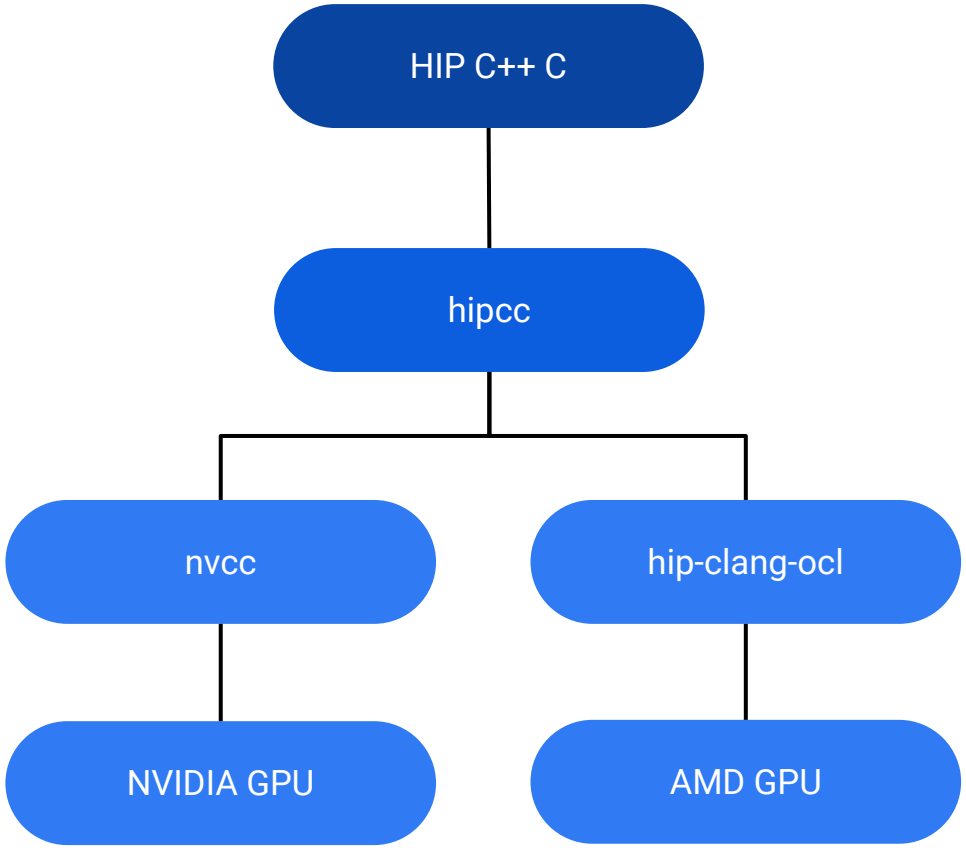
What it provides

- HIP compilers
- ROCm runtime
- ROCm libaries
  - rocBLAS, rocFFT, rocThrust, rocRAND, rocSolver
- Diagnostics tools
  - rocminfo, rocm-smi (requires physical GPU)
- hipify-perl/-clang

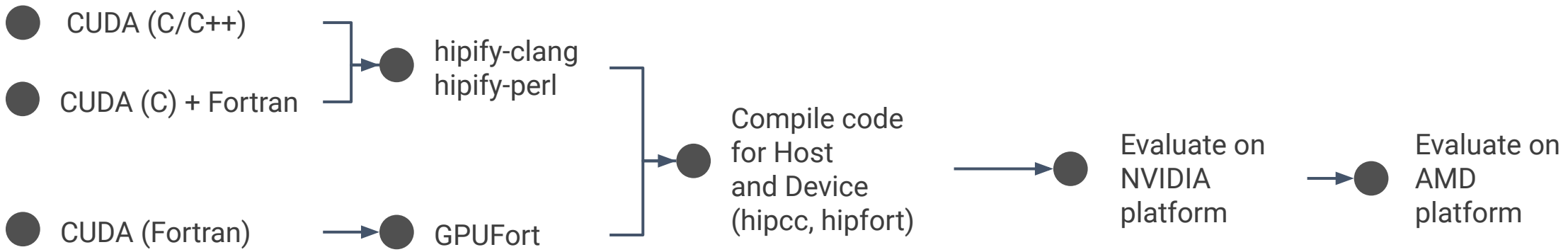
# HIP

HIP introduces Portability Layer

- allows unified kernel implementation
- little or no performance impact over native CUDA code
- single-source C++ programming language
- The HIPIFY tools automatically convert source from CUDA to HIP



# Converting CUDA to HIP




Tools

- `hipify-perl`
- `hipify-clang`

# Building codes with hipcc

- Select platform (NVIDIA/AMD)

  **`export HIP_PLATFORM=amd`**

- Select GPU target

  **`export GPU_TARGET=gfx90a`**

| PLATFORM | NVIDIA | AMD |
|---|---|---|
| TARGET | P100: sm_62 | MI25: gfx900 |
| | V100: sm_72 | MI50: gfx906 |
| | A100: sm_86 | MI100: gfx908 |
| | H100: sm_90 | MI2XX: gfx90a |

