Cabeus includes 538 GPU nodes based on the NVIDIA GH200 Grace Hopper Superchip.

To help you prepare for running jobs on the Grace Hopper GPU nodes, this short user guide includes information on the general configuration of the nodes, compiling your code, and running PBS jobs.

Overview of Cabeus Nodes with NVIDIA Grace Hopper Superchip

The nodes are accessed via two front ends:

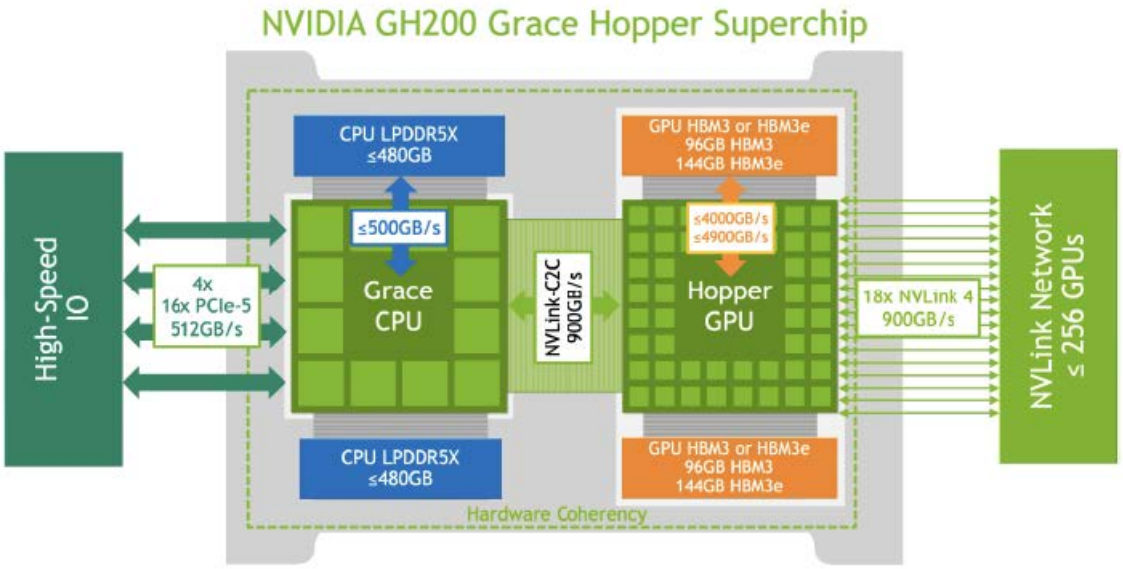
- cghfe01
- cghfe02

The hostnames for the GH200 compute nodes are as follows:

- cgh[1-3,6-15,17-19,21-23]n[01-26]
- cgh[04,20]n[01-12]
- cgh16n[01-20]

The PBS resource model name is `gh200`.

Each of the front ends and compute nodes contains a single GH200 superchip, which is shown in this diagram published by NVIDIA:



As shown in the diagram, the GH200 superchip includes:

- One Grace CPU containing 72 Arm Neoverse V2 physical cores (maximum 3.465 GHz on the front ends; slight variation on compute nodes but should be all above 3.4 GHz; use the `lscpu` command to check) and up to 480 GB Low-Power Double Data Rate 5X (LPDDR5X) host memory with a memory bandwidth of up to 500 GB/s.

  Note: The memory bandwidth varies with memory capacity: 384 GB/s with a capacity of 480 GB as in all NAS GH200 nodes, and 512 GB/s with a capacity of 120 GB or 240 GB.

  The Arm Neoverse V2 CPU uses the Armv9.0-A architecture which comes with four 128-bit Single Instruction Multiple Data (SIMD) units that support Scalable Vector Extension 2 (SVE2) instructions.
- One H100 (Hopper) GPU containing 132 Streaming Multiprocessors (SMs) with a clock rate of 1.980 GHz and 96 GB High Bandwidth Memory 3 (HBM3) device memory with a memory bandwidth of up to 4000 GB/s. Each SM contains 128 single precision CUDA cores (total is 16,896 per H100 GPU), 64 double precision CUDA cores (total is 8,448 per H100 GPU), and 4 Tensor cores (total is 528 per H100 GPU).
- A high-bandwidth, memory-coherent NVIDIA NVLink-C2C (chip-2-chip) interconnect that connects the CPU and GPU with a bidirectional bandwidth of 900 GB/s.

  NVLink-C2C enables applications to oversubscribe the Hopper GPU's memory and directly utilize the Grace CPU's memory at high bandwidth. The Grace CPU can also directly access the GPU memory. The memory coherency enables programming with a unified programming model.
- 4x (4 links) PCIe 5.0 x16 (16 lanes) with a total bi-directional bandwidth of 512 GB/s.

  The GH200 nodes at NAS use the NVIDIA MGX platform design where the nodes are interconnected with PCIe and host channel adapter (HCA) on each node and NVIDIA's internode InfiniBand networking solution. Specifically, InfiniBand Next-Generation Data Rate (NDR) HCAs, switches, and cables are utilized in a fat tree topology.

  Note: The 18x NVLink 4 network shown on the right side of the diagram above does not apply to the NVIDIA MGX platform. It is used in the NVIDIA DGX platform for connecting up to 256 GPUs together.

For more information about the Grace Hopper superchip, see NVIDIA's GH200 architecture whitepaper.

Requesting GPU Allocations

A Standard Billing Unit (SBU) allocation specifically for the GPUs is required to use all GPU resources, including the GH200 compute nodes. The SBU rate for the GH200 nodes is 18.34.

Accessing the Cabeus Grace Hopper Front Ends and Compute Nodes

The GH200 front end nodes (cghfe01 and cghfe02) are for building, testing your software dependencies, and running short tests of your executables.

Note: Termination of long-running CPU and GPU processes may be enforced in the future.

You can access the GH200 front ends from your local desktop using one of the following methods:

One-Step Connection With SSH passthrough

If you normally connect to NAS front-ends using SSH passthrough, you can modify your existing .ssh/config file on your local host to add cghfe01 and cghfe02. If you have been using a configuration file based on the NAS config template, you can simply replace this line:

```
Host *.nas.nasa.gov lou lou? ?fe ?fe? ?fe?? *-last pbspl?
```

with the following line:

```
Host *.nas.nasa.gov lou lou? *fe* *-last pbs*
```

where *fe* matches any NAS front-end system. Then, run:

```
your_localhost% ssh cghfe01 (or ssh cghfe02)
```

Two-Step Connection Without SSH Passthrough

To access the front ends without SSH passthrough, run:

```
your_localhost% ssh sfe6 (or ssh sfe7, ssh sfe8)
sfe% ssh cghfe01  (or ssh cghfe02)
Password: xxxxxxxxxxx
cghfe01%
```

The gh200 compute nodes are for running long testing or production jobs. They are accessible only through PBS jobs submitted from the cghfe01 or cghfe02 front ends. The PBS server for the GH200 nodes is pbs05a (users are not allowed to SSH into pbs05a).

You can access your filesystems from the GH200 front-ends and compute nodes, including: $HOME, Lustre /nobackup, /nobackupnfs1, /nasa, /nasa2, and /vast_swbuild.

