
May 20-22, 2025
ALCF
INCITE GPU Hackathon

On Boarding on Aurora

May 6, 2025

Marta García Martínez
Computational Scientist
Argonne National Laboratory

TOP500 List

https://www.top500.org/lists/top500/list/2024/11/

(*) TOP #1

LLNL El Capitan

AMD CPU / AMD GPU

HIP

44,544 devices

(*) TOP #2

ORNL Frontier

AMD CPU / AMD GPU

HIP

37,632 devices

(*) TOP #3

ANL Aurora

Intel CPU / Intel GPU

SYCL/DPC++

63,744 devices

NOTE *In terms of numbers of GPUs, Aurora is the largest of the three DOE Exascale systems*

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698

In production

Monday, January 27th, 2025

#3 in Top500 w/9234 nodes – HPL 1.012 exaFLOPS Max

#1 in Top500 w/9500 nodes – HPL-MxP 11.6 exaFLOPS

Aurora

Argonne's exascale supercomputer will leverage several technological innovations to support machine learning and data science workloads alongside traditional modeling and simulation runs.

SUSTAINED PERFORMANCE
≥2 Exaflop DP

Xe ARCHITECTURE-BASED GPU
Ponte Vecchio

INTEL XEON SCALABLE PROCESSOR
Sapphire Rapids

PLATFORM
HPE Cray EX

Compute Node
2 Intel® Xeon CPU Max Series processors: 64GB HBM on each, 512GB DDR5 each; 6 Intel Data Center GPU Max Series, 128GB on each, RAMBO cache on each; Unified Memory Architecture; 8 Slingshot 11 fabric endpoints

GPU Architecture
Xe arch-based "Ponte Vecchio" GPU
Tile-based chiplets, HBM stack,
Foveros 3D integration, 7nm

CPU-GPU Interconnect
CPU-GPU: PCIe; GPU-GPU: Xe Link

System Interconnect
HPE Slingshot 11, Dragonfly topology with adaptive routing, Peak Injection bandwidth 2.12 PB/s, Peak Bisection bandwidth 0.69 PB/s

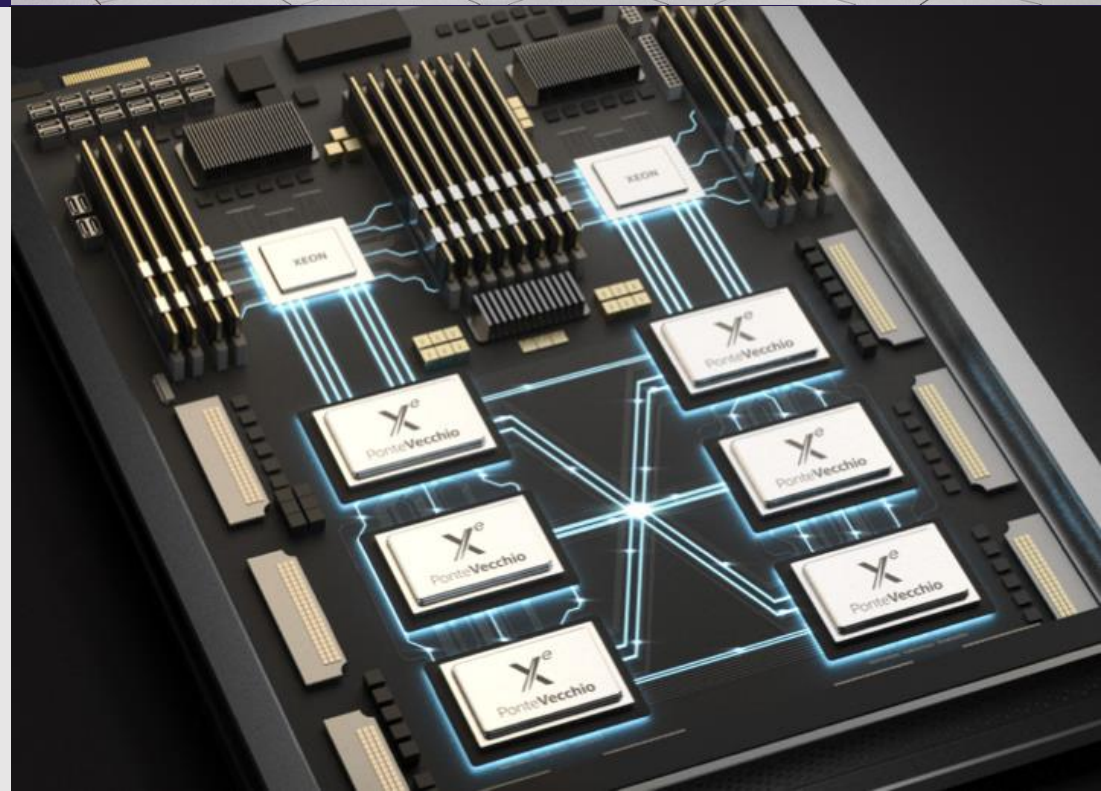
Network Switch
25.6 Tb/s per switch, from 64–200 Gbs ports (25 GB/s per direction)

High-Performance Storage
230 PB, 31 TB/s, 1024 nodes (DAOS)