LUMI

# OpenMP Target Offload

- ≥4.5 supports target regions (GPU offloading)
- Compiler directives to “hint” at acceleration strategies
- Similar to OpenACC
- Supports Fortran and C/C++
- Supports multi-core CPU and GPU platforms (NVIDIA & AMD)

# OpenMP Programming Model

```
#pragma omp target map(to:A,x) map(Ax)
{
  for( int j = 0; j < N; j++ ){
    for( int i = 0; i < N; i++ ){
      for( int r = 0; r < N; r++) {
        Ax[i+N*j] = Ax[i+N*j]+A[r+N*i]*x[r]
      }
    }
  }
}
```

```
#pragma omp target map(to:A,x) map(Ax)
{
  #pragma teams distribute parallel for
  for( int j = 0; j < N; j++ ){
    for( int i = 0; i < N; i++ ){
      for( int r = 0; r < N; r++) {
        Ax[i+N*j] = Ax[i+N*j]+A[r+N*i]*x[r]
      }
    }
  }
}
```

**`target`** and **`map`** directives

- Target directive creates a target task to be executed on the device (GPU)
- Map clause allocates space on the device's memory (GPU) and copies data between host (CPU) and device (GPU)

**`teams`** and **`parallel`** directives

- Teams directive creates a league of teams with a single thread per team
- Distribute construct specifies the do/for loop iterations will be distributed amongst the teams
- Parallel directive makes the teams multi-threaded

# Building codes with OpenMP Offload

LUMI

- Enable OpenMP pragmas

  **-fopenmp**

- Select GPU Target

  **-fopenmp-targets=<target>**

- Select GPU Architecture

  **-march=<gpu-arch>**

| Target | nvptx64-nvidia-cuda | amdgcn-amd-amdhsa |
|---|---|---|
| GPU Architecture | P100: sm_62 | MI25: gfx900 |
| | V100: sm_72 | MI50: gfx906 |
| | A100: sm_86 | MI100: gfx908 |
| | H100: sm_90 | MI2XX: gfx90a |

# Tricks for Smart Execution

Lets consider PyTorch execution example

```
python main.py -a resnet18 --dummy --epochs 1
```