Operating Systems

The operating system on all GH200-based front ends and compute nodes is TOSS 5 with Red Hat Enterprise Linux 9.8. The Linux kernel is Release 5.x for the Arm architecture 64-bit execution state (aarch64).

```
% cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="9.8 (Plow)"

% uname -r
5.14.0-687.39.1.1toss.t5.aarch64
```

Software Modules

For AI/ML Applications

If you run AI/ML applications: the miniconda3/gh2 module in the /swbuild/analytix/tools directory is the only one that will work with the GH200 nodes:

```
cghfe01% module use -a /swbuild/analytix/tools/modulefiles
cghfe01% module load miniconda3/gh2
```

For more information, including the conda environments that can be used with the Grace Hopper nodes, see Using Conda Environments on NAS Systems.

The procedure to access JupyterLab on the GH200 system is the same as those outlined in Secure Setup for Using Jupyter Notebook on NAS Systems and Using Jupyter Notebook for Machine Learning Development on NAS Systems, except on the local system the user needs to proxyjump through a Grace Hopper front end. For example:

```
your_local_system% ssh -l username -o "StrictHostKeyChecking ask" \
-L 8080:/tmp/user.sock -o ProxyJump=sfe,cghfe01
```

For HPC Applications

You can find HPC software modules for the Arm-based GH200 nodes in the /nasa/arm directory. The current default $MODULEPATH includes /nasa/arm/modulefiles/toss5, where you can find many of the modules mentioned in this article.

```
cghfe01% echo $MODULEPATH
/etc/scl/modulefiles:/usr/share/Modules/modulefiles:/etc/modulefiles:/usr/share/ \
modulefiles:/nasa/arm/modulefiles/toss5
```

Note: The command line above is too long to be formatted as one line, so it is broken with a backslash (\).

Some of the available software packages are mentioned in the Programming and Compiling for Grace Hopper Nodes and Running a GPU Application sections. Be aware that many nvhpc or nvhpc-xxx modules (packaged and pre-built by NVIDIA) contain multiple components (such as CUDA, compiler, math, MPI, .., and so on.). If you want to use only one of the components, you can try other modules, for example, those with module names such as cuda, hpcx, openmpi, and so on.

WARNING: Software modules and executables built for the x86_64 platform (such as NAS PFEs, AFEs, and CFEs; bro_ele, sky_ele, cas_ait, rom_ait, mil_ait, and mil_a100 GPU nodes) cannot be used on the aarch64 platform (i.e., gh200 GPU nodes) because x86_64 and aarch64 architectures have different instruction sets. You must use modules under the /nasa/arm directory to recompile and run your applications for the Arm-based GH200 systems.

If your system startup script loads modules meant for use on x86_64 based systems (for example, the .bashrc, .profile, .cshrc, or .login scripts, which are accessible from all NAS x86_64 and Arm systems), you should unload those modules when you are on the GH200 front-ends and compute nodes.

Programming and Compiling for Grace Hopper Nodes

In addition to the information in this section, read Programming for NVIDIA GPUs and the references listed there for information on the recommended programming models for developing applications suitable for running on the NVIDIA GPUs. Some of the most commonly used software packages needed with these programming languages are CUDA toolkits, compilers, and math and MPI libraries.

CUDA Toolkit

You can find the CUDA version and its associated driver installed on the GH200 front ends and compute nodes by running the nvidia-smi command. Currently, it is CUDA version 13.3 and driver is 610.57.04.

Since the CUDA driver is backward compatible, an application compiled with an older CUDA Toolkit version can run with a newer driver release. For example, you can build your GPU application with CUDA Toolkit 12.6, and run it on a system with CUDA 13.3. See CUDA Toolkit and CUDA Driver for more information.

The following CUDA Toolkits are available for use on the GH200 nodes:

- **CUDA 12.6**

  ```
  module load nvhpc/25.1 (or module load nvhpc-nompi/25.1)
  nvcc --version
  nvcc: NVIDIA (R) Cuda compiler driver
  Copyright (c) 2005-2024 NVIDIA Corporation
  Built on Wed_Oct_30_00:08:18_PDT_2024
  Cuda compilation tools, release 12.6, V12.6.85
  Build cuda_12.6.r12.6/compiler.35059454_0
  ```

  OR

  ```
  module load nvhpc/24.11 (or module load nvhpc-nompi/24.11)
  nvcc --version
  nvcc: NVIDIA (R) Cuda compiler driver
  Copyright (c) 2005-2024 NVIDIA Corporation
  Built on Thu_Sep_12_02:27:38_PDT_2024
  Cuda compilation tools, release 12.6, V12.6.77
  Build cuda_12.6.r12.6/compiler.34841621_0
  ```

- **CUDA 12.3**

  ```
  module load nvhpc/24.3 (or nvhpc-nompi/24.3 or nvhpc-openmpi3/24.3)
  nvcc --version
  nvcc: NVIDIA (R) Cuda compiler driver
  Copyright (c) 2005-2023 NVIDIA Corporation
  Built on Fri_Nov__3_17:22:02_PDT_2023
  Cuda compilation tools, release 12.3, V12.3.103
  Build cuda_12.3.r12.3/compiler.33492891_0
  ```

Compilers

There are several compiler options for building your applications on the GH200 front ends. With any of these compilers, make sure your application builds, runs successfully, and validates correctly.

NVIDIA Compilers: nvc, nvc++, nvfortran, nvcc

To use an NVIDIA compiler, load one of the nvhpc modulefiles on either the cghfe01 or cghfe02 front end or on the GH200 compute nodes. Below are examples of a few possible choices:

- For the 25.1 version of the NVIDIA compilers, use:

  ```
  % module load nvhpc/25.1
  or
  % module load nvhpc-nompi/25.1
  ```

- For the 24.11 version of the NVIDIA compilers, use:

  ```
  % module load nvhpc/24.11
  or
  % module load nvhpc-nompi/24.11
  ```

- For the 24.3 version of the NVIDIA compilers, use:

  ```
  % module load nvhpc/24.3
  or
  % module load nvhpc-nompi/24.3
  ```

The NVIDIA compilers include the HPC compilers and the CUDA C/C++ compiler:

- NVIDIA HPC Compilers: nvc, nvc++, nvfortran

  These compilers can be used to compile pure CPU applications or GPU applications written in C/C++ and Fortran language standards (through the -stdpar[=gpu,multicore] flag); OpenACC (-acc[=gpu,multicore] flag); OpenMP (-mp[=gpu,multicore] flag), and CUDA Fortran. The gpu and multicore options let you choose whether to compile the application to run on the GPUs or on the multicore CPUs.

  Use the -tp=native flag or the -mcpu=native flag when compilation and execution occur on the same CPU processor (in this case, the Grace CPU) and use the -gpu=cc90 flag (i.e., compute capability 9.0) to target the Hopper GPU.

- NVIDIA CUDA C/C++ Compiler: nvcc

  The nvcc compiler is used to build CUDA C/C++ applications to run on the GPUs. This compiler splits the source code into *device code* and *host code*. The device code is compiled by nvcc. nvcc sends the host code to a C/C++ compiler on the host for compilation—for example, the gcc and g++ GNU compilers in the /usr/bin directory. You can choose a different host compiler—for example, a different version of GNU compiler—through the nvcc flag -ccbin=/*path_to_compiler* or --compiler-bindir=/*path_to_compiler*.