Programming Models
Intel oneAPI, MPI, OpenMP, C/C++, Fortran, SYCL/DPC++

Node Performance
>130 TF

System Size
10,624 nodes, 166 compute racks
CPUs: 21,248
GPUs: 63,744

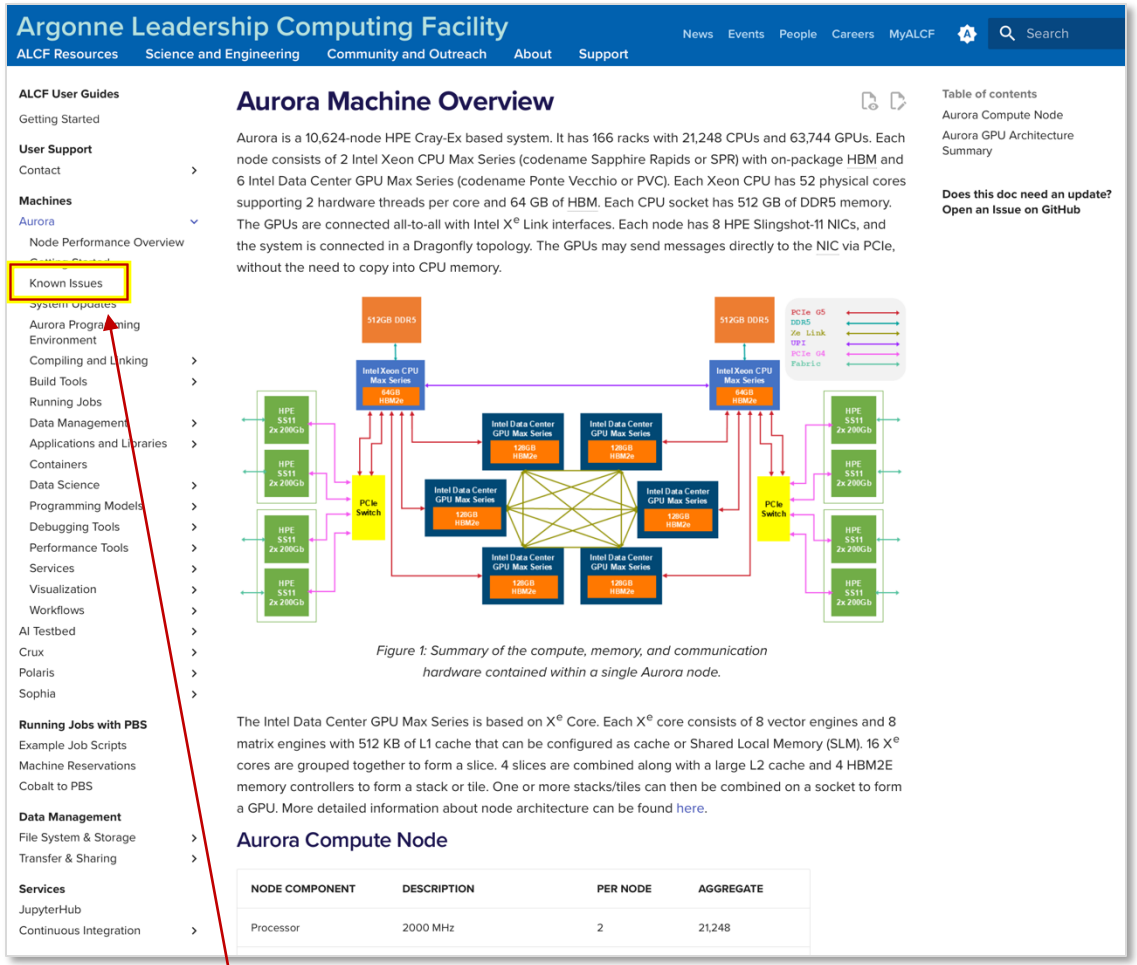


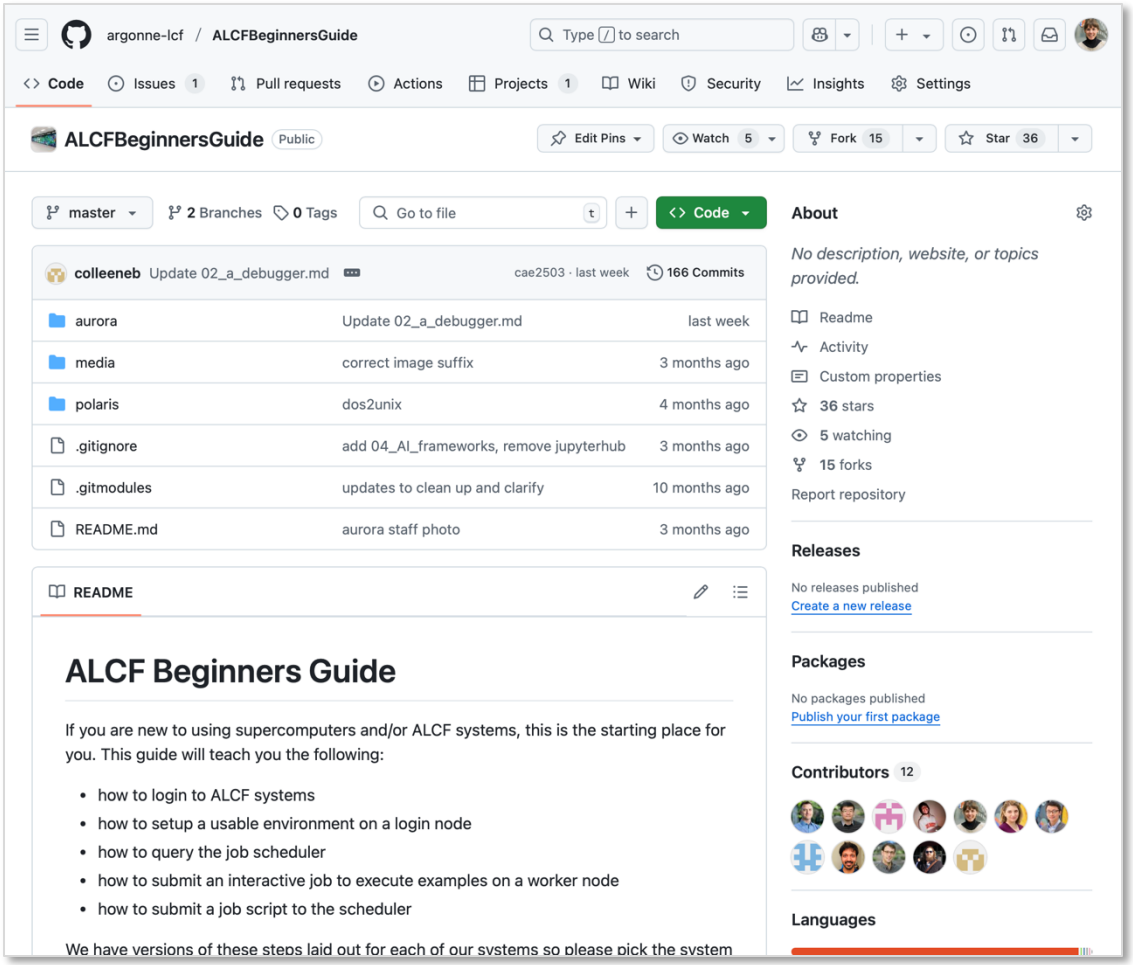
AURORA BEGINNERS GUIDE

Favorites

https://docs.alcf.anl.gov/aurora/

https://github.com/argonne-lcf/ALCFBeginnersGuide

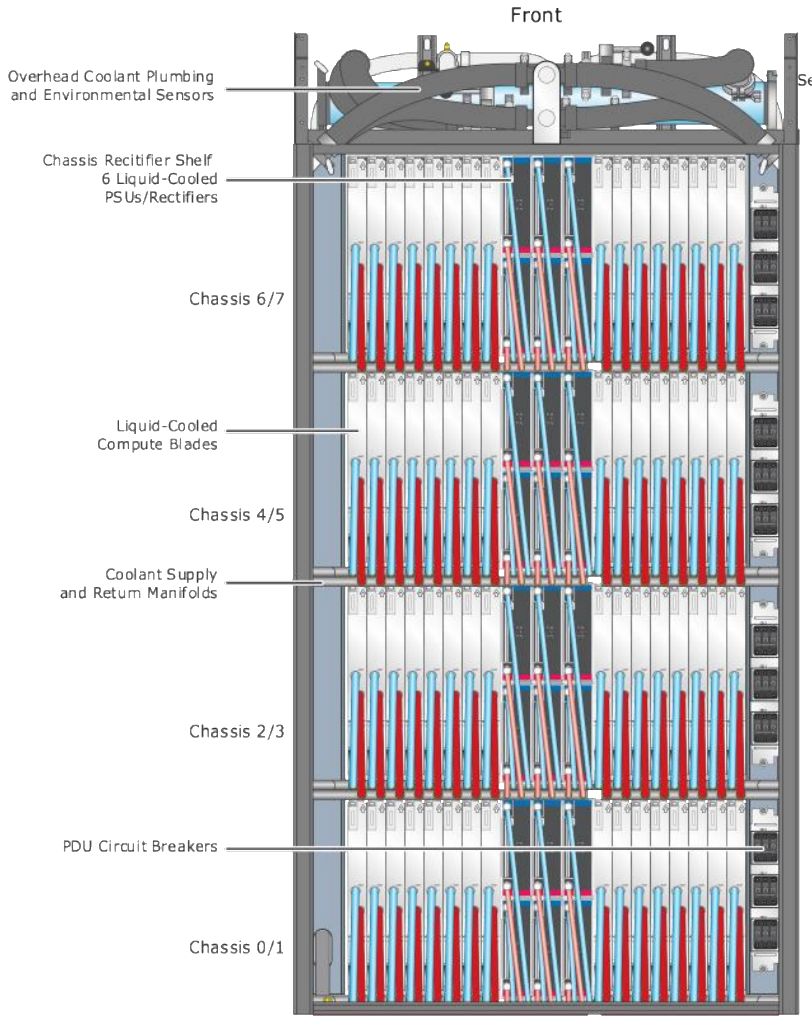


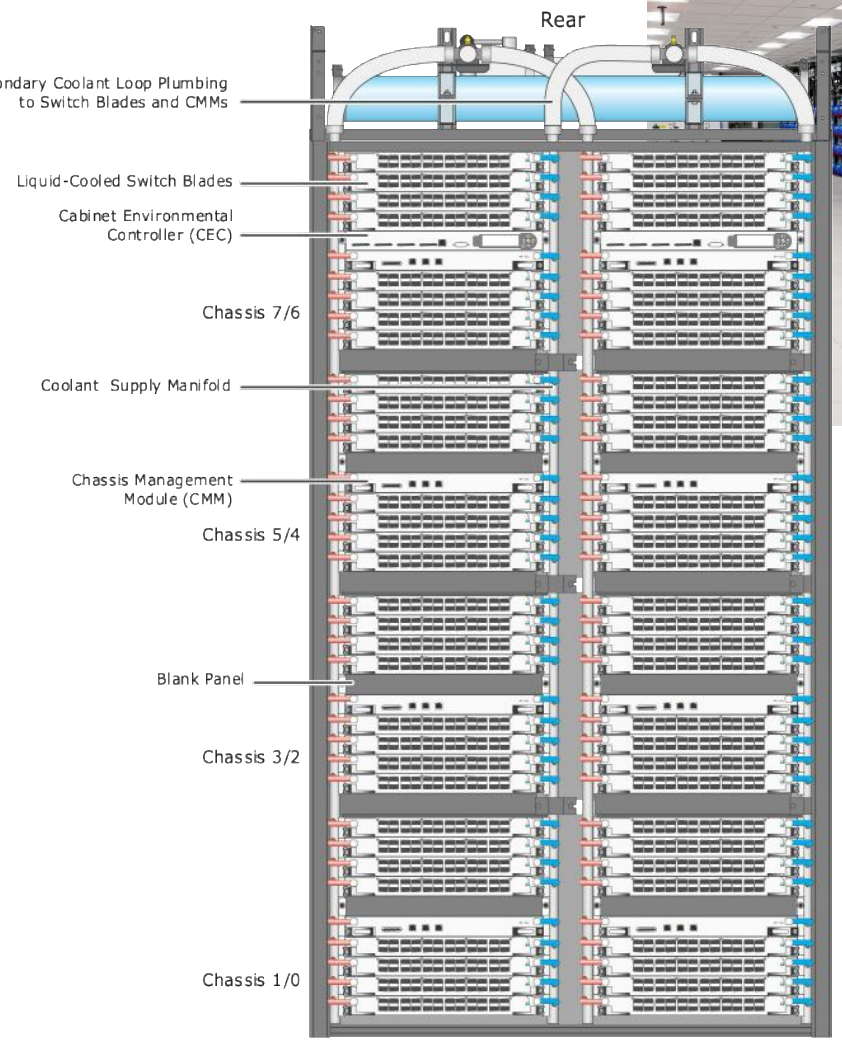


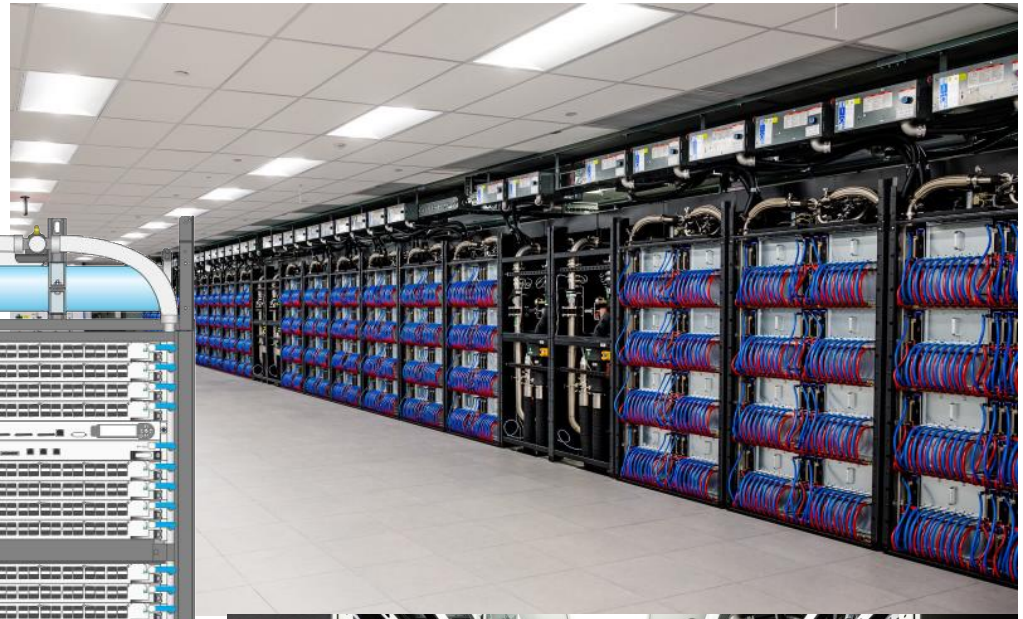
Known issues: https://docs.alcf.anl.gov/aurora/known-issues/