# Tricks for Smart Execution

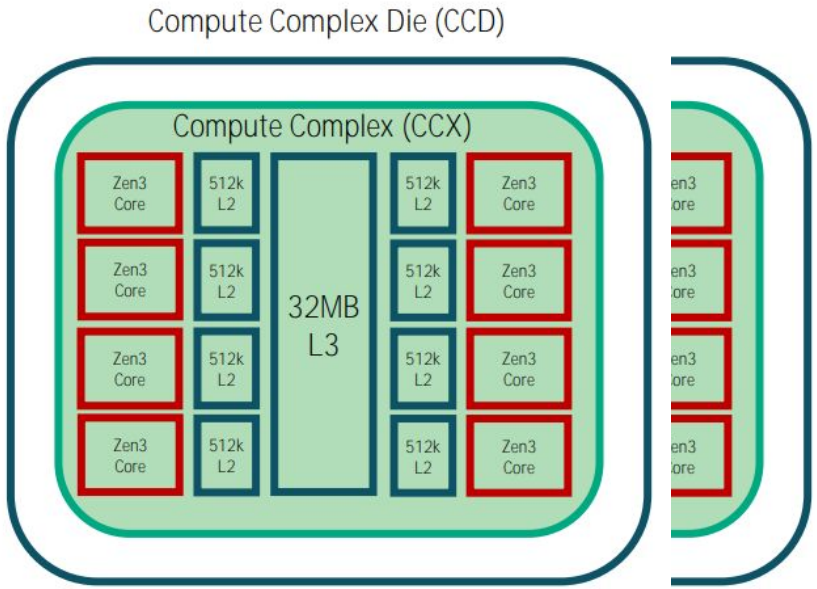
1. Run on a compute node over SLURM resource manager

```
srun --partition=standard --ntasks=1 --cpus-per-task=64 --time=20:00 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Compute Node Architecture

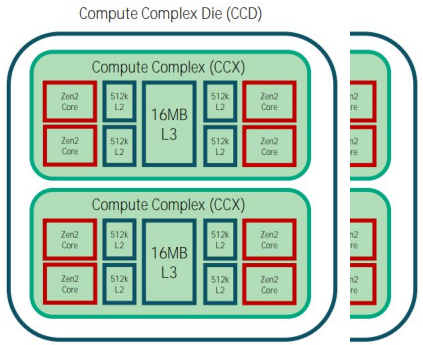
## LUMI-C Node

- AMD EPYC 7763 - Zen3 "Milan" architecture
- 2 x 64-core Milan CPU




## LUMI-L (login) Node

- Different CPU type
- Zen2 "Rome" architecture



# Tricks for Smart Execution

2. Run on a compute node with GPU allocated

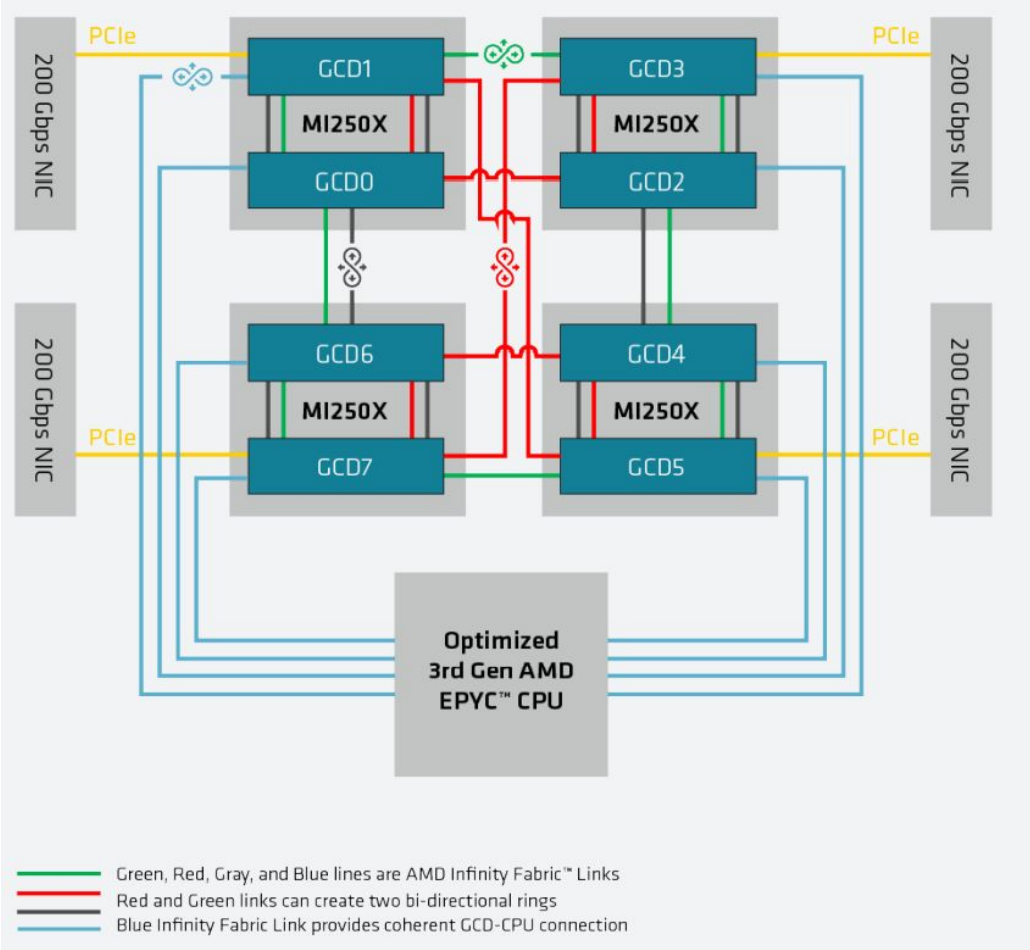
```
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=1 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Compute Node Architecture

Source:
https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

## LUMI-G Node

- AMD Optimized EPYC 7A53 - Zen3 "Trento"
- 64-core "Trento" CPU
- 8 GCDs act as 8 separate GPUs - "CDNA2" architecture
  - each having 64 GB of high-bandwidth memory (HBM2E)
- The CPU is connected to each GCD via Infinity Fabric CPU-GPU
  - Coherent memory CPU-GPU
- The 2 GCDs on the same MI250X are connected with Infinity Fabric GPU-GPU
- The GCDs on different MI250X are connected with Infinity Fabric GPU-GPU in the arrangement shown in the diagram on the right
- The yellow interface is a PCIe 4.0 ESM link, coupled to a PCIe root complex, which can drive I/O devices that are connected to the GPU. With it, CPUs and GPUs are acting as equals in a system.



# Tricks for Smart Execution

3. Run in the correct environment

```
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=1 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Module Environment

LUMI uses Lmod modules framework to provide multiple versions and combinations of software