For more information, see Compiling GPU Applications for Different Generations of NVIDIA GPUs at NAS.

GNU Compiler Collection: gcc, g++, gfortran

- GNU compilers at the system location:

  ```
  cghfe01% which gcc
  /usr/bin/gcc
  cghfe01% gcc --version
  gcc (GCC) 11.5.0 20240719 (Red Hat 11.5.0-14)

  cghfe01% which gfortran
  /usr/bin/gfortran
  cghfe01% gfortran --version
  GNU Fortran (GCC) 11.5.0 20240719 (Red Hat 11.5.0-14)
  ```

  These compilers can only be used for building CPU-only applications, as they were not built with offloading support.

For all of the above compilers, NVIDIA recommends:

- Use the -mcpu=native flag instead of the -march=xxx or -mtune=xxx flags to optimize for the native CPU and use its features.
- Depending on the char sign type you assume for your application. you may need to use one of these C language flags: -fsigned-char, -funsigned-char, or -fnosigned-char.
  Note: For NVIDIA HPC compilers (nvc, nvc++, nvfortran), the default char sign type is -fsigned-char on x86_64 systems and -funsigned-char on aarch64 systems. For the Arm Compiler for Linux, the default is -fno-signed-char.

For the GNU compiler and/or Arm compiler:

- If using -Ofast results in unacceptable fast math optimizations, use -O3 -ffp-contract=fast. For even more accuracy, use -ffp-contract=off to disable floating point operation contraction used in fused multiply-add.
- Use -flto to enable link-time optimization, which can be significant.

For the GNU compiler, you can use -fsanitize=address -fsanitize=undefined to help debugging memory issues that might appear on

aarch64 systems (but are hidden on x86 systems due to differences on how compiler generate codes and layout data).

Math Libraries

There are two sets of math libraries. One targets the GPUs and the other targets the Grace CPUs.

- **CUDA libraries targeting the Hopper GPUs:**
  This set of libraries includes: cuBLAS, cuFFT, cuRAND, cuSOLVER, cuSPARSE, and cuTENSOR. They are located in the following directories:
  - If using module nvhpc/25.1:
    /nasa/arm/software/nvhpc/25.1/Linux_aarch64/25.1/math_libs/lib64
  - If using module nvhpc/24.11: /nasa/arm/software/nvhpc/24.11/Linux_aarch64/24.11/math_libs/lib64
  - If using module nvhpc/24.3: /nasa/arm/software/nvhpc/24.3/Linux_aarch64/24.3/math_libs/lib64

  When using the NVIDIA HPC compilers (nvc, nvc++, and nvfortran), these libraries can be linked with the flag -cudalib[=cublas|cufft|...].

- **NVIDIA Performance Libraries (NVPL) targeting the Grace CPUs:**
  NVPL is a set of optimized math libraries (BLAS, FFT, LAPACK, RAND, SCALAPACK, SPARSE, and TENSOR) that are used for the Grace CPUs. The NVPL set is available with NVIDIA HPC SDK version 24.3 (and later) in the /nasa/arm/.../math_libs/nvpl/libs directory.
  When using the NVIDIA HPC compilers (nvc, nvc++, and nvfortran), these libraries (with the exception of SCALAPCK) can be linked with the flag -Mnvpl[=blas,fft,...]. To link with NVPL SCALAPACK, use both -Mscalapack and -Mnvpl.
  If using the GNU compiler for linking with NVPL, use the -I, -L, and -lnvpl_.... options explicitly.

MPI Libraries

The GH200 nodes are not supported by HPE, and therefore HPE MPT library is not available on these nodes.

There are many modules containing OpenMPI. If you are not sure which one to use, start with one of these OpenMPI modules. Read Using OpenMPI on NAS Systems to learn how to use OpenMPI.

OpenMPI Version 4.1.7

- nvhpc/25.1: provided by NVIDIA, and built with gcc, g++, nvfortran, HPCX 2.21, and CUDA 12.6.
- nvhpc/24.11: provided by NVIDIA, and built with gcc, g++, nvfortran, HPCX 2.20, and CUDA 12.5.
- nvhpc/24.3: provided by NVIDIA, and built with gcc, g++, nvfortran, HPCX 2.17.1, and CUDA 12.2.
- hpcx/2.21: open source version of precompiled HPCX. The OpenMPI provided in this distribution was built with gcc, g++, gfortran, HPCX 2.21, and CUDA 12.6.
- openmpi/4.1.7-cuda12.6-gnu: built by NAS staff with GNU compiler, UCX 1.17, and CUDA 12.6.
  Loading openmpi/4.1.7-cuda12.6-gnu automatically loads nvhpc-nompi/24.11, which provides access to CUDA 12.6.
- openmpi/4.1.7-cuda12.6-nv: built by NAS staff with nvc, nvc++, nvfortran, UCX 1.17, and CUDA 12.6.
  Loading openmpi/4.1.7-cuda12.6-nv automatically loads nvhpc-nompi/24.11 which provides access to NVIDIA HPC compilers and CUDA 12.6.

OpenMPI Version 3.1.5

- nvhpc-openmpi3/24.3: provided by NVIDIA, and built with nvc, nvc++, nvfortran, CUDA 11.0, and without UCX.

TIP: When using newer versions of OpenMPI with UCX, for example, from nvhpc/24.11 or hpcx/2.21, setting the UCX_PROTO_ENABLE=n and UCX_GVA_ENABLE=n environment variables may help to reduce run-to-run variations or degraded performance in MPI communication. Set them as follows:

```
mpiexec -np .. -x UCX_PROTO_ENABLE=n -x UCX_GVA_ENABLE=n ...
```

For applications that perform UCX atomic operations on the Grace CPU, you may also need to add:

```
UCX_IB_MLX5_DEVX=no
```

See NVIDIA HPCX Known Issues (3606445, 3884209, 4026461 and 4050321) for more information.

Running a GPU Application

PBS jobs running on the GH200 nodes are served by a dedicated PBS server, pbs05a, running with PBS version 2024.1.4.

A GH200 compute node (72 CPU cores; 1 GPU; all host and device memory) cannot be divided or shared between PBS jobs or between different chunks of a PBS job (for example, -lselect=*N* means *N* chunks). The GH200 node model name is gh200. In your PBS resource request, you must use :model=gh200. You do not need to use -lplace=scatter:excl.

To run on the GH200 nodes, submit your PBS interactive or batch job from one of the two Grace Hopper front ends, cghfe01 or cghfe02. Use one of the following queues:

- gpu_normal: default priority (0), maximum walltime 24 hours.
- gpu_devel: priority 10, maximum walltime 2 hours, number of nodes 2.
- gpu_wide: priority 20, maximum walltime 12 hours (wide is considered 85+ nodes requested for the job).

- gpu_long: default priority (0); maximum walltime 120 hours; number of nodes 8; number of running jobs 2.

Note: The available queues will change from time to time. Use the command qstat -Q and qstat -fQ to find what queues have been created and enabled to be used by your group.

See the following sample PBS script.