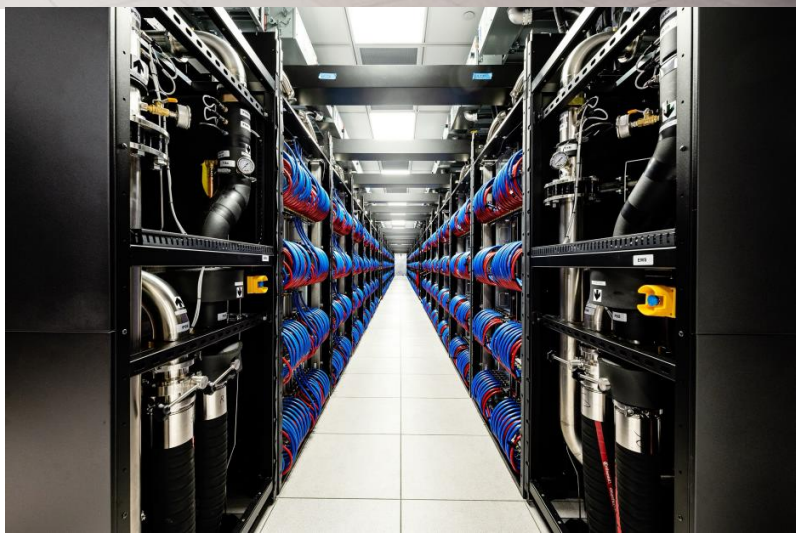
NOTE https://github.com/argonne-lcf/GettingStarted/tree/incite-hackathon-2025

AURORA CABINETS AT ARGONNE

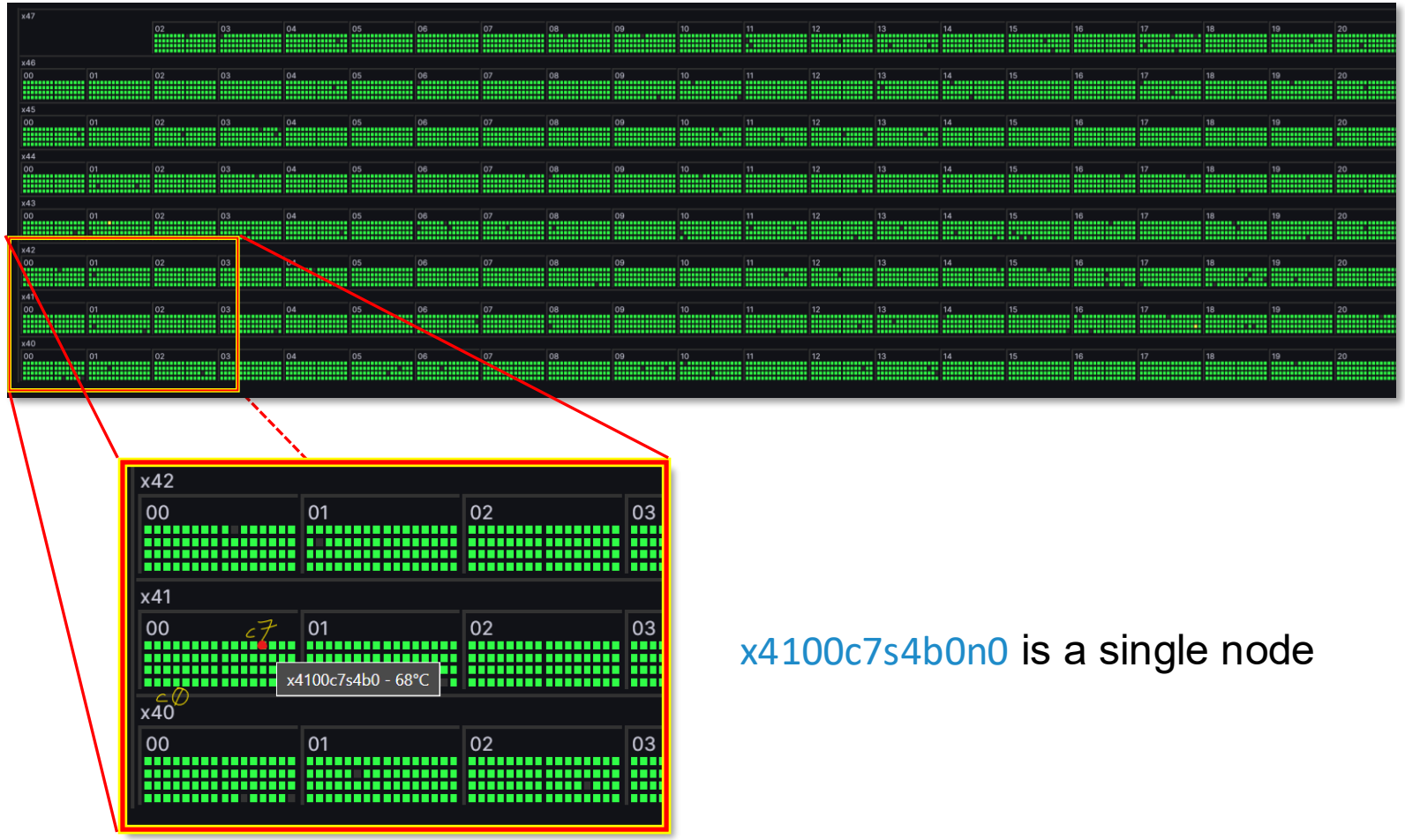




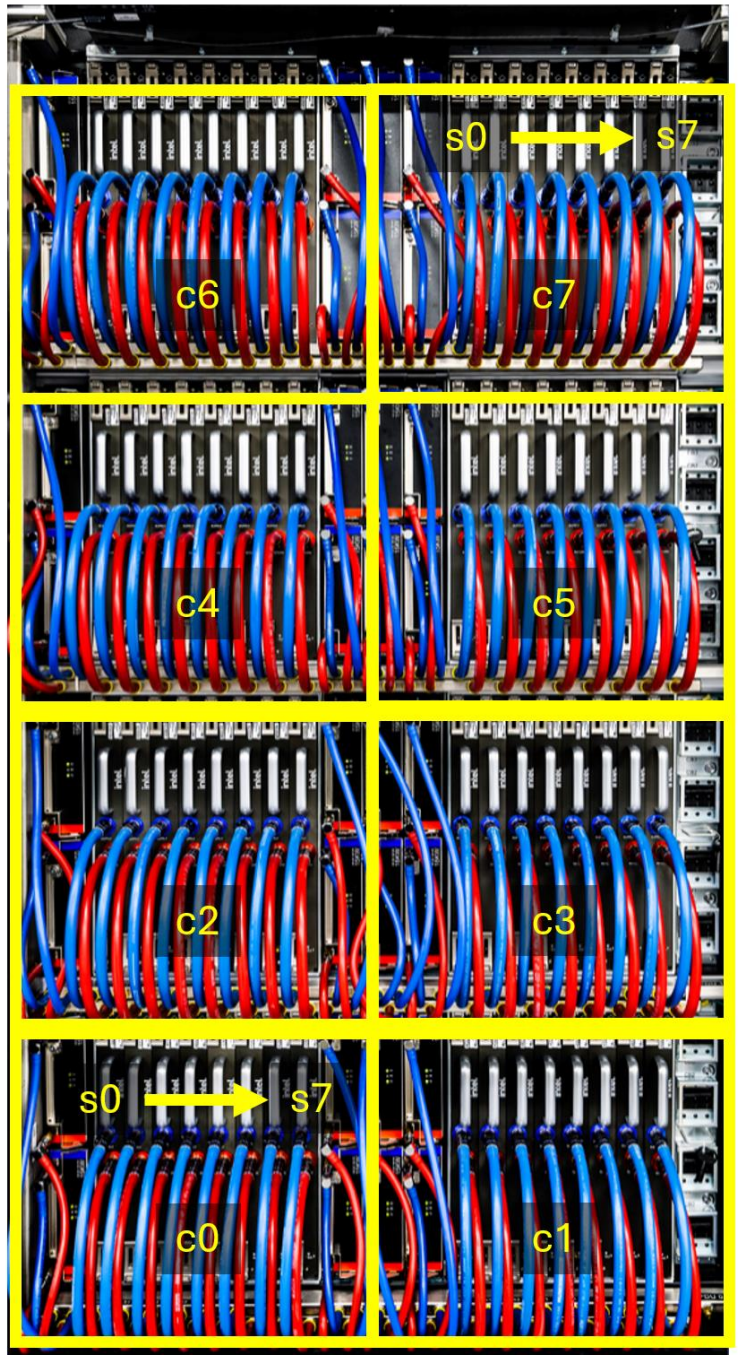




AURORA NODES NAME CONVENTION



x4100c7s4b0n0 is a single node

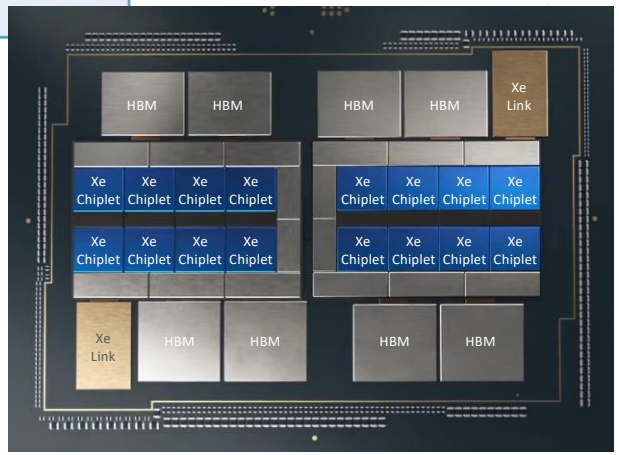


x4100c7s4b0n0 == Rack x4100 Chassis c7 Slot s4 Board b0 Node n0

never changes on Aurora
(board 0, node 0)