```
> module load LUMI/22.08
> module available

------------------------------- Software stacks ------------------------------------------------
   CrayEnv (S)    LUMI/21.12 (S,D)    LUMI/22.06 (S)    LUMI/22.08 (S,L)    spack/22.08
```

# Tricks for Smart Execution

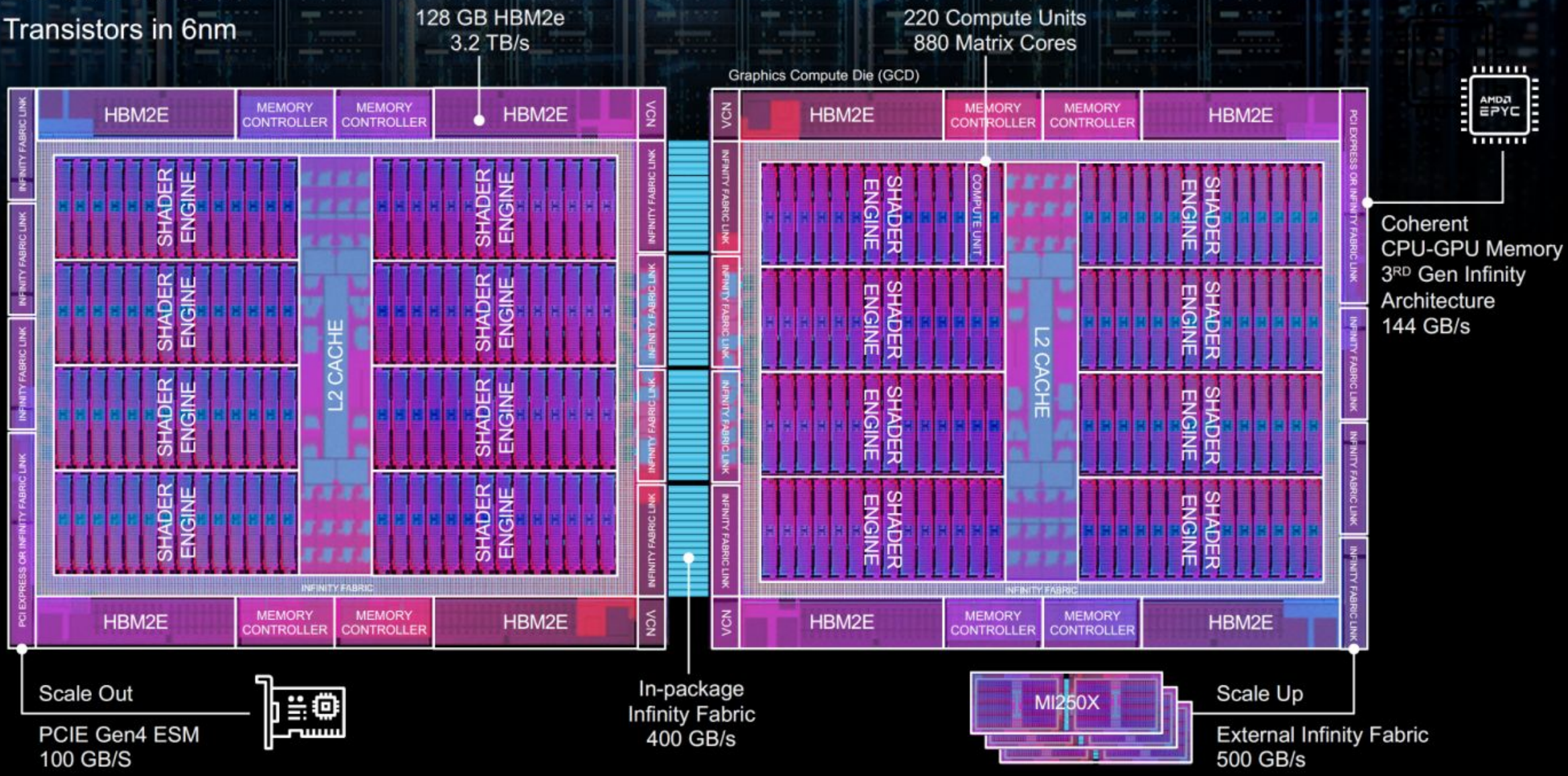
## 4. Go for more GPUs

```
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=8 \
    python main.py -a resnet18 --dummy --epochs 1
```

This will fail and needs some fine tuning

# MI250X MCM

58B Transistors in 6nm




Connecting to the PCI root complex (NIC)

Connecting the other comput devices (GPU)



AMD

# Tricks for Smart Execution

## 5. Tune RCCL options

```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=8 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Tricks for Smart Execution

## 5. Tune RCCL options

specifies which network interface to use for communication

indicates to use GPUDirect RDMA when GPU and NIC are on the same PCI root complex (level 3)

```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=8 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Tricks for Smart Execution

## 6. Enable GPU aware MPI