Sample PBS Script for Running Jobs on Grace Hopper Nodes

```
#!/bin/bash
#PBS -lselect=2:ncpus=72:ngpus=1:model=gh200
#PBS -q name_of_queue

module load nvhpc/24.11

# for running 144 MPI ranks on the CPU cores across two nodes
# with 72 MPI ranks per node
# if necessary, also add -x LD_LIBRARY_PATH
mpiexec -np 144 -N 72 --machinefile $PBS_NODEFILE ./a_CPU_executable

# for running 2 MPI ranks on two GPUs across two nodes
# with 1 MPI rank per node
# if necessary, also add -x LD_LIBRARY_PATH
mpiexec -np 2 -N 1 --machinefile $PBS_NODEFILE ./a_GPU_executable
```

After job submission, you should see a JOBID similar to *xxx*.pbs0a5.gh.nas.nasa.gov.

Note: The versions of qstat on newer NAS systems (including Athena and Cabeus GH200 nodes) and older NAS systems (including Electra, Aitken, and Endeavour, and Cabeus V100 and A100 nodes) are different. To get the same look and feel of qstat on the newer systems as the older systems, it is recommended to use a NAS provided script called nas_qstat, available under /usr/local/bin on athfe[01-04] and cghfe[01-02]. In addition, the script showq under /u/hsp/bin can be used to visually see all jobs that are running or pending and their status, along with node availability at the bottom of the output.

References

- NVIDIA Grace Hopper Superchip Architecture White Paper
- NVIDIA Grace Performance Tuning Guide

The Cabeus cluster comprises multiple NVIDIA GPU nodes, including:

- 128 nodes with Milan CPU and NVIDIA A100 GPUs (model=mil_a100)
- 350 nodes with NVIDIA GH200 Superchip (model=gh200)

Among them, the gh200 nodes are served by their own front ends (cghfe01 and cghfe02) and PBS server (pbs05a), while the other GPU nodes are served by the front ends cfe01, cfe02 and PBS server pbspl4.

In this article, the term *Cabeus A100 GPU nodes* refers to the 128 nodes with Milan CPU and NVIDIA A100 GPUs.

To help you prepare for running jobs on the Cabeus A100 GPU nodes, this short user guide includes information on the general configuration of the nodes, compiling your code, and running PBS jobs.

Overview of Cabeus Nodes with AMD EPYC CPU and NVIDIA A100 GPUs

The Cabeus cluster includes two Cabeus front-end nodes (cfe01 and cfe02) for the GPU nodes with x86_64-based CPUs (mil_a100). Each of the CFEs includes one 16-core AMD EPYC 7313p processor and 256 GB memory.

Note: The Cabeus compute nodes are not accessible from the PFEs.

Each of the 128 A100 compute nodes, referred to as the mil_a100 nodes, contains:

- One 64-physical-core AMD EPYC 7763 Milan CPU socket (2.45 GHz) with 512 GB DDR4 host memory.
- Four NVIDIA A100 accelerator cards each with 3,456 double precision CUDA cores, 432 Tensor cores (boost clock 1410 MHz), and 80 GB HBM2e device memory
- Third-generation NVLink with 12 links yielding 600 GB/sec total bandwidth for GPU-to-GPU communication
- PCIe 4.0 bus connecting the CPU socket and the GPU accelerator cards with a bandwidth of 31.5 GB/s per direction

These 128 compute nodes are partitioned into twelve physical racks. They are connected in a fat-tree topology to the Cabeus InfiniBand network (ib0 and ib1) via four-lane High Data Rate (4X HDR) devices and 4X HDR switches. These HDR devices and switches provide a maximum bandwidth of 200 Gbits/s (i.e., 25 GB/s) for internode communication.

All existing HECC filesystems such as the home, VAST, and Lustre filesystems are mounted on both CFEs and on the Cabeus compute nodes.

PBS jobs for running on Cabeus A100 nodes are managed by the PBS server pbspl4.

Requesting GPU Allocations

An SBU allocation specifically for the GPUs is required to use the GPU resources. For more information about requesting GPU allocations, see this article.

The SBU rates for using one GPU node for one hour, for each type of node, are:

- mil_a100 with four A100s: 37.86
- gh200 with one GH200 Superchip: 18.34

Depending on how you request the GPU resources, your job may be charged the full rate or a fraction of the SBU rate. See the next section for some examples.

Accessing the CFEs and the mil_a100 nodes

You can access the Cabeus front ends (CFEs) either from your local desktop or from the PFEs. If you have set upSSH passthrough for the CFEs, then do:

```
your_localhost% ssh cfe01   (or ssh cfe02)
```

If you are connected to a PFE, then do:

```
pfe% ssh cfe01 (or ssh cfe02)
```

You can access the Cabeus Milan A100 compute nodes only through PBS jobs submitted from the CFEs or from pbspl4. To request Milan A100 (mil_a100) resources, specify one of the queue names shown below and the model type mil_a100.

The following PBS attributes are required when requesting the A100 resources:

- **model=mil_a100:** The A100 nodes with the AMD EPYC 7763 CPU are designated with a model name mil_a100.
- **-q**, followed by one of the queues shown below, to access the mil_a100 nodes.

The PBS queues for the Milan A100 nodes are as follows:

- **gpu_normal**
  - Maximum wall time: 24 hrs
  - Maximum node count: 32

- **gpu_devel**
  - Maximum wall time: 2 hrs
  - Maximum node count: 2
  - Maximum user jobs (queued or running): 1
  - Priority: 10

- **np_gpu_wide (non-primetime)**
  - Maximum wall time: 12 hrs
  - Maximum node count: 64
  - Minimum node count: 24
  - Maximum user jobs (queued or running): 2
  - Priority: 20

Important: The **np_gpu_wide** queue will run jobs between 7:00 a.m. Thursdays and 7:00 a.m. Saturdays (Pacific Time). You can submit jobs to the **np_gpu_wide** queue at any time during the week, but the jobs will run only during this period.

As shown in the examples below, adding @pbspl4 to the queue name is optional, whether you submit the job from a CFE or from pbspl4.

```
cfe% qsub -I -q gpu_devel@pbspl4
   -lselect=1:ncpus=10:ngpus=1:mem=xxGB:model=mil_a100
```

or

```
pbspl4% qsub -I -q gpu_devel
   -lselect=1:ncpus=10:ngpus=1:mem=xxGB:model=mil_a100
```

Note: The qsub commands are shown on two lines due to a KB formatting issue. They should be on one line.

When not specified, the default value of ncpus, ngpus, and host mem for all GPU node types (including mil_a100) is 1, 1, and 32 GB, respectively. The minimum value of ncpus, ngpus, and host mem is 1, 1, and 256 MB, respectively.

From the perspective of PBS, one physical node can be made up of multiple virtual nodes (*vnodes*). In the case of mil_a100, one physical node with 4 GPUs is partitioned as 4 vnodes, where each vnode includes 1 GPU, 16 physical cores, and 128 GB of memory.

PBS allocates resources in the vnodes differently depending on the actual selection specified in -lselect=... and the choice of the PBS -lplace statement, which includes an arrangement attribute (free, pack, scatter, vscatter) and a sharing attribute (shared, excl, and exclhost). Read **man qsub** for the definition of each value of the arrangement and the sharing attribute.

