To help you prepare for running jobs on Athena's Turin nodes, this short user guide includes information on the general configuration of Turin nodes, compiling your code, and running PBS jobs.

Athena Front-End Systems

Four new front-ends (athfe[*01-04*]) serve the Athena Turin compute nodes. Each Athena front-end node includestwo 128-core AMD EPYC 9745 processors with a total of 256 physical cores and 768 GB memory, two 200 Gb/s Slingshot interconnects, and two 25 Gb/s dedicated Ethernet interfaces for external connections.

From your local desktop, access the ATHFEs via a two-step login or a one-step login:

- Two-step login:

  local_desktop% ssh -l *your_NAS_username* sfe6.nas.nasa.gov
  (or replace sfe6 with sfe7 or sfe8)
  sfe6% ssh athfe01 (or ssh athfe02, athfe03, athfe04)

  You can omit -l *your_NAS_username* if your usernames on NAS systems and your local system are identical. If you are not using the NAS config file template, you can add the following lines in your local config file to include all of the ATHFEs:

```
Host athfe01
   HostKeyAlias        athfe01.nas.nasa.gov
   ProxyCommand        ssh -ax -oCompression=no sfe ssh-balance %h

Host athfe02
   HostKeyAlias        athfe02.nas.nasa.gov
   ProxyCommand        ssh -ax -oCompression=no sfe ssh-balance %h

Host athfe03
   HostKeyAlias        athfe03.nas.nasa.gov
   ProxyCommand        ssh -ax -oCompression=no sfe ssh-balance %h

Host athfe04
   HostKeyAlias        athfe04.nas.nasa.gov
   ProxyCommand        ssh -ax -oCompression=no sfe ssh-balance %h
```

- Then, run the command:

  local_desktop% ssh athfe01 (or ssh athfe02, athfe03, athfe04)

Athena Turin Compute Nodes

Athena currently includes 1,024 Turin compute nodes partitioned into four physical racks. These racks are housed in Compute Module 2 of the Modular Computing Facility (MSF), located less than one mile from the NAS main building.

Note: We plan to add more nodes to Athena in 2026.

Each node contains two 128-core AMD EPYC 9745 Turin sockets (2.4 GHz) for a total of 256 cores. Each node has 768 GB of memory. With Simultaneous Multithreading (SMT) enabled, each node has 256 physical cores and 512 logical cores.

TIP: It is possible that during Turin's early production, some nodes are configured with SMT on and some with SMT off. To check whether SMT is enabled on a particular Turin compute node, run lscpu | grep "Thread(s) per core" on the node. If it shows 2, then SMT is enabled on that node.

The Athena compute nodes are inter-connected with Cray Slingshot 11 Cassini network interface cards (NICs) and Rosetta Ethernet switches. All nodes have access to the NAS compute resources home filesystems (/u/*username*) and Lustre filesystems (/nobackup/*username*).

The Athena compute nodes are only accessible by submitting PBS jobs from the ATHFEs, not from the PFEs. The PBS server for these compute nodes is pbs06a. Users cannot login to pbs06a.

Requesting CPU Allocations

An SBU allocation for the NAS CPU systems, named hecc in the acct_ytd command output, is required to use the Athena compute nodes through PBS. If you are new to the NAS systems, you can apply for allocations on this web site.

The SBU rates (for using 1 node for 1 hour) for all NAS Systems can be found underJob Accounting.

Operating System and Software Modules

The operating system on all Turin compute nodes is the Red Hat Enterprise (9.6) Linux-based Tri-Lab Operating System Stack Version 5 (TOSS 5). The HPE Performance Cluster Manager (HPCM 1.14) is used to manage the cluster.