```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3
export MPICH_GPU_SUPPORT_ENABLED=1
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
    --gpus-per-node=8 \
    python main.py -a resnet18 --dummy --epochs 1
```

# Cray MPICH

- Implementation based on MPICH3 source from ANL
- Includes many improved algorithms and tweaks for Cray hardware
- Improved algorithms for many collectives
- Asynchronous progress engine allows overlap of computation and comms
- Customizable collective buffering when using MPI-IO
- Optimized Remote Memory Access (one-sided) fully supported including passive RMA
- MPI_LONG_DOUBLE and MPI_C_LONG_DOUBLE_COMPLEX for CCE
- Includes support for Fortran 2008 bindings
- **GPU-aware communications**

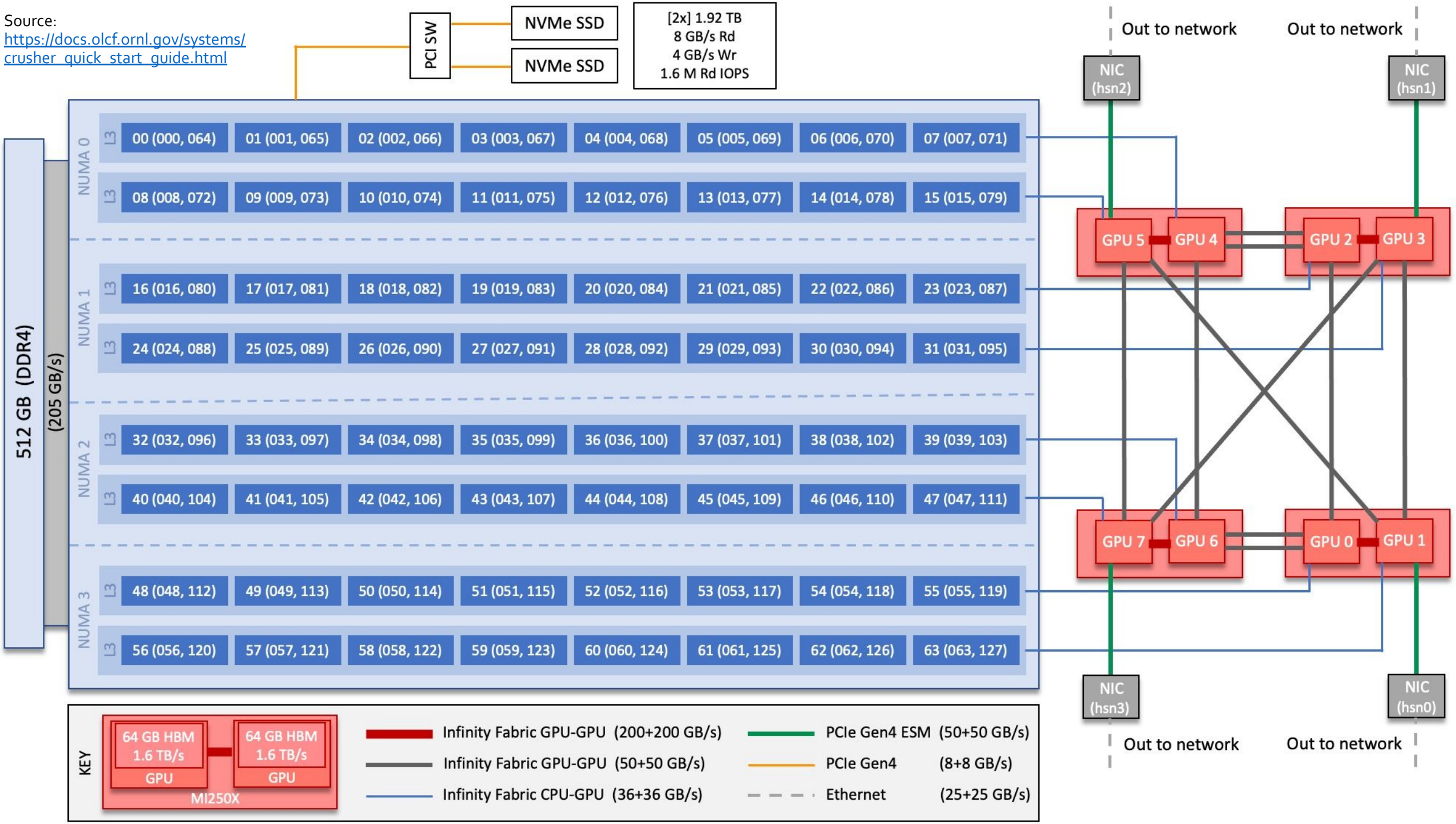
# Tricks for Smart Execution

## 7. Setup optimal GPU binding

```
export NCCL_SOCKET_IFNAME=hsn0,hsn1,hsn2,hsn3
export NCCL_NET_GDR_LEVEL=3
export MPICH_GPU_SUPPORT_ENABLED=1
module load LUMI/22.08
module load partition/G
module load rocm
module load PyTorch/1.12.1-cpeGNU-22.08
srun --partition=gpu --ntasks=1 --cpus-per-task=64 --time=20:00 \
     --gpus-per-node=8 \
     ./select_gpu.sh \
     python main.py -a resnet18 --dummy --epochs 1
```