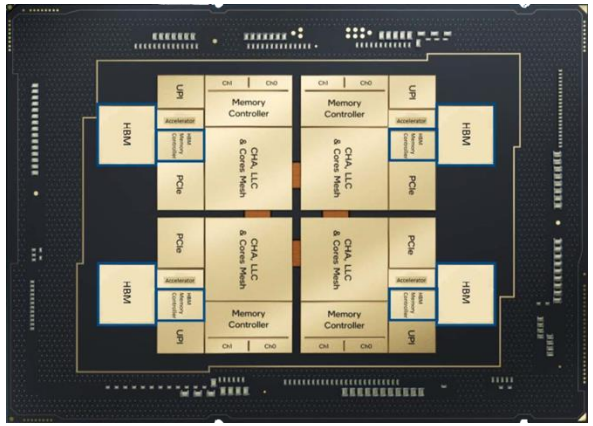
NODE CHARACTERISTICS

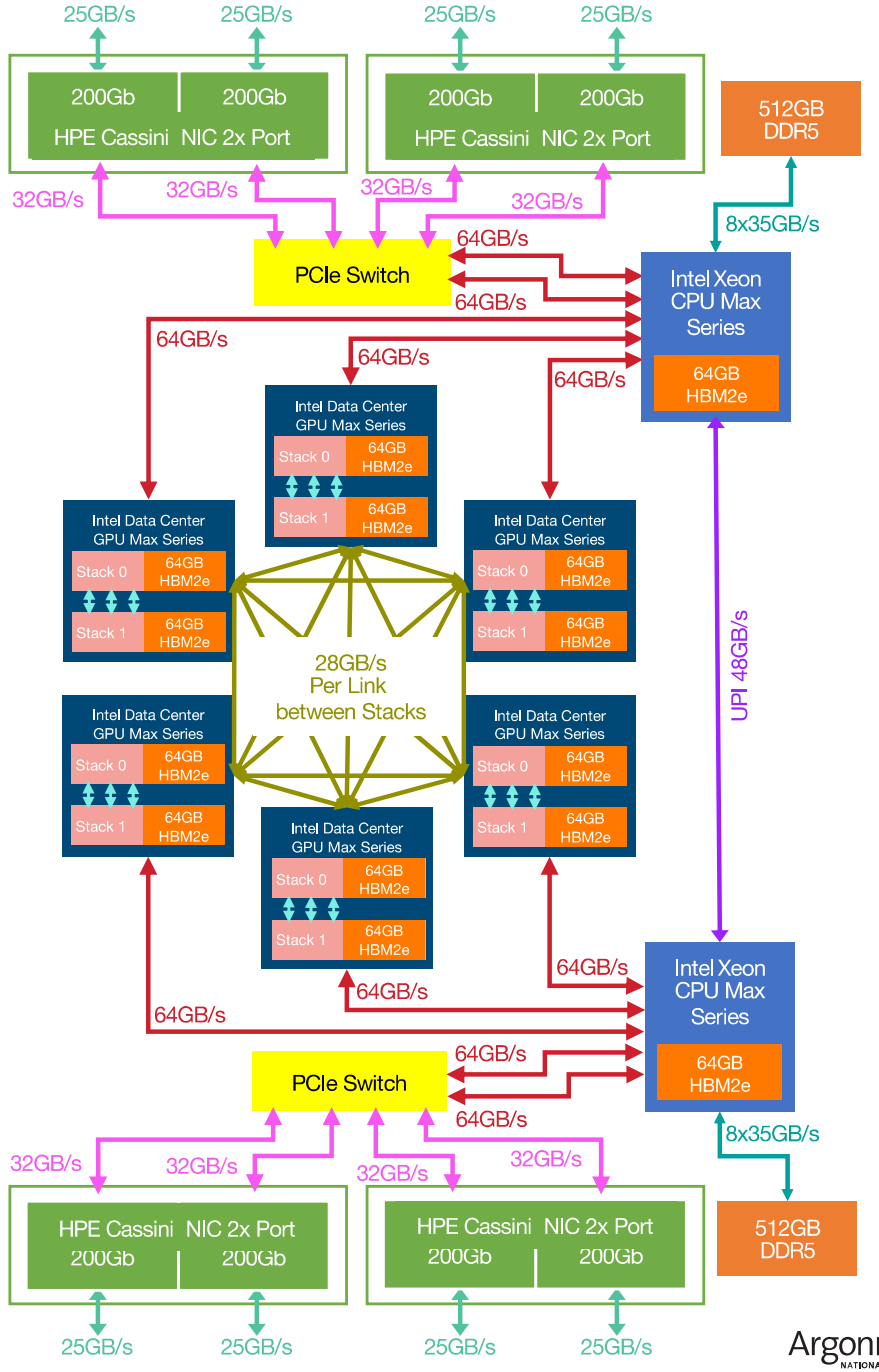
NODE CHARACTERISTICS	
6	GPU - Intel Data Center GPU Max Series (#)
2	CPU - Intel Xeon CPU Max Series (#)
768	GPU HBM Memory (GB)
19.66	Peak GPU HBM BW (TB/s)
128	CPU HBM Memory (GB)
2.87	Peak CPU HBM BW (TB/s)
1024	CPU DDR5 Memory (GB)
0.56	Peak CPU DDR5 BW (TB/s)
≧ 130	Peak Node DP FLOPS (TF)
200	Max Fabric Injection (GB/s)
8	NICs (#)



Intel® Data Center GPU Max

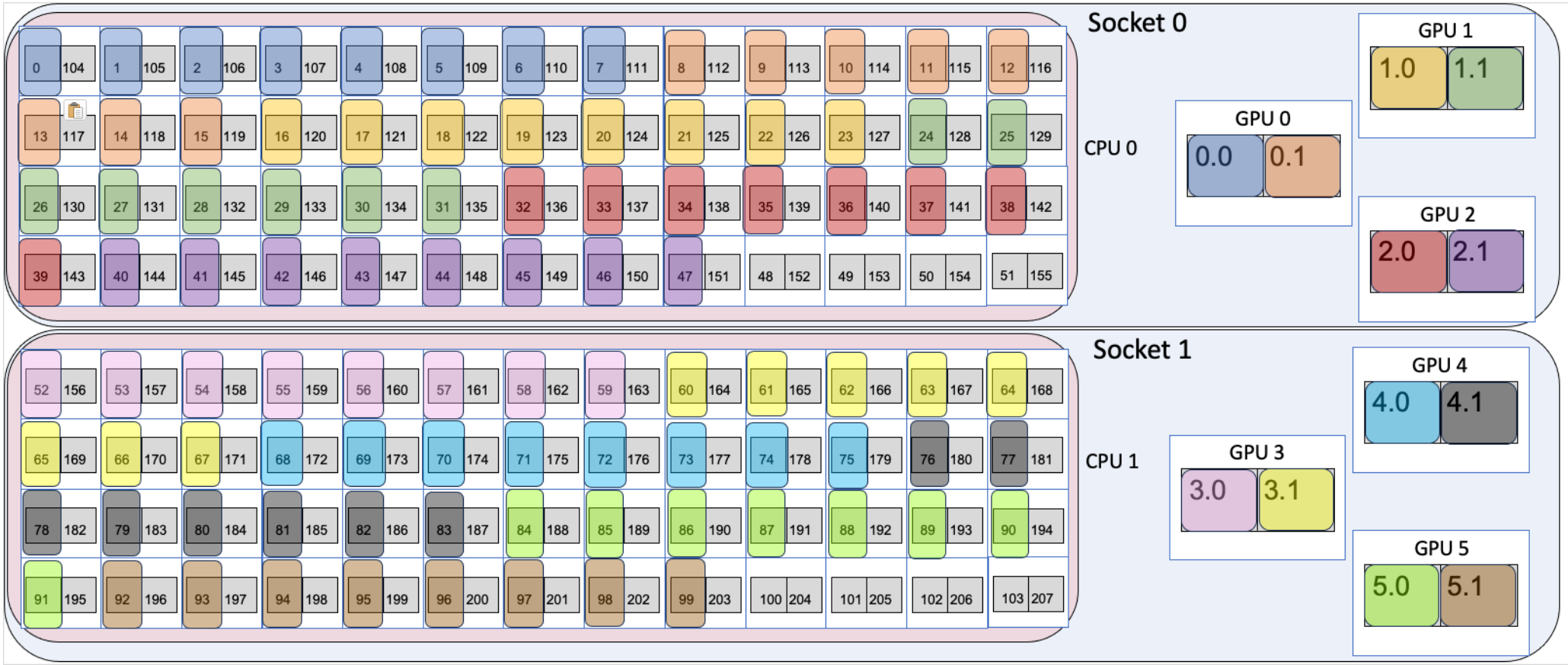
Intel® Xeon Max Series CPU with HBM





Argonne NATIONAL LABORATORY

PHYSICAL HARDWARE



This image shows a deeper dive into the physical hardware from the perspective of how an application might see the compute node. Though not quite correct, we can think of the compute blade as consisting of two sockets, each having a 52-core CPU and 3 GPUs. Each CPU core supports 2 hyperthreads. The GPUs physically consist of two tiles with a fast interconnect and many applications may be benefit by binding processes to individual tiles as indicated by the color assignments (one of many possibilities).

LOGGING IN

ssh <username>@aurora.alcf.anl.gov

You will be prompted for your password, which is a six digit code generated uniquely each time using the MobilePASS+ app or a physical token (if you have one).