If you plan to use fewer than 4 GPUs per physical mil_a100 node, you can use the shared sharing attribute which allows other jobs to use the remaining GPUs in the node. It also allows your job to be charged for only the number of GPUs assigned to your job. For example, the following request will be charged for the use of only 1 GPU in a mil_a100 node:

```
-lselect=1:ncpus=16:ngpus=1:mem=60GB:model=mil_a100 -lplace=scatter:shared
```

If you do not want other jobs to share resources in the mil_a100 nodes assigned to your job, it is easier if you simply request the use of all 4 GPUs in each mil_a100 node. For example:

```
-lselect=1:ncpus=16:ngpus=4:mem=60GB:model=mil_a100 -lplace=scatter:excl
```

This request will result in being charged for all 4 GPUs in a mil_a100 node.

Operating System and Software Modules

The operating system on all A100 compute nodes is TOSS 4 (Red Hat Enterprise Linux-based Tri-Lab Operating System Stack Version 4). Batch jobs use :aoe=toss4 by default, so you do not need to add :aoe=toss4 in your PBS resource request. The default $MODULEPATH includes:

- /usr/share/Modules/modulefiles
- /nasa/modulefiles/toss4
- /nasa/modulefiles/spack
- /nasa/modulefiles/pkgsrc/toss4

Programming and Compiling For Cabeus A100 Nodes

Read Programming for NVIDIA GPUs and the references listed there for information on the recommended programming models for developing applications suitable for running on the NVIDIA GPUs.

Depending on the programming model and language chosen, use an appropriate compiler available in the NVIDIA HPC SDK to compile your application:

- NVIDIA HPC Compilers
  - nvfortran
  - nvc
  - nvc++
- NVIDIA CUDA C/C++ Compiler - nvcc

Run the command module avail nvhpc to find available versions of NVIDIA HPC SDK on the system. The examples shown in this article

use nvhpc/23.7. However, you can choose a different version.

```
% module load nvhpc/23.7
```

or

```
% module load nvhpc-nompi/23.7
```

For building NVIDIA GPU applications with MPI for inter-GPU or inter-node communication, you can use either OpenMPI or HPE MPT:

OpenMPI from nvhpc

Example 1:

```
module load nvhpc/23.7
mpif90 -o your_application.exe -acc=gpu -gpu=cc80
    -Minfo=accel your_application.f
```

Note: The mpif90 line is shown on two lines due to KB formatting issues. It should be a single line.

Example 2:

```
module load nvhpc/23.7
mpicc -o main.o -c main.c
nvcc  -o cuda_code.o -c cuda_code.cu
mpicc -o your_program.exe main.o cuda_code.o -lcudart
```

Note: The default OpenMPI version in nvhpc/23.7 is 3.1.5.

HPE MPT library

This is recommended for building/running applications when using NAS GPU nodes supported by HPE.

Example 1:

```
module load nvhpc-nompi/23.7
module load mpi-hpe/mpt
nvfortran -o your_application.exe \
-acc=gpu -gpu=cc80 -Minfo=accel your_application.f -lmpi
```

Example 2:

```
module load nvhpc-nompi/23.7
module load mpi-hpe/mpt
nvc  -o main.o -c main.c -lmpi
nvcc -o cuda_code.o -c cuda_code.cu
nvc  -o your_program.exe main.o cuda_code.o -lcudart -lmpi
```

Instead of explicitly using nvfortran/nvc/nvc++ ... -lmpi in the command line, you can use the mpif90, mpicc, and mpicxx wrappers from HPE MPT. However, it is better to set the following variables before invoking mpif90, mpicc, and mpicxx to ensure nvfortran/nvc/nvc++ are used instead of gfortran/gcc/g++:

```
setenv MPIF90_F90 nvfotran
mpif90 [options] program.f

setenv MPICC_CC   nvc
mpicc  [options] program.c

setenv MPICXX_CXX nvc++
mpicxx [options] program.cc
```

TIP for Advanced Developers: Instead of using MPI, you can develop applications using the NVIDIA Collective Communications Library (NCCL) for multi-GPU and multi-node communication. Read this NVIDIA page to learn more.

Running a CUDA Application

The following environment variables may be useful during runtime:

- MPI_USE_CUDA: If using HPE MPT library for running a CUDA application across multiple GPUs or across multiple nodes, setting this variable will enable CUDA support so that the buffers on the GPUs can be used with sends, receives, collectives, and MPI RMA calls.
- CUDA_VISIBLE_DEVICES: Use this variable to control which GPUs your application uses. This is useful when you want to target a specific GPU or a set of GPUs for debugging or testing.

Sample PBS Script

```
#PBS -l select=1:ncpus=16:ngpus=4:mem=240GB:model=mil_a100
#PBS -l walltime=8:00:00
## If you request more than 1 chunk (i.e., select=2 or more),
## with resources which are scattered (i.e., not using the same physical nodes)
## and you want exclusive access to the vnodes assigned,
## uncomment the following place statement
##PBS -l place=scatter:excl
#PBS -q gpu_devel

module load nvhpc-nompi/23.7
module load mpi-hpe/mpt

cd $PBS_O_WORKDIR

setenv MPI_USE_CUDA 1

## if you want to run on two specific GPUs,
## for example, GPU0 and GPU2

setenv CUDA_VISIBLE_DEVICES 0,2
mpiexec -np 2 ./your_application.exe

## if you want to run on all four GPUs in the node

setenv CUDA_VISIBLE_DEVICES 0,1,2,3
mpiexec -np 4 ./your_application.exe
```

References

- Cabeus Configuration Details
- PBS Job Sharing on GPU Nodes
- Requesting GPU Resources
- Programming for NVIDIA GPUs

Cabeus, named after a lunar crater in the south polar region of the Moon, is a cluster system that supports HPC and machine learning/artificial intelligence (ML/AI) applications that can benefit from the GPU technology.

GPU Hardware Summary

The node statistics are shown in two groups: nodes based on the x86_64 operating system (OS), and those based on the aarch64 OS.

Cabeus x86_64 GPU Nodes

- 12 racks
- 128 nodes
- 8,192 CPU cores + 1,769,472 double-precision GPU cores
- Total memory: 106 terabytes (65 TB from CPU host memory + 41 TB from GPU memory)
- Theoretical-double precision peak performance: 5.31 Petaflop/s (0.32 Pflop/s from CPUs + 4.99 Pflop/s from GPUs)

Cabeus aarch64 GPU Nodes

- 23 racks
- 538 nodes
- 38,736 CPU cores + 4,545,024 double-precision GPU cores
- Total memory: 258,240 GB (CPU) + 51,648 GB (GPU)
- Theoretical-double precision peak performance: 2,147 Teraflops (CPU) + 17,998 Tflops (GPU)

GPU Nodes by Model

Each GPU compute node includes one CPU socket as the host and one or four GPU cards as the coprocessors. These compute nodes are grouped by the CPU processor models as listed below:

- **model=mil_a100**

  128 nodes total. Each node contains one AMD Milan CPU socket and four NVIDIA Amper A100 GPU cards, where the CPU and GPUs are connected via PCI Express 4.0 bus.

  The Milan CPU socket used in the `mil_a100` nodes are AMD EPYC 7763.