```
#!/bin/bash
GPUSID="4 5 2 3 6 7 0 1"
GPUSID=(${GPUSID})
if [ ${#GPUSID[@]} -gt 0 ]; then
  export ROCR_VISIBLE_DEVICES=${GPUSID[$((SLURM_LOCALID / ($SLURM_NTASKS_PER_NODE / ${#GPUSID[@]})))]}
fi
exec $*
```

Source:
https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html



# Differences in Terminology and Architecture

LUMI

| NVIDIA | AMD |
|---|---|
| Workload Distributor | Command Processor |
| Streaming Multiprocessor | Compute Unit |
| Warp | Wavefront |
| Warp size = 32 | Wavefront size = 64 |
| Maximal workgroup size 1024 (32 warps) | Maximal workgroup size 1024 (16 wavefronts) |

- Kernels are structured into work groups that map to device compute units
- Compute units on GPUs consist of SIMT processing elements
- Workgroups are broken down into hardware schedulable groups of threads for the SIMT hardware
- These groups are referred to as warps or wavefronts

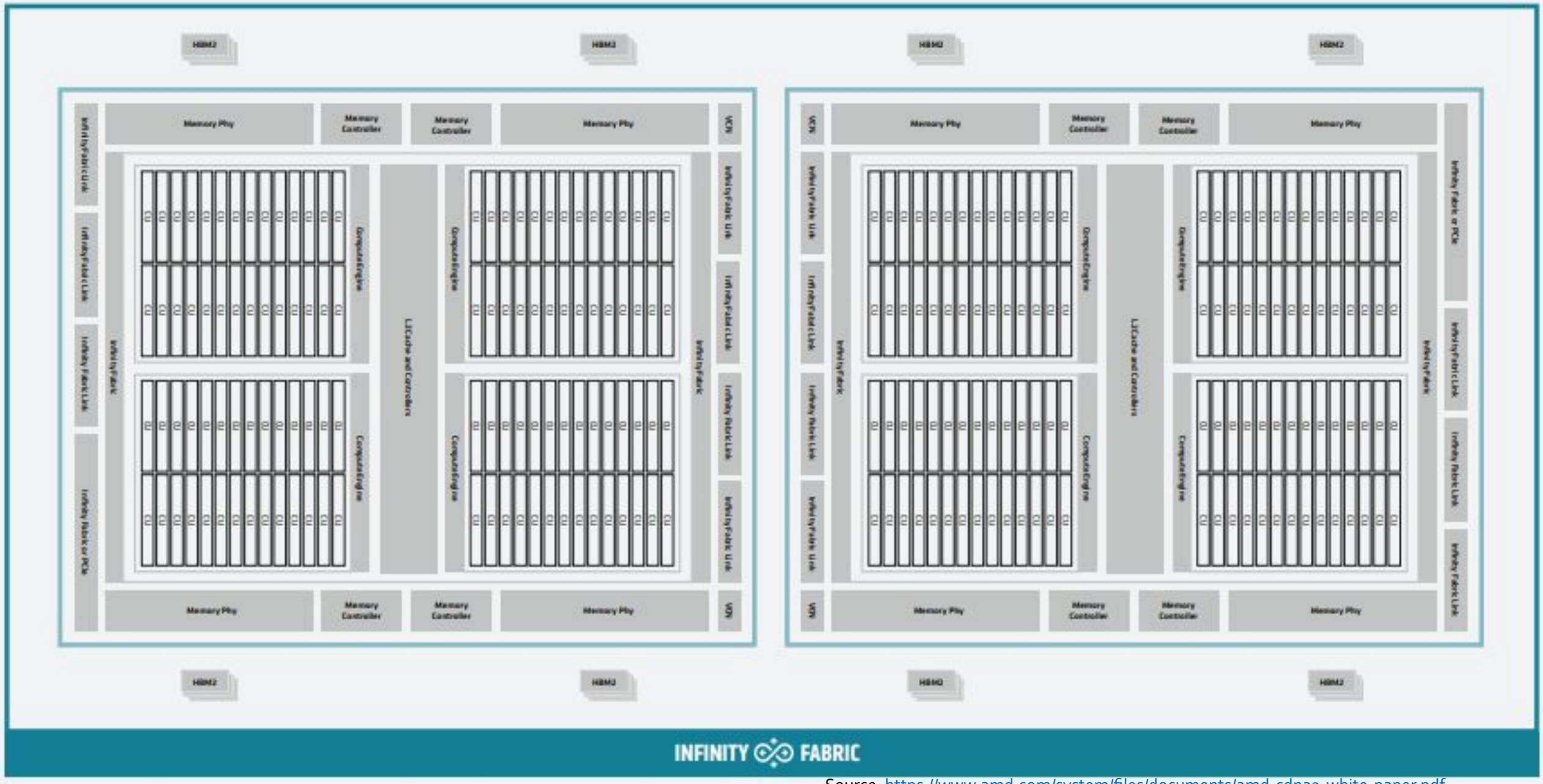
# Node Types

## Login nodes

- User Access Nodes (uan01…uan04 currently)
- Run full version of SLES Linux
- Environment partition `L`

## Compute nodes

- Run the optimized version of Linux (Cray)
- Only accessible via allocation in SLURM (salloc or sbatch)
- Environment partitions `C`, `G`




Source: https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

# LUMI Software Stack

There are two choices for accessing software packages:

- **Full-feature LUMI Environment**
  - Available with `**LUMI**` module
  - Based on EasyBuild specific contribution
- Native Cray Programming Environment
  - Available with `**CrayEnv**` module
  - Utilizes PrgEnv- and craype- modules for environment selection
  - Software provided by the vendor
- Both follow same numbering convention i.e.
  - Now default `**LUMI/22.08**` and `**cpe/22.08**` (CrayEnv)

# What if my App is not there?

# Discovering Available Software

```
> module load EasyBuild
> eb --list-installed-software
```

```
> module load EasyBuild-user
> eb --list-software
```

# EasyBuild

There are two public GitHub repositories:

- **LUMI EasyBuild configs** - installed packages
- **Contrib. EasyBuild configs** - packages available for self-installation

Installing packages with EasyBuild

- `module load EasyBuild-user`
- `eb FullEasyConfigName.eb`
- Useful options: `--trace, --robot`
- Packages are installed for active partition only
- EasyBuild on LUMI uses specific toolchains, do not use plain installation (from pip or git clone)

# Partition Modules