<username>@aurora-uan-0012:~>

FILESYSTEM

/home/<username>

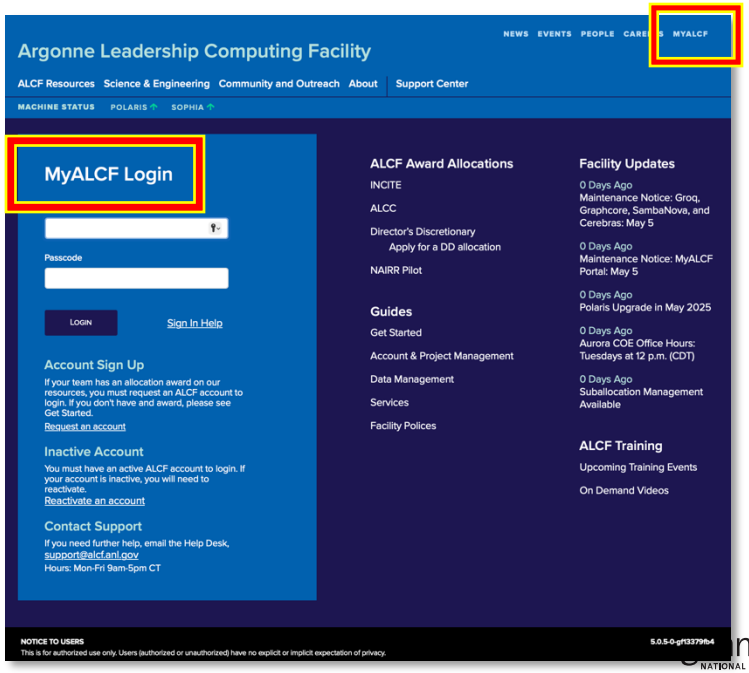
/lus/flare/projects/<project-name>

Users should use project spaces for large scale storage and software installations. Increases can be requested via support@alcf.anl.gov.

TIPS

myprojectquotas

MyALCF



RESERVATION QUEUE MAY 6-16

NOTE

Polaris will be down for maintenance for parts in May.

GRONK: https://status.alcf.anl.gov/#/polaris

gpu_hack_prio:
Special priority queue up to 256 nodes, 2 node max size

E.g. qsub -I -l select=1 -l walltime=00:60:00 -l filesystems=home:flare -A **gpu_hack** -q **gpu_hack_prio**

```
mgarcia@aurora-uan-0009:~/gpu_hack/examples> qsub -I -l select=1 -l walltime=00:60:00 -l filesystems=home:flare -A gpu_hack -q gpu_hack_prio
qsub: waiting for job 4673776.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov to start
qsub: job 4673776.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov ready

mgarcia@x4516c2s0b0n0:~>
```

GETTING TO KNOW THE ENVIRONMENT

ALCF uses Environment Modules to provide users with loadable software packages. This includes compilers, python installations, and other software. Here are some basic commands:

```
module list
```

```
module avail
```

By default, MODULEPATH only includes system libraries from Intel/HPE. One can include pre-built modules from ALCF staff by adding the path /soft/modulefiles to MODULEFILE using either of these commands:

```
export MODULEPATH=$MODULEPATH:/soft/modulefiles
# OR
module use /soft/modulefiles
```

Loading modules

```
module load cmake
```

Using Spack

Spack is an HPC oriented build management system. In this case of this quick introduction, Spack is simply used to offer additional pre-compiled software.
On Aurora, these additional spack packages are made available by default from the /soft/modulefiles area:

```
module use /soft/modulefiles
```

OUTPUT

```
mgarcia@x4516c2s0b0n0:~> module list

Currently Loaded Modules:
  1) gcc-runtime/13.3.0-ghotoln (H)   7) libiconv/1.17-jjpb4sl         (H)  13) cray-pals/1.4.0
  2) gmp/6.3.0-mtokfaw          (H)   8) libxml2/2.13.5                     14) cray-libpals/1.4.0
  3) mpfr/4.2.1-gkcdl5w         (H)   9) hwloc/2.11.3-mpich-level-zero      15) xpu-smi/1.2.39
  4) mpc/1.3.1-rdrlvsl          (H)  10) yaksa/0.3-7ks5f26             (H)  16) forge/24.1.2
  5) gcc/13.3.0                      11) mpich/opt/develop-git.6037a7a
  6) oneapi/release/2025.0.5         12) libfabric/1.22.0

  Where:
   H:  Hidden Module
```

```
mgarcia@x4516c2s0b0n0:~  module load cmake
mgarcia@x4516c2s0b0n0:~  module list

Currently Loaded Modules:
  1) gcc-runtime/13.3.0-ghotoln (H)   7) libiconv/1.17-jjpb4sl         (H)  13) cray-pals/1.4.0
  2) gmp/6.3.0-mtokfaw          (H)   8) libxml2/2.13.5                     14) cray-libpals/1.4.0
  3) mpfr/4.2.1-gkcdl5w         (H)   9) hwloc/2.11.3-mpich-level-zero      15) xpu-smi/1.2.39
  4) mpc/1.3.1-rdrlvsl          (H)  10) yaksa/0.3-7ks5f26             (H)  16) forge/24.1.2
  5) gcc/13.3.0                      11) mpich/opt/develop-git.6037a7a      17) gmake/4.4.1
  6) oneapi/release/2025.0.5         12) libfabric/1.22.0                   18) cmake/3.30.5

  Where:
   H:  Hidden Module
```

```
mgarcia@x4516c2s0b0n0:~> module avail

------------------------------------------ /soft/modulefiles ------------------------------------------
   alcf-reframe/alcf-reframe                      daos/base
   ascent/develop/2025-03-19-c1f63e7-openmp       daos_ops/base_old_pre_DAOS_15236_advice
   ascent/develop/2025-03-19-c1f63e7-sycl   (D)   daos_ops/base                              (D)
   bbfft/2022.12.30.003/eng-compiler/bbfft        daos_perf/base
   chipStar/1.2.1                                 daos_real_user/base
   chipStar/latest-math                           headers/cuda/12.0.0
   chipStar/latest-static                         jax/0.4.4
   chipStar/testing                               jax/0.4.25                                 (D)
   codee/2024.4.5                                 libraries/libdrm-devel/2.4.104-1.12
   codee/2025.1                                   paraview/paraview-5.13.2
   codee/2025.1.2                                 tau/modulepath
   codee/2025.1.3                                 visit/visit-3.4.2
   codee/2025.2                             (D)

---- /opt/aurora/24.347.0/spack/unified/0.9.2/install/modulefiles/mpich/develop-git.6037a7a-sxnhr7p/oneapi/2025.0.5 ----
   adios/1.13.1                   hdf5-vol-async/1.7        parallel-netcdf/1.12.3
   adios2/2.10.2-cpu              hdf5/1.14.5               petsc/3.21.4-cpu
   adios2/2.10.2-sycl       (D)   heffte/2.4.1-cpu          pumi/2.2.9
   amrex/24.11-sycl               hypre/2.33.0-sycl         py-mpi4py/4.0.1
   boost/1.84.0                   launchmon/1.2.0           spindle/0.13
   cabana/0.7.0-omp-sycl          mpifileutils/0.11.1       stat/develop-git.5aa0d93
   copper/main                    netcdf-c/4.9.2            superlu-dist/9.1.0
   darshan-runtime/3.4.6          netcdf-cxx4/4.3.1         umpire/2024.07.0-omp
   fftw/3.3.10                    netcdf-fortran/4.6.1      valgrind/3.24.0
   geopm-runtime/3.1.0-omp        netlib-scalapack/2.2.0

---------------------- /opt/aurora/24.347.0/spack/unified/0.9.2/install/modulefiles/Core ----------------------
lines 1-29
```

USING THE AURORA JOB SCHEDULER: PBS

https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/aurora/00_scheduler.md

Aurora uses the PBS scheduler similar to other ALCF systems, such as Polaris. PBS is a third party product that comes with extensive documentation. This is an introduction, not an extensive tutorial so we will only cover some basics.

Running interactively

qsub -I -l select=1 -l walltime=00:60:00 -l filesystems=home:flare -A **gpu_hack** -q **gpu_hack_prio**

module load xpu-smi

xpu-smi discovery

Some tests

mkdir /lus/flare/projects/<your_project_name>/gpu_hack/

cp –r /lus/flare/projects/gpu_hack/alcf_training/examples/ .

```
mgarcia@x4516c2s0b0n0:~> ls /lus/flare/projects/gpu_hack/alcf_training/examples/
00_hello_world.sh              01_example_openmp.sh          02_tools_example  HelperScripts
01_example.cpp                 01_example_sycl_affinity.sh   04_AI_frameworks  logs
01_example_openmp_affinity.sh  01_example_sycl.cpp           copper_example
01_example_openmp.cpp          01_example_sycl.sh            daos_example
```

chmod u+x 00_hello_world.sh

USING THE AURORA JOB SCHEDULER: PBS

Submit your first job

The more standard method for running a job is to submit it to the scheduler via qsub with a script that will execute your job without you needing to login to the worker nodes. Let's walk through an example.

First we need to create a job script (example: examples/00_hello_world.sh):

```
#!/bin/bash -l
#PBS -l select=1
#PBS -l walltime=00:30:00
#PBS -q debug
#PBS -l filesystems=home:flare
#PBS -A <project-name>
#PBS -o logs/
#PBS -e logs/

GPUS_PER_NODE=6

mpiexec -n $GPUS_PER_NODE -ppn $GPUS_PER_NODE echo Hello World
```

```
chmod u+x job_script.sh

qsub job_script.sh
```

You'll notice we can use the #PBS line prefix at the top of our script to set qsub command line options. We can still use the command line to override the options in the script.

Here we used -o logs/ and -e logs/ which just redirects the STDOUT(-o) and the STDERR(-e) log files from the job into the logs/ directory to keep things tidy. The logs directory must exist before the job is submitted.