- **model=gh200**

  538 nodes total. Each node contains one NVIDIA GH200 Grace Hopper Superchip, which consists of one Grace CPU (Neoverse V2 core with ARM v9.0-A architecture) and one H100 GPU, connected by a high-bandwidth, memory-coherent NVLink-C2C interconnect.

Cabeus Processor, Memory, and Network Subsystems Statistics

	Node Details	
	mil_a100	**gh200**
Architecture	Apollo 6500 Gen10 Plus	Supermicro SuperServer ARS-111GL-NHR
Total # of Nodes	128	538
Host name	cb[01-05,08-09]n[01-12], cb[06-07]n[01-10], cb[10-12]n[01-08]	cgh[1-3,6-15,17-19,21-23]n[01-26], cgh[4,20]n[01-12], cgh16n[01-20]
	CPU Host	
Processors Model	64-core EPYC 7763	72-core Grace (Neoverse V2)
# of Physical CPU Cores/Node	64	72
CPU-Clock	2.45 GHz	3.465 GHz
Maximum Double Precision Floating Point Operations per cycle per core	16	16
Total CPU Double Precision Flops/Node	2,508 GFlops	3.99 TFlops
Host Memory/Node	512 GB (8 x 64 GB DDR4)	480 GB (LPDDR5X)
	GPU	
GPU Architecture	Ampere	Hopper
Compute Capability**	cc80	cc90
Device Name	NVIDIA A100-SXM4-80GB	NVIDIA GH200 480GB

Clock Rate (Base/Boost)	1,275 MHz/1,410 MHz	(n/a)/1,980 MHz
# of coprocessors/Node	4	1
# of Streaming Multiprocessors (SMs)/coprocessor	108	132
# of Single Precision CUDA Cores/coprocessor	6,912	16,896
# of Double Precision CUDA Cores/coprocessor	3,456	8,448
# of Tensor Cores/coprocessor	432	528
Maximum double precision floating point operations per cycle per CUDA core	2	2
Total GPU Double Precision Flops/Coprocessor (Base/Boost) using Double Precision CUDA Cores	8.8 TFlops /9.7 TFlops	33.4 TFlops
Total GPU Double Precision Flops/Node (Base/Boost) using Double Precision CUDA Cores	35.2 TFlops/38.8 TFlops	33.4 TFlops
L2 Cache	40 MB	60 MB
Global Memory/coprocessor	80 GB (HBM2e)	96 GB (HBM3)
Memory Clock Rate	1,593 MHz	2,619 MHz
Memory Bus Width	5,120 bits	6,144 bits
Memory Bandwidth	2,039 GB/s (if using SXM form factor; 1,935 GB/s if using PCI form factor)	4,000 GB/s
GPU-to-GPU Communication	with NVLink 3.0: 600 GB/s	N/A
CPU-to-GPU Communication	PCIe 4.0 with 31.5 GB/s bandwidth for each direction	NVLink-C2C with 900 GB/s bi-directional
Inter-node Network		
IB Device on Node	1 HDR 200 Gb 2-port card + 1 GbE 4-port card	NDR 200 Gb 1-port
Local Disk		
SSD/node***	raw (2 x 1.92 TB SSD drives + 2 x 3.84 TB NVMe SSD) or (6 x 2.93 TB NVMe SSD)	960 GB Samsung NVMe M.2

** The compute capability identifies the features supported by the GPU hardware and is used by applications at runtime to determine which hardware features and/or instructions are available on the present GPU. Do not confuse the compute capability number with the CUDA version.

*** Not yet available for public usage, pending configuration decisions.

For more hardware details, access a GPU node through a PBS session and run one of the following commands:

For CPU info:

```
cat /proc/cpuinfo
```

For host memory info:

```
cat /proc/meminfo
```

Note: For the gh200 nodes with the unified memory feature,
the total memory shown in the output of /proc/meminfo is the sum
of the CPU memory and the GPU memory.

For GPU info:

```
/usr/bin/nvidia-smi -q
```

or load a nvhpc module and run the command pgaccelinfo or nvaccelinfo.

For example, for systems with x86_64 architecture:

```
module avail nvhpc
---------- /nasa/modulefiles/toss4 ----------

nvhpc-nompi/22.3 nvhpc-nompi/23.11 nvhpc/23.5 nvhpc/24.1
nvhpc-nompi/23.5 nvhpc-nompi/24.1 nvhpc/23.7 nvhpc/openmpi-3.1.5
nvhpc-nompi/23.7 nvhpc/22.3 nvhpc/23.11 nvhpc/openmpi-4.0.5

module load nvhpc/23.11
```

For example, for systems with aarch64 architecture:

```
module avail nvhpc
module load nvhpc/24.11

pgaccelinfo
or
nvaccelinfo
```

Running Jobs on Athena

To run a PBS job on the Athena compute nodes, submit the job from one of the Athena front-end systems (ATHFEs) to one of the queues managed by the PBS server pbs06a: devel, normal, long, and wide.

Note: You cannot submit jobs to run on Athena from a PFE.

Running Jobs on Aitken and Electra

To run a PBS job on the Aitken or Electra compute nodes, submit the job from one of the PFEs to one of the queues managed by the PBS server pbspl1.

Aitken Milan Nodes: You can submit batch jobs to run on Aitken's Milan nodes (mil_ait) either from a PFE or from one of the two Aitken Milan front-end systems (afe01, afe02). Interactive PBS jobs for Aitken Milan nodes must be submitted from the AFEs because currently there is no direct network connection to the PFEs.

Running Jobs on Endeavour

To run a PBS job on the shared memory systems Endeavour3/4, submit the job from a PFE to one of the queues managed by the PBS server pbspl4.

Running Jobs on Cabeus

To run a PBS job on:

- Milan A100 (mil_a100) submit the job from a Cabeus front end (cfe01 or cfe02) to one of the queues managed by the PBS server pbspl4.
- Grace Hopper nodes (gh200), submit your job from a Cabeus Grace Hopper front end (cghfe01 or cghfe02) to one of the queues served by the PBS server pbs05a.

PBS Queues

To view a complete list of all queues, including their defaults or limits on the number of cores, wall time, and priority, run theqstat command from an appropriate front-end system as follows:

```
qstat -Q
```

To view more detailed information about each queue, such as access control, defaults, limits on various resources, and status:

```
qstat -fQ queue_name
```

Brief summaries of the various queues are provided in the following sections.

Queues for Athena Users

You can run jobs on Athena using any of the following queues:

```
Queue   NCPUs/      Time/
name     max/def    max/def    pr
======= =====/=== ======/====== ===
devel    256/ 1    02:00/--     100
normal    --/ 1    08:00/--       0
long      --/ 1   120:00/--       0
wide     128/64    24:00/--      20
```

Queues for Aitken and Electra Users

You can run jobs on Aitken and Electra using any of the following queues:

```
Queue   NCPUs/      Time/
name     max/def    max/def    pr
======= =====/=== ======/====== ===
low       --/  8  04:00/ 00:30 -10
normal    --/  8  08:00/ 01:00   0
long      --/  8 120:00/ 01:00   0
debug     --/  8  02:00/ 00:30  15
devel     --/  1  02:00/    -- 101
```

The normal, long and low queues are for production work. The debug and devel queues have higher priority and are for debugging and development work.