```
> module avail partition
-------------------------------- LUMI partitions for the software stack LUMI/22.08 ---------------
   partition/C (S,CPUcompute)    partition/L (S,L,D:login)      partition/G   (H,S,GPUcompute)
```

# What if my App is still not there?

LUMI

# Programming Environment

- Cray Programming Environment
  - Cray Compiling Environment (CCE) -Optimizing compilers for Fortran, C, C++ and UPC
  - Cray Scientific and Math Libraries (LibSci, FFTW, NetCDF, HDF5)
  - Cray Message Passing Toolkit - Cray MPICH and OpenSHMEM
  - Cray Performance Measurement and Analysis Tools - Cray PAT
  - Cray Debugging Support Tools
- GNU Compilers Collection
- AMD Compiler Suite
  - ROCm programming environment and libraries for AMD GPU
- For both GNU and AMD there are Cray libraries provided

# Compilers

| Compiler Family | Language | Cray | AMD | GNU |
|---|---|---|---|---|
| Programming Environment | | `cpeCray`<br>`PrgEnv-cray` | `cpeAMD`<br>`PrgEnv-amd` | `cpeGNU`<br>`PrgEnv-gnu` |
| Compiler Wrapper | `C` | `cc` | `cc` | `cc` |
| | `C++` | `CC` | `CC` | `CC` |
| | `Fortran` | `ftn` | `ftn` | `ftn` |
| | `HIP` | `CC -x hip` | `hipcc` | `-` |

## Compiler drivers

- GNU Collection (v11): gcc, g++, gfortran
- Cray: craycc/craycxx (clang-14), Cray Fortran crayftn
- AMD: AOCC (clang-13), flang
- HIP: AMD clang-14/Cray clang-14

# PE Libraries

**Cray provided**

- LibSci - includes BLAS, CBLAS, BLACS, LAPACK, ScaLAPACK
- Cray MPICH - MPI Implementation
- Cray Python
- HDF5, Parallel HDF5
- NetCDF, Parallel NetCDF
- FFTW

**ROCm**

- rocBLAS, rocFFT, rocThrust, rocRAND, rocSolver
- RCCL
- HIP

# PE Modules

```
> module av craype
------------------------------------------------------------ HPE-Cray PE target modules ------------------------------------------------------------
   craype-accel-amd-gfx908        craype-hugepages16M     craype-hugepages32M     craype-network-none        craype-x86-spr
   craype-accel-amd-gfx90a        craype-hugepages1G      craype-hugepages4M      craype-network-ofi  (L)    craype-x86-trento
   craype-accel-host       (L)    craype-hugepages256M    craype-hugepages512M    craype-network-ucx
   craype-accel-nvidia80          craype-hugepages2G      craype-hugepages64M     craype-x86-milan
   craype-hugepages128M           craype-hugepages2M      craype-hugepages8M      craype-x86-rome     (L)