USING THE AURORA JOB SCHEDULER: PBS

Monitor your job

```
qstat –u <username>
```

Without specifying the username we will get a full print out of every job queued and running. This can be overwhelming so using the username reduces the output to jobs for just that username. Adding alias qsme='qstat -u <username>' to your .bashrc is a nice shortcut.

Delete your job

```
qdel <jobID>
```

Job output

Any job STDOUT or STDERR output will go into two different files that by default are named:

```
<script_name>.o<pbs-job-id>
<script_name>.e<pbs-job-id>
```

In our example submit script, we specify -o logs/ and -e logs/ so that the files go into the logs/ directory. In that case, the output files are named differently:

```
logs/${PBS_JOBID}.ER
logs/${PBS_JOBID}.OU
```

OUTPUT

```
mgarcia@x4516c2s0b0n0:~> xpu-smi discovery
+-----------+--------------------------------------------------------------------------------------+
| Device ID | Device Information                                                                   |
+-----------+--------------------------------------------------------------------------------------+
| 0         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-e1c3-1c0d8a8e9392                                       |
|           | PCI BDF Address: 0000:18:00.0                                                        |
|           | DRM Device: /dev/dri/card0                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 1         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-46e0-d0969a8e940e                                       |
|           | PCI BDF Address: 0000:42:00.0                                                        |
|           | DRM Device: /dev/dri/card1                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 2         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-df0a-686d0813f2f8                                       |
|           | PCI BDF Address: 0000:6c:00.0                                                        |
|           | DRM Device: /dev/dri/card2                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 3         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-60b5-7aa827b18071                                       |
|           | PCI BDF Address: 0001:18:00.0                                                        |
|           | DRM Device: /dev/dri/card3                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 4         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-b684-572f3c35f00f                                       |
|           | PCI BDF Address: 0001:42:00.0                                                        |
|           | DRM Device: /dev/dri/card4                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
| 5         | Device Name: Intel(R) Data Center GPU Max 1550                                       |
|           | Vendor Name: Intel(R) Corporation                                                    |
|           | SOC UUID: 00000000-0000-0000-3695-0a03c5be4597                                       |
|           | PCI BDF Address: 0001:6c:00.0                                                        |
|           | DRM Device: /dev/dri/card5                                                           |
|           | Function Type: physical                                                              |
+-----------+--------------------------------------------------------------------------------------+
```

Submit a job

```
mgarcia@x4516c2s0b0n0:~/gpu_hack/My_Tests> qsub -A gpu_hack -q gpu_hack_prio job_script.sh
4673810.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov
mgarcia@x4516c2s0b0n0:~/gpu_hack/My_Tests> qstat -u mgarcia

aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov:
                                                                  Req'd  Req'd   Elap
Job ID                  Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
4673776.aurora-pbs-* mgarcia  gpu_hac* STDIN      137497   1 208    --  01:00 R 00:26
4673810.aurora-pbs-* mgarcia  gpu_hac* job_scrip*    --    1 208    --  00:05 Q   --
```

Remember to create the directory logs

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests/logs> ls -ltr
total 4
-rw-r--r-- 1 mgarcia users  0 May  6 09:34 4673810.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov.ER
-rw-r--r-- 1 mgarcia users 72 May  6 09:34 4673810.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov.OU
```

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests/logs> more 4673810.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov.OU
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

PBS CHEATSHEET

User Commands

Command	Description
`qsub`	Submit a job
`qsub -I`	Submit an interactive job
`qstat <jobid>`	Job status
`qstat -Q`	Print Queue information
`qstat -B`	Cluster status
`qstat -x`	Job History
`qstat -f <jobid>`	Job status with all information
`qstat -ans`	Job status with comments and vnode info
`qhold <jobid>`	Hold a job
`qrls <jobid>`	Release a job
`pbsnodes -a`	Print node information
`pbsnodes -l`	Print nodes that are offline or down
`qdel <jobid>`	kill a job
`qdel -W force <jobid>`	Force kill a job
`qmove`	Moves PBS batch job between queues
`qalter`	Alters a PBS job
`pbs_rstat`	Shows status of PBS advance or standing reservations

QSUB Options

Option	Description
`-P project_name`	Specifying a project name
`-q destination`	Specifying queue and/or server
`-r value`	Marking a job as rerunnable or not
`-W depend = list`	Specifying job dependencies
`-W stagein=list stageout=list`	Input/output file staging
`-W sandbox=<value>`	Staging and execution directory: user's home vs. job-specific
`-a date_time`	Deferring execution
`-c interval`	Specifying job checkpoint interval
`-e path`	Specifying path for output and error files
`-h`	Holding a job (delaying execution)
`-J X-Y[:Z}`	Defining job array
`-j join`	Merging output and error files
`-k keep`	Retaining output and error files on execution host
`-l resource_list`	Requesting job resources
`-M user_list`	Setting email recipient list
`-m MailOptions`	Specifying email notification
`-N name`	Specifying a job name
`-o path`	Specifying path for output and error files

PBS CHEATSHEET

Environment Variables

Your job will have access to these environment variables

Option	Description
`PBS_JOBID`	Job identifier given by PBS when the job is submitted. Created upon execution
`PBS_JOBNAME`	Job name given by user. Created upon execution
`PBS_NODEFILE`	The filename containing a list of vnodes assigned to the job.
`PBS_O_WORKDIR`	Absolute path to directory where qsub is run. Value taken from user's submission environment.
`TMPDIR`	Pathname of job's scratch directory
`NCPUS`	Number of threads, defaulting to number of CPUs, on the vnode
`PBS_ARRAY_ID`	Identifier for job arrays. Consists of sequence number.
`PBS_ARRAY_INDEX`	Index number of subjob in job array.
`PBS_JOBDIR`	Pathname of job's staging and execution directory on the primary execution host.

COMPILERS ON AURORA

https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/aurora/01_compilers.md

This section describes how to compile C/C++ code standalone, with SYCL and OpenMP, and with MPI.

Specifically it introduces the Intel software environment for compiling system compatible codes. The same flags apply to Fortran applications as well.

User is assumed to know:

- how to compile and run code
- basic familiarity with MPI
- basic familiarity with SYCL and/or OpenMP

Learning Goals:

- MPI compiler wrappers for oneAPI C/C++/FORTRAN compilers
- How to compile a C++ code
- How to compile a C++ code with SYCL and MPI
- How to compile a C++ code with OpenMP and MPI
- How to control CPU and GPU affinities in job scripts

COMPILING C/C++/FORTRAN CODE

When you first login to Aurora, there will be a default list of loaded modules (see them with module list). This includes the oneAPI suite of compilers, libraries, and tools and Cray MPICH. It is recommend to use the MPI compiler wrappers for building applications:

- mpicc - C compiler (use it like oneAPI icx or GNU gcc)
- mpicxx - C++ compiler (use it like oneAPI icpx or GNU g++)
- mpif90 - Fortran compiler (use it like oneAPI ifx or GNU gfortran)

Next an example C++ code is compiled.

Example code: 01_example.cpp

```cpp
#include <iostream>

int main(void){

   std::cout << "Hello World!\n";
   return 0;
}
```

Build and run on an Aurora login node or worker node

```
mpicxx 01_example.cpp -o 01_example

./01_example
```

This example only uses the CPU. A GPU programming model, such as SYCL, OpenMP, or OpenCL (or HIP) is required to use the GPU.

COMPILING C/C++ WITH OPENMP

Users have the choice when compiling GPU-enabled applications to compile the GPU kernels at link-time or at runtime.

Compiling the kernels while linking the application is referred to **Ahead-Of-Time (AOT)** compilation.
Delaying the compilation of GPU kernels to runtime is referred to as **Just-In-Time (JIT)** compilation.

AOT
- Compile: -fiopenmp -fopenmp-targets=spir64_gen
- Link: -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device pvc".

JIT
- Compile: -fiopenmp -fopenmp-targets=spir64
- Link: -fiopenmp -fopenmp-targets=spir64

Both options are available to users, though we recommend using AOT to reduce overhead of starting the application.
The examples that follow use AOT compilation.

Example code: 01_example_openmp.cpp

```
mpicxx -fiopenmp -fopenmp-targets=spir64_gen -c 01_example_openmp.cpp

mpicxx -o 01_example_openmp -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device pvc"
01_example_openmp.o
```

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> mpicxx -fiopenmp -fopenmp-targets=spir64_gen -c 01_example_openmp.cpp

mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests>
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> mpicxx -o 01_example_openmp -fiopenmp -fopenmp-targets=spir64_gen -Xopenmp-target-backend "-device pvc" 01_example_openmp.o
Compilation from IR - skipping loading of FCL
Build succeeded.
```

COMPILING C/C++ WITH OPENMP

Running the code: 01_example_openmp.cpp

```
mgarcia@aurora-uan-0009:~> which icpx
/opt/aurora/24.347.0/oneapi/compiler/latest/bin/icpx
```

```
mgarcia@aurora-uan-0009:~> icpx --version
Intel(R) oneAPI DPC++/C++ Compiler 2025.0.4 (2025.0.4.20241205)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /opt/aurora/24.347.0/oneapi/compiler/2025.0/bin/compiler
Configuration file: /opt/aurora/24.347.0/oneapi/compiler/2025.0/bin/compiler/../icpx.cfg
```

```
#!/bin/bash -l
#PBS -l select=1
#PBS -l walltime=00:10:00
#PBS -q debug
#PBS -A <project-name>
#PBS -l filesystems=home:flare
#PBS -o logs/
#PBS -e logs/

cd ${PBS_O_WORKDIR}

mpiexec -n 1 --ppn 1 ./01_example_openmp
```

Submit your job: qsub -A gpu_hack -q gpu_hack_prio 01_example_openmp.sh

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> qsub -A gpu_hack -q gpu_hack_prio 01_example_openmp.sh
4673830.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov
```

The output should look like this in the logs/<jobID>.<hostname>.OU file:

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> more logs/4673830.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov.OU
# of devices= 6
Rank 0 on host 6 running on GPU 0!
Using double-precision

Result is CORRECT!! :)
```

COMPILING C/C++ WITH SYCL

Now you can compile your C/C++ with SYCL code. Users again have the choice of JIT or AOT compilation.