For the debug queue, each user is allowed a maximum of 2 running jobs using a total of up to 128 nodes. For the devel queue, each user is allowed only 1 job (either queued or running).

Queues for Cabeus GPU Users

For information about queues for the Cabeus GPU nodes, see Requesting GPU Resources.

See also:

- Preparing to Run on Cabeus Grace Hopper Nodes
- Preparing to Run on Cabeus A100 GPU Nodes

Queues for Endeavour Users

Jobs submitted from PFEs to the e_normal, e_long, e_debug, and e_low queues are routed to run on the Endeavour shared-memory system, which has its own PBS server.

See Endeavour PBS Server and Queue Structure for more information.

Queue for Lou Users

The ldan queue provides access to one of the Lou data analysis nodes (LDANs) for post-processing archive data on Lou.

The ldan queue is managed by the same PBS server as all of the Electra and Aitken queues and is accessible by submitting jobs from an LFE or PFE.

Each job on the ldan queue is limited to 1 node and 72 hours. Each user can have a maximum of 2 running jobs on this queue.

See Lou Data Analysis Nodes for more information.

Queues with Restricted Access

The vlong Queue

Access to the vlong queue requires authorization from the NAS User Services manager.

A maximum of 1024 nodes and 16 days are allowed on this queue.

To avoid losing all useful work in the event of a system or application failure, ensure that your application includes checkpoint and restart capability; this is especially important when using this queue.

The wide Queue

The wide queue is a high priority queue used for jobs that meet either one of the following conditions:

- The number of requested NCPUs (number of nodes * number of cores per node) exceeds 25% of the user's mission shares
- The number of nodes requested of a processor model exceeds 25% of the total number of nodes of that model. See the Electra and Aitken configuration pages for the total number of nodes for each model.

Users do not have direct access to the wide queue. Instead, a job that meets the above criteria is moved into the wide queue by NAS staff. Only 1 job in the wide queue is allowed among Electra and Aitken. When more than one job meets the criteria, preference is given to the job that has been waiting for longer time or whose owner does not have other jobs running.

Athena: You can submit jobs to the Athena wide queue (with minimum of 64 nodes and maximum of 128 nodes for up to 24 hours) without requesting approval from NAS staff.

Various Special Queues

Special queues are higher priority queues for time critical projects to meet important deadlines. Some examples of special queues are armd_spl, astro_ops, heomd_spl, kepler, smd1, smd2, smd_ops, and so on. Request for creation and access of a special queue is carefully reviewed by the mission directorate or by the program POC who authorizes the project's SBU allocation. The request is rarely granted unless the project's principal investigator demonstrates that access to a special queue is absolutely necessary.

Advanced Reservation Queues

An advanced reservation queue allows you to reserve resources for a certain period ahead of time. One circumstance in which the NAS User Services manager may authorize the use of a reservation queue is when a user plans to run an extra-large job, such as one exceeding 15,000 cores, which would never start running without a reservation.

A reservation queue is created by NAS staff for the user and is normally named with the letter R followed by a number.

To find the status of existing PBS advanced reservations, issue the pbs_rstat command on a PFE.

Note: Queues such as alphatst, ded_time, diags, dpr, idle, resize_test, and testing are used for staff testing purposes, and are not accessible by general users.

The Cabeus cluster comprises two types of NVIDIA GPU nodes, including nodes with x86_64-based CPUs and nodes with aarch64-based CPUs.

Note: You must have a Standard Billing Unit (SBU) allocation specifically for GPUs in order to use the GPU nodes.

Nodes with x86_64-Based CPUs

- 128 nodes each with one Milan CPU and four NVIDIA A100 GPUs (model=mil_a100)

These GPU nodes are served by PBS server pbspl4 and are accessible from the Cabeus front ends, cfe01 and cfe02.

The nodes are sharable by different jobs. Instead of each node being treated as one unit for exclusive access by a single job, each node is logically split into four *vnodes*.

If you want exclusive access to a node, request the use of all the resources in a node and specify the placement -lplace=scatter:excl. for example:

```
#PBS -lselect=1:ncpus=128:ngpus=4:mem=480GB:model=mil_a100
#PBS -lplace=scatter:excl
#PBS -q queue_name@pbspl4
```

where the use of @pbspl4 in the line #PBS -q is optional.

Note: There are 64 physical cores per mil_a100 node. However, PBS sees each mil_a100 node as having 128 logical cores due to hyperthreading enabled. If you specify :ncpus=64, PBS will assign 64 logical cores all from the first 32 physical cores. To get all 64 physical cores assigned to your job, use :ncpus=128.

Nodes with aarch64-based CPU

Cabeus contains 538 nodes with NVIDIA GH200 Superchips (model=gh200).

These nodes are accessible from the Cabeus Grace Hopper front ends, cghfe01 and cghfe02. PBS jobs are served by pbs05a, where user login is not allowed.

The gh200 nodes have only one GPU per node. A node is configured via PBS for exclusive use by a single job, and it is not necessary to explicitly request the amount of resources. For example, the following two PBS resource requests provide you with the same exclusive access to all resources of a whole node:

```
#PBS -lselect=1:model=gh200
or
#PBS -lselect=1:ncpus=72:ngpus=1:mem=460GB:model=gh200
#PBS -q queue_name@pbs05a
```

where the use of @pbs05a in the line #PBS -q is optional.

Requesting AMD Milan A100 GPU Nodes (mil_a100)

Each mil_a100 node contains only one socket; each socket is further split into four vnodes where eachmil_a100 vnode contains 1 GPU, 16 CPU physical cores, and 128 GB of memory.

The new queue structure utilizes the PBS primetime feature. A "nonprime-time" queue, np_gpu_wide, will run jobs only from 7:00 a.m. Thursday to 7:00 a.m. Saturday (Pacific Time). The gpu_normal and gpu_devel queues will run during both primetime and non-primetime (i.e., any time). You can submit jobs to any of the queues at any time during the week, but the non-prime time jobs will run only during their designated period.

The following PBS queues are enabled for use:

- **gpu_normal**
  - Maximum wall time: 24 hrs
  - Maximum node count: 32

- **gpu_devel**
  - Maximum wall time: 2 hrs
  - Maximum node count: 2
  - Maximum user jobs (queued or running): 1
  - Priority: 10

- **np_gpu_wide (non-primetime)**
  - Maximum wall time: 12 hrs
  - Maximum node count: 64
  - Minimum node count: 24
  - Maximum user jobs (queued or running): 2
  - Priority: 20

Notes:

- Long running jobs need to implement checkpoint/restart if they cannot fit into the wall-time limit of these queues.
- The maximum amount of resources or certain settings for these queues may change from time to time. To get the current setting, run the command `qstat -fQ queue_name`. Note that the `resources_max.nodect` value refers to the number of physical nodes allowed per job.

In your job request, specify the number of CPU cores, the number of GPUs, and the amount of host memory you want for your job. The following example shows how to request exclusive usage of a whole node:

```
#PBS -lselect=1:ncpus=128:ngpus=4:mem=480GB:model=mil_a100
#PBS -lplace=scatter:excl
```

If `ncpus`, `ngpus`, or `mem` is not specified, the default values (`ncpus=1`, `ngpus=1`, `mem=32GB`) will be used.