  Where:
   L:  Module is loaded
```

# Cross-Compilation

- The login nodes have AMD “Rome” CPUs, the CPU nodes - “Milan” while the GPU nodes - “Trento” CPUs
- You can decide on the compilation target with
  - Partition modules from the LUMI environment: **`partition/L, /C, /G`**
  - Cray target CPU modules: **`craype-x86-rome, -milan, -trento`**
- The wrappers ftn, cc, CC will use the corresponding CPU target flag
  - You can use the flag **`-craype-verbose`** to check which flags are used by the backends
- Caution:
  - The apps compiled for compute nodes might not run on login nodes
  - Avoid directly specifying compiler hardware target flags, e.g. *`-march=native -mtune=native -mavx2`*

# General User Environment

- Modules Environment
  - Provides access to Programming Environment and Software Stack
- Programming Environment
  - Provides compilers, base libraries and tools
- Software Stack
  - Consists of PE and optimized software packages
- SLURM
  - Controls user resource allocations and job execution

# Getting support

- PL-Grid Helpdesk (recommended)
  - https://helpdesk.plgrid.pl/
- LUMI Helpdesk
  - https://lumi-supercomputer.eu/user-support/need-help/
- Monthly User's coffee break (Wednesday at 13 CE[S]T)
  - Announced in advance
- F2F consultancy
  - Please contact User Support member directly (read me)
- Porting Calls
- Trainings for LUMI
  - Announced on LUMI Website

# Where to find more information

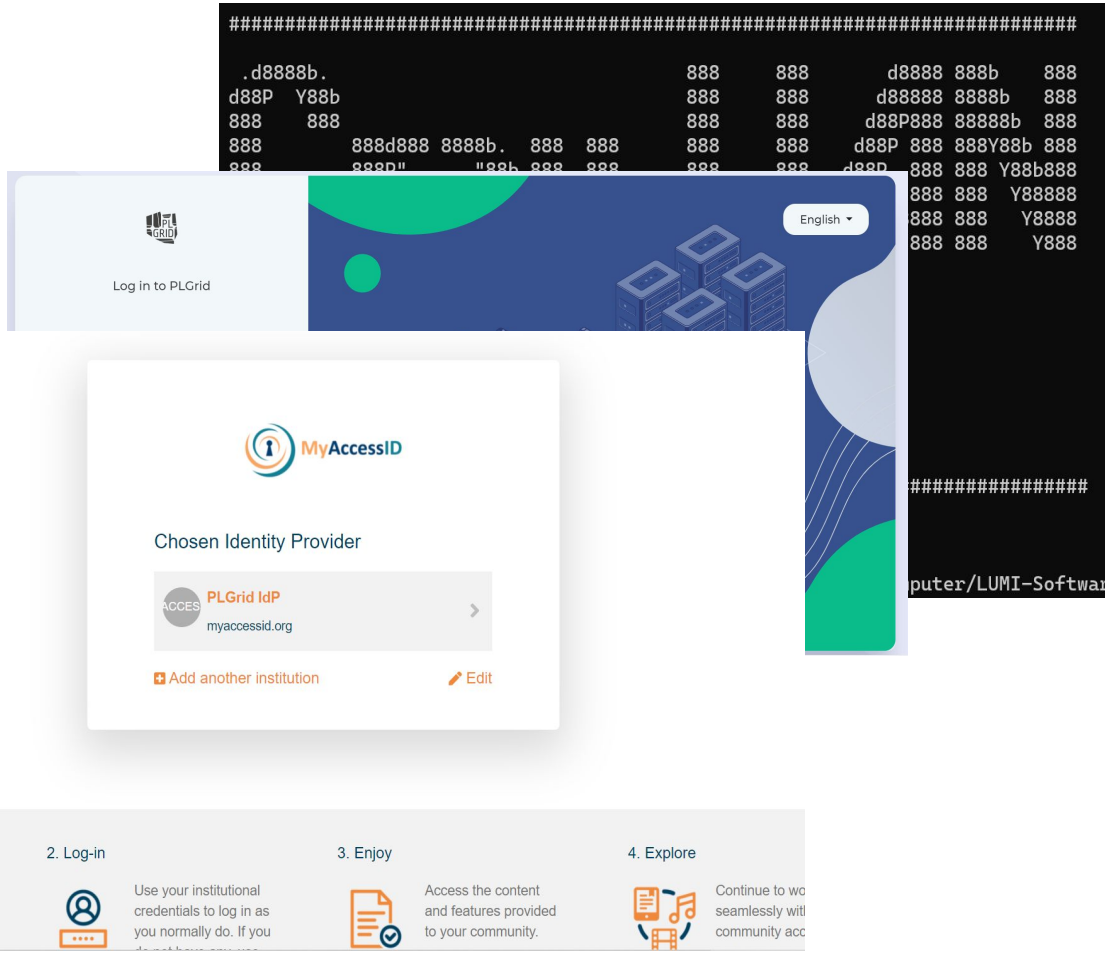
- LUMI Software Stack EasyConfig files (GitHub)
  - https://github.com/Lumi-supercomputer/LUMI-SoftwareStack
- LUMI Contributed EasyConfig files (GitHub)
  - https://github.com/Lumi-supercomputer/LUMI-EasyBuild-contrib
- EasyBuild documentation
  - https://docs.easybuild.io/en/latest/
- AMD ROCm documentation
  - https://docs.amd.com/
- Crusher Quick-Start Guide
  - https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html
- An In-Depth Analysis of the Slingshot Interconnect
  - https://arxiv.org/pdf/2008.08886
- AMD CDNA™ 2 ARCHITECTURE White Paper
  - https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

# How to get Apply

- Apply via PLGrid Portal
  - **Regular Access**
    - Half-year calls /March, September/
  - **Preparatory Access**
    - Continuous call
- Alternative access path - EuroHPC JU resource share
  - via PRACE access calls https://prace-ri.eu/hpc-access/eurohpc-access/

# How to Access

- Project based allocations
- Federative authentication
- Identity provided by PLGrid
  - Log in with your PLGrid cretential
  - You need to provide your SSH key



# More Opportunities

- **Porting Project Call**
  - PL - no projects
- **Regular Project Calls**
  - National - Cyfronet, EuroHPC
- **Test Projects**
  - National - Cyfronet

LUMI

# LUMI User Support Team

**Maciej Szpindler**

**m.szpindler@cyfronet.pl**

**Follow us**

**Twitter:** @LUMIhpc

**LinkedIn**: LUMI supercomputer

**YouTube**: LUMI supercomputer

www.lumi-supercomputer.eu

contact@lumi-supercomputer.eu

EuroHPC Joint Undertaking

The acquisition and operation of the EuroHPC supercomputer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and the Horizon 2020 research and innovation programme, as well as the of Participating States FI, BE, CH, CZ, DK, EE, IS, NO, PL, SE.

Leverage from the EU 2014–2020

European Union
European Regional Development Fund

REGIONAL COUNCIL OF KAINUU

EURO