AOT
- Compile: --intel -fsycl -fsycl-targets=spir64_gen
- Link: --intel -fsycl -fsycl-targets=spir64_gen -Xsycl-target-backend "-device pvc"

JIT
- Compile: --intel -fsycl -fsycl-targets=spir64
- Link: --intel -fsycl -fsycl-targets=spir64

mpicxx --intel -fsycl -fsycl-targets=spir64_gen -c 01_example_sycl.cpp

mpicxx -o 01_example_sycl --intel -fsycl -fsycl-targets=spir64_gen -Xsycl-target-backend "-device pvc" 01_example_sycl.o

Running the code: 01_example_sycl.cpp

Submit your job: qsub -A gpu_hack -q gpu_hack_prio 01_example_sycl.sh

```
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> mpicxx --intel -fsycl  -fsycl-targets=spir64_gen -c 01_example_sycl.cpp
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> mpicxx -o 01_example_sycl --intel -fsycl  -fsycl-targets=spir64_gen -Xsycl-target-backend "-device pvc" 01_example_sycl.o
Compilation from IR - skipping loading of FCL
Build succeeded.
mgarcia@aurora-uan-0009:~/gpu_hack/My_Tests> qsub -A gpu_hack -q gpu_hack_prio 01_example_sycl.sh
4673854.aurora-pbs-0001.hostmgmt.cm.aurora.alcf.anl.gov
```

CPU AFFINITY AND NUMA DOMAINS

Each Aurora node consists of dual 52-core CPUs, each with 2 hyperthreads.
Output of the lscpu command can be used to quickly identify the CPU ids for cores in the two sockets.

lscpu: identify the CPU ids for cores in the 2 sockets.

```
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          52 bits physical, 57 bits virtual
  Byte Order:             Little Endian
CPU(s):                   208
  On-line CPU(s) list:    0-207
Vendor ID:                GenuineIntel
  Model name:             Intel(R) Xeon(R) CPU Max 9470C
    CPU family:           6
    Model:                143
    Thread(s) per core:   2
    Core(s) per socket:   52
    Socket(s):            2
    Stepping:             8
    Frequency boost:      enabled
    CPU max MHz:          2001.0000
    CPU min MHz:          800.0000
...
NUMA:
  NUMA node(s):           4
  NUMA node0 CPU(s):      0-51,104-155
  NUMA node1 CPU(s):      52-103,156-207
  NUMA node2 CPU(s):
  NUMA node3 CPU(s):
...
```

Looking at the first two NUMA domains, we see that CPU cores 0-51 are in the first socket and CPU cores 52-103 in the second socket. The hyperthreads on each socket are CPU cores 104-155 and 156-207.

numactl -hardware: more information on NUMA domains

```
available: 4 nodes (0-3)
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
node 0 size: 515489 MB
node 0 free: 342653 MB
node 1 cpus: 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 8
node 1 size: 515028 MB
node 1 free: 508138 MB
node 2 cpus:
node 2 size: 65536 MB
node 2 free: 63977 MB
node 3 cpus:
node 3 size: 65536 MB
node 3 free: 64146 MB
node distances:
node   0   1   2   3
  0:  10  21  13  23
  1:  21  10  23  13
  2:  13  23  10  23
  3:  23  13  23  10
```

As a means to quickly get started, one could opt for naively binding MPI ranks and processes to the CPU cores using depth logic whereby each MPI rank is assigned to a consecutive set of CPU cores.

For an application running 6 MPI ranks per node (1 per GPU) and each with 4 OpenMP threads on the host, one could set the depth as 4 (or something larger).

```
mpiexec -n 6 --ppn 6 --depth=4 --cpu-bind depth --env
OMP_NUM_THREADS=4 ...
```

CPU AFFINITY AND NUMA DOMAINS

mpiexec -n 6 --ppn 6 --cpu-bind=list:~~0~~-3,4-7,8-11,~~52~~-55,56-59,60-63 --env OMP_NUM_THREADS=4 …

1 53

[Aurora-notify] Upcoming change: March 31, 2025

Aurora-notify <aurora-notify-bounces@lists.alcf.anl.gov> on behalf of ALCF Support <support@alcf.anl.gov> Tuesday, March 25, 2025 at 12:57 PM

To: aurora-notify@alcf.anl.gov

ATT00001.txt
0.2 KB

Download All • Preview All

Dear ALCF users,

Set to become effective during the planned maintenance scheduled for March 31, 2025, an upcoming change will disable user applications from running on cores 0, 52, 104, and 156.

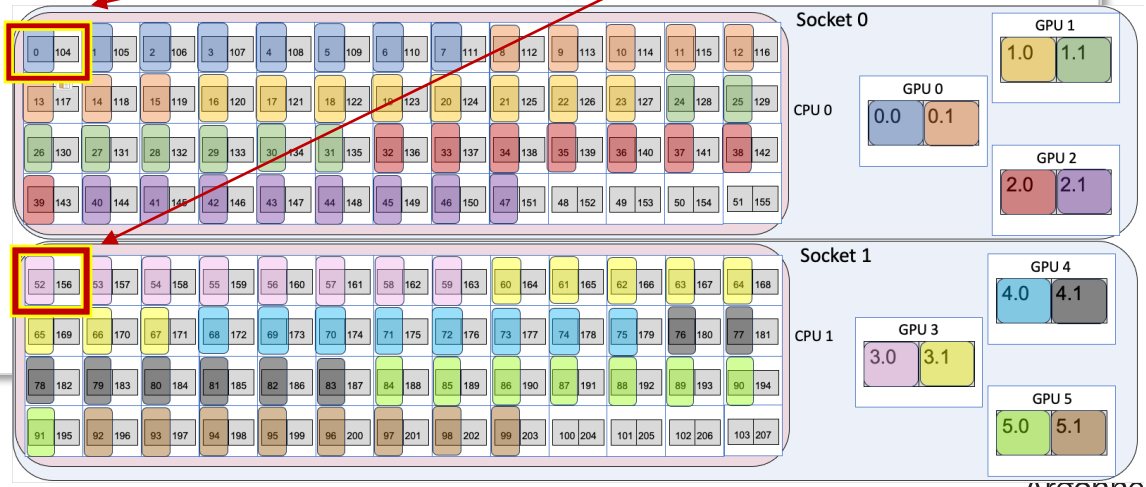
These are the first physical cores on each CPU socket; they will be reserved for system services on each compute node. Testing applications on these cores has shown slower performance.

For example, after the upcoming change, an application binding to cores 0 and 52 using mpiexec will display the below error and exit.

```
~> mpiexec -n 2 --ppn 2 --cpu-bind list:0,0:1,52 hostname
launch failed on x4000c1s0b0n0: cpubind list:0,0:1,52[0] has fewer CPUs (0) than depth (1)
Via taskset in an interactive job
~> taskset --cpu-list 0 bash -c 'exit' ; echo $?
taskset: failed to set pid 100207's affinity: Invalid argument
1
```

Please contact support@alcf.anl.gov with any questions or concerns.

Regards,
ALCF Support



GPU AFFINITY

Similar to the fare degree of flexibility in how one binds software processes to the CPU hardware, one can also bind processes to the GPU hardware at different levels. The default is that each of the 6 GPUs is viewed as a single device. Each Aurora GPU consists of two physical tiles and each can be targeted individually by applications. In other words, a typical configuration for an application may be to spawn 12 MPI ranks per compute node with each MPI rank bound to a single GPU tile. Furthermore, each GPU tile can be targeted in a more granular fashion to bind MPI ranks to individual **Compute Command Streamers** (CCSs). The latter may prove beneficial when an application has considerable work on the CPUs that warrants additional parallelism.

A set of helper scripts are provided which accept the local MPI rank ID as input and assigns the appropriate GPU hardware in a round-robin fashion.