Requesting NVIDIA GH200 GPU Nodes (gh200)

To run on the GH200 nodes, submit your PBS interactive or batch job from one of the two Grace Hopper front ends, cghfe01 or cghfe02. Use one of the following queues:

- **gpu_normal**
  - Maximum wall time: 24 hrs
  - Priority: Default (0)

- **gpu_devel**
  - Maximum wall time: 2 hrs
  - Maximum node count: 2
  - Maximum user jobs (queued or running): 1
  - Priority: 10

- **gpu_wide**
  - Maximum wall time: 12 hrs
  - Priority: 20
  - Wide is considered 85 or more nodes requested for a job.

- **gpu_long**
  - Maximum wall time: 120 hrs
  - Maximum node count: 8
  - Maximum user jobs (queued or running): 2
  - Priority: Default (0)

Note: Since each `gh200` node is set up to not allow job sharing, the use of PBS placement such as `-lplace=scatter:excl` is not required.

TIP: The available queues will change from time to time. Use the command `qstat -Q` and `qstat -fQ` to find what queues have been created and enabled to be used by your group.

Charging for Use of the GPU Nodes

A separate SBU allocation exclusively for the GPU nodes is required. For information on requesting SBU allocation for the GPU nodes, see this article.

The SBU rates for using 1 GPU node for 1 hour are:

- `mil_a100` with four A100s: 37.86
- `gh200` with one H100: 18.34

Note: For the GPU nodes with aarch64-based CPU (`model=gh200`), your job will be charged for the SBU cost of a 1 whole node regardless of how the resources are requested, since these nodes are not set up for job sharing (each job is allocated with all of the resources in a whole node).

For the GPU nodes with x86_64 based CPUs, since a node is sharable by multiple jobs, your job may be charged for the SBU cost of 1 whole node or a fraction of the node depending on the resource request and the place statement in your PBS job. In addition, charging is based on the number of GPUs allocated to or excluded from use by other jobs, with no consideration of CPUs and memory. Below are three examples.

Example 1

This resource request will be charged for one-quarter of the `mil_a100` SBU rate of 37.86, because only 1 GPU will be allocated to the job. In addition, it allows the unused CPU, GPU, and memory in the node to be used by other jobs:

```
#PBS -lselect=1:ncpus=1:ngpus=1:mem=60GB:model=mil_a100
#PBS -lplace=scatter:shared
```

Example 2

This job will be charged for 2 times the SBU rate of 37.86, because all four GPUs in each of the two`mil_a100` nodes will be allocated

and no other jobs will be able to share the resources:

```
#PBS -lselect=2:ncpus=1:ngpus=4:mem=60GB:model=mil_a100
#PBS -lplace=scatter:excl
```

Example 3

This job will be charged for one half (2 x 0.25) the SBU rate of 37.86, because only 1 GPU each from 2 different nodes will be allocated and the remaining 3 GPUs in each node can be used by other jobs:

```
#PBS -lselect=2:ncpus=1:ngpus=1:mem=60GB:model=mil_a100
#PBS -lplace=scatter:excl
```

Checking Job Status

To check the status for all of your jobs submitted to a particular queue, run the qstat command. The following examples use the queue gpu_normal served by the pbspl4 PBS server. For jobs submitted to queues served by the pbs05a PBS server, change the front end from cfe to cghfe01 or cghfe02, the PBS server from pbspl4 to pbs05a, and use an appropriate queue name.

```
cfe% qstat gpu_normal@pbspl4 -u your_username
```

You can also check the status of a single job. For example, to check the status of job 140101 on pbspbl4:

```
cfe% qstat 140101.pbspl4.nas.nasa.gov
```

For single-node I/O that uses memory-based temporary filesystem, we recommend using a job-based temporary directory, $TMPDIR, instead of the node-wide /tmp directory. This recommendation is particularly important for jobs running on the NAS GPU nodes with x86_64-based CPUs (mil_a100) where node sharing is enabled.

Background

All of the NAS compute nodes—including the CPU nodes in Electra, Aitken, and the GPU nodes in Cabeus—are configured to allow up to 50% of the node's main memory for use as a temporary filesystem (/tmp) to achieve faster single-node I/O. However, using the /tmp directory is not feasible for multi-node I/O, as /tmp is a filesystem local to a single node.

Many users use the /tmp directory on the CPU nodes to speed up their jobs' I/O. When these jobs end, data under/tmp are not always removed by the application or user. In order to provide a clean /tmp directory for a new job that uses the same node, the PBS prologue will first unmount /tmp to clean up leftover data from the previous job, then will remount it for the new job.

On the GPU nodes where node sharing is enabled, the resources on each node (including/tmp) can be shared by multiple jobs. Therefore, the process of cleaning up /tmp depends on whether a new job is the only job on a node or whether there are other jobs running on the node:

- If a new job starting is the only job on a node, the PBS prologue will use the same process as on CPU nodes by unmounting and remounting /tmp.
- If there are other jobs currently running on the same node, the prologue will not unmount /tmp, because doing so would remove the data needed by the other running jobs. Instead, the prologue will look for /tmp/pbs.*jobID* directories that do *not* have a running job, and delete those directories if they exist. The $TMPDIR path is /var/tmp/pbs.*jobid*.

Therefore, to ease the removal of user data from old jobs, you should use the $TMPDIR directory instead of writing directly to /tmp.

Using the $TMPDIR Directory

Note: For some NAS systems, /var/tmp is a symlink of /tmp. For the Milan A100 (mil_a100) nodes, /var/tmp and /tmp are two different directories, but they both use the same 50% main memory pool of the nodes.

How much $TMPDIR space can a job use?

On the GPU nodes, $TMPDIR cannot use more than the amount of CPU host memory requested by a job. For example, in the following PBS job request, the 16 GB of CPU host memory requested by the job will be shared by the memory usage of the application; buffer cache usage of the application (if buffered I/O is enabled); and the usage for storing data under /tmp or $TMPDIR:

```
#PBS -lselect=1:ncpus=10:ngpus=1:mem=16GB:model=mil_a100
```

Therefore, the maximum space available for $TMPDIR is 16 GB in this example.

In addition, the combined $TMPDIR usage by all jobs running on the same node cannot exceed 50% of the node's main memory. If another job running on the same node as your job already has data stored in the job's $TMPDIR with an amount equal to the size of 50% of the node's main memory, you will encounter a "No space left on device" error when you attempt to write data into your job's $TMPDIR.

Sample Script for Using $TMPDIR

A job's resources (CPU, GPU, and memory) are accessible only during the lifetime of the job. To use $TMPDIR for I/O, you must copy data into $TMPDIR before the data are needed by your application. Any generated data needed for post-processing or restarting the next job must be copied to the $HOME or Lustre /nobackup directory before the current job is terminated. You can accomplish this by using the following sample PBS script:

```
#PBS -lselect=1:ncpus=16:ngpus=1:mem=16GB:model=mil_a100
#PBS -q gpu_devel

module load ....

cd $TMPDIR

cp /nobackup/username/project/input .

mpiexec -np ....

cp output /nobackup/username/project
#End of script
```