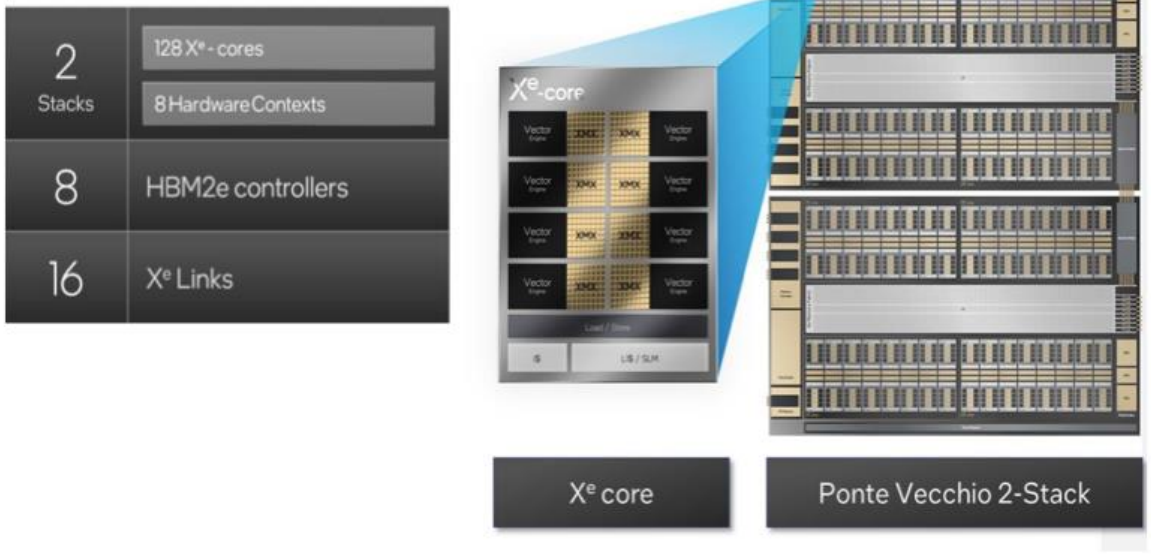
./examples/HelperScripts

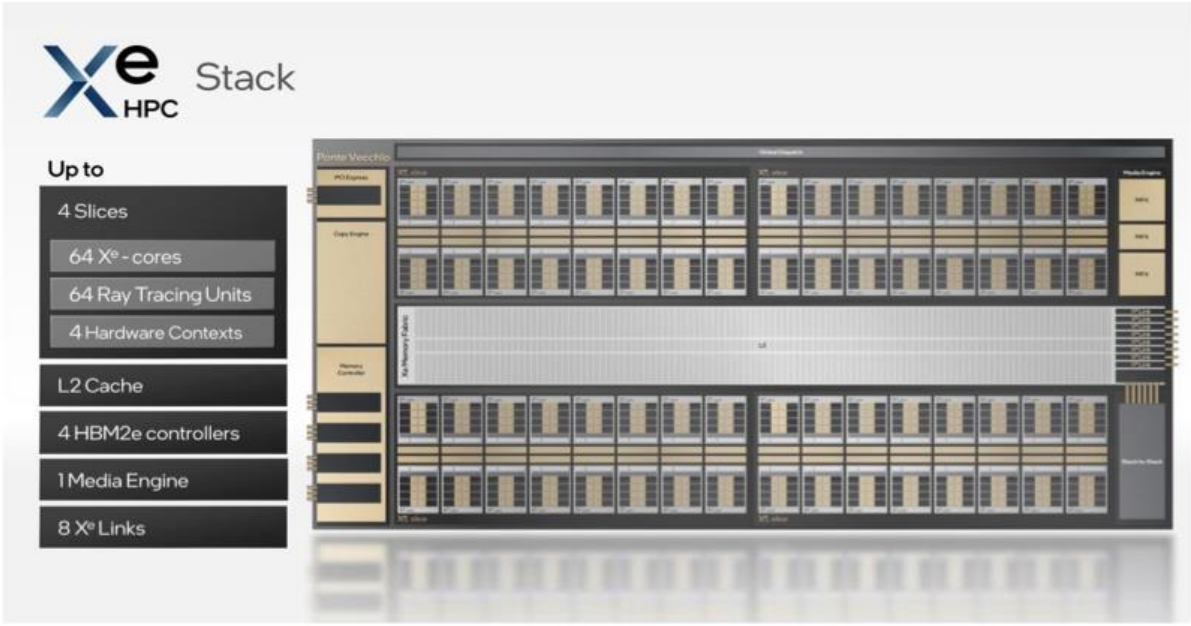
- set_affinity_gpu.sh: bind MPI ranks to GPU tile
  - useful for when running at least 2 MPI ranks per PVC GPU (i.e. 12 MPI ranks per node)
- set_affinity_gpu_2ccs.sh: bind MPI ranks to 1/2 GPU tile
  - useful for when running at least 4 MPI ranks per PVC GPU (i.e. 24 MPI ranks per node)
- set affinity_gpu_4ccs.sh: bind MPI ranks to 1/4 GPU tile
  - useful for when running at least 8 MPI ranks per PVC GPU (i.e. 48 or 96 MPI ranks per node)

Example submission scripts: 01_example_openmp_affinity.sh & 01_example_sycl_affinity.sh

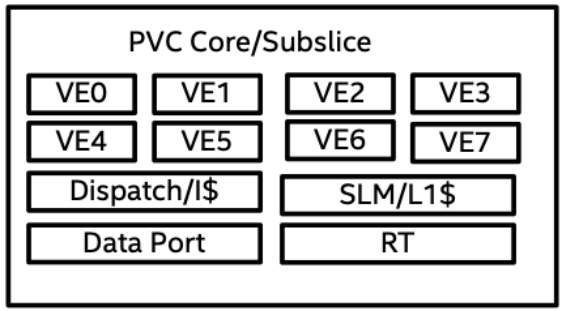
GPU AFFINITY

https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-2/intel-xe-gpu-architecture.html





- Vector Engines execute SIMD math & load/store
- Each Vector Engine services multiple HW threads, issuing one thread instruction per clock tick
- Multiple Vector Engines form a Core, sharing one memory load/store unit
- Multiple Cores form a Slice
- Four Slices form a Stack
- Two Stacks form a PVC



EU = Vector Engine

Sub-slice = Core

Slice = Slice

Tile = Stack

Total Threads = #_slices * #_cores_per_slice * #_ve_per_core * #_threads_per_ve

(3,584 = 4 * 14 * 8 * 8)

CHOOSE YOUR OWN ADVENTURE

AURORA

TOP #3

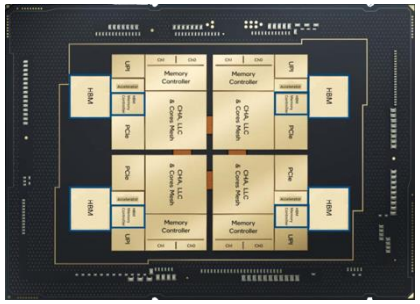
ALCF Aurora

Intel CPU / Intel GPU

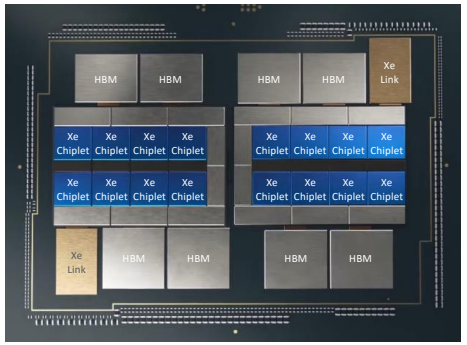
SYCL/DPC++

Xe-core

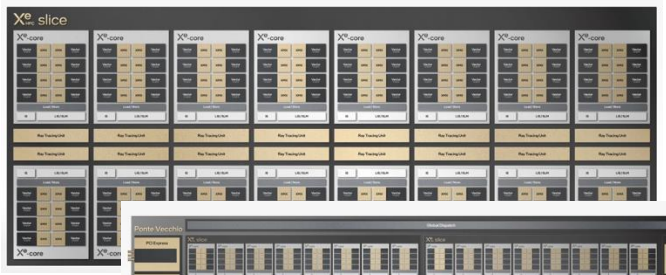
Intel® Xeon Max Series CPU with HBM



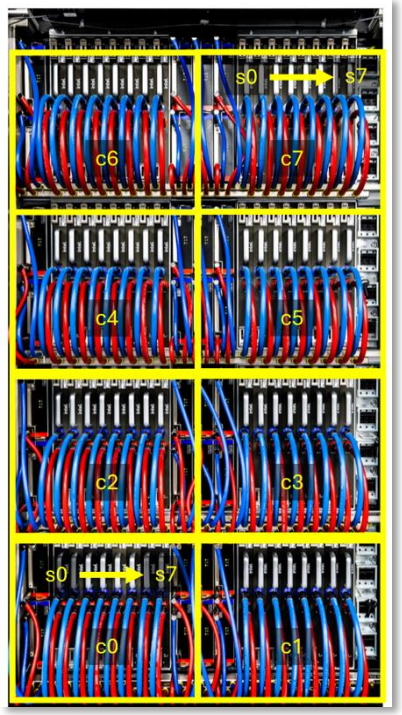
Intel® Data Center GPU Max



Xe-slice

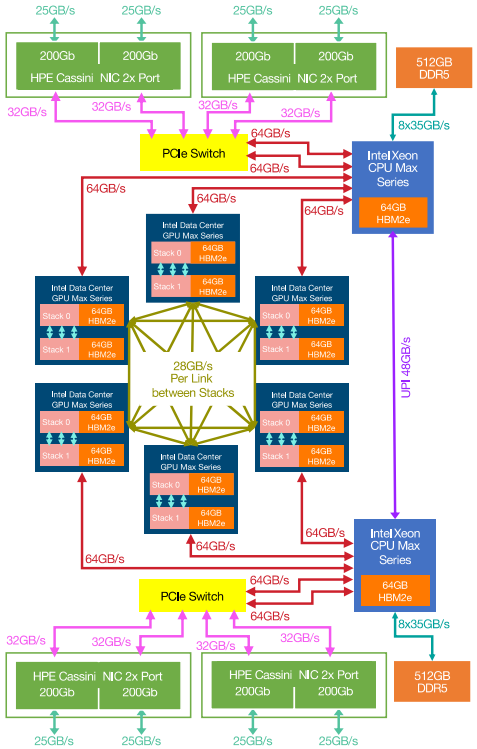


Xe- stack

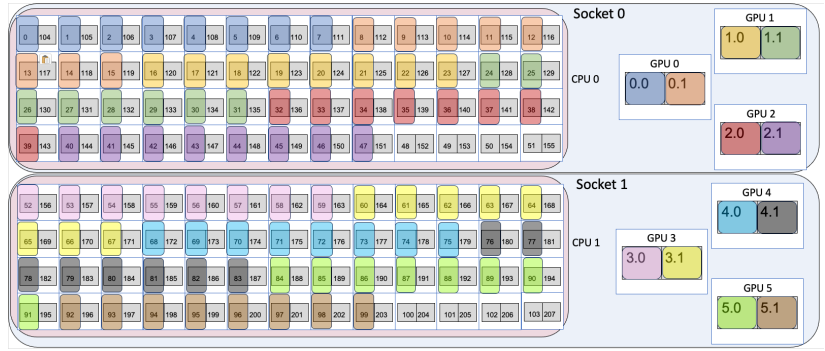


x4XXXc{0-7}s{0-7}b0n0

Node Characteristics



Representation of Physical Hardware



Argonne NATIONAL LABORATORY