Software packages are managed with the Tool Command Language (Tcl) environment modules system. On Athena, you should use

the module command from sourcing init scripts under /opt/cray/pe/modules/3.2.11.7, not the one under the /usr/local directory. In your login sessions or job scripts, check whether you are using the correct version of the module command by using:

```
% module -V
```

A module -V output that begins with VERSION=3.2.11.7 shows you are using the right version of the module command:

If your output shows Modules Release 5.3.0 you should modify your scripts to use the following lines for Athena:

```
source /opt/cray/pe/modules/3.2.11.7/init/csh (for csh)
or
source /opt/cray/pe/modules/3.2.11.7/init/bash (for bash)
```

One way to accomplish this is as follows:

For .cshrc:

```
if ( -e /opt/cray/pe/modules/default/init/csh ) then
   source /opt/cray/pe/modules/default/init/csh
else
   source /usr/local/lib/init/global.cshrc
endif
```

For .bashrc:

```
if [ -f /opt/cray/pe/modules/default/init/bash ]; then
  source /opt/cray/pe/modules/default/init/bash
else
  source /usr/local/lib/global.profile
fi
or
source /opt/cray/pe/modules/default/init/bash >& /dev/null || \
   source /usr/local/lib/global.profile
```

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

The default $MODULEPATH upon login includes:

- /opt/modulefiles
- /opt/cray/modulefiles
- /opt/cray/pe/modulefiles
- /opt/cray/pals/modulefiles
- /opt/cray/pe/perftools/25.09.0/modulefiles
- /opt/cray/pe/craype-targets/default/modulefiles
- /nasa/modulefiles/toss5_crayex
- /usr/share/Modules/modulefiles
- /usr/share/modulefiles
- /etc/modulefiles
- /etc/scl/modulefiles

To see what software modules are available, run module avail at the command prompt.

Porting Your Application

The Athena cluster is based on the x86_64 architecture and Cray's proprietary Slingshot interconnects. In order to take advantage of the high-performance Slingshot interconnects, an MPI or an MPI/OpenMP hybrid application needs to be built/linked with the Cray MPICH library. An existing executable built on other NAS CPU systems with the HPE/SGI MPT library or OpenMPI library will not run on Athena. One must rebuild the MPI application on Athena with a cray-mpich module, which is integrated in the Cray Programming Environment (CPE).

The Cray Programming Environment offers choices of various compilers. The default is to use a Cray compiler integrated in the PrgEnv-cray module. If you prefer to use the Intel, AMD AOCC or GNU compiler, one is available and integrated in the corresponding PrgEnv-xxx module.

Note: PrgEnv-xxx is case-sensitive.

Cray Programming Environment

The HPE Cray Programming Environment offers a set of tools tightly integrated for developing, porting, debugging, and tuning applications. It is the main software stack recommended for use on the Athena Turin nodes.

Various PrgEnv-xxx modules are available for choosing an environment with a specific compiler flavor. After loading a PrgEnv-xxx module, invoking the ftn, cc, CC drivers will correctly use the corresponding Fortran, C and C++ compiler integrated within that environment with appropriate options to build and link applications with relevant dynamic libraries.

Choosing the Cray Compiler Integrated in PrgEnv-cray

The CPE version 25.09 is installed on Athena. By default, several modules of CPE 25.09 are loaded automatically when you login to

an Athena front-end or compute node.

Note: When a new CPE is added and loaded by default as configured by the NAS system adminstrators in the future, some of the following modules may be updated with a new version.

```
% module list
Currently Loaded Modulefiles:
  1) cray-libpals/1.7.1       7) cce/20.0.0
  2) cray-pals/1.7.1          8) craype/2.7.35
  3) craype-x86-turin         9) cray-dsmml/0.3.1
  4) libfabric/12.0          10) cray-mpich/8.1.33
  5) craype-network-ofi      11) cray-libsci/25.09.0
  6) perftools-base/25.09.0  12) PrgEnv-cray/8.6.0
```

where, in the order of relevance to porting your application,

- 12) PrgEnv-cray
  is a meta-module that sets up a complete programming environment based on Cray's compilers and tools
- 8) craype (Cray Programming Environment Drivers)
  provides compiler/link driver wrappers (ftn, cc, CC) that automatically include the correct compiler options and link relevant libraries as required by the currently loaded modules. The --help option of ftn, cc, CC will print the help page of the corresponding compiler in that environment.
- 7) cce (Cray Compiling Environment)
  includes a suite of Cray compilers, libraries, and utilities that enable users to compile, analyze code to automatically generate highly optimized executables for Cray systems. This modulefile defines the system paths and environment variables needed to run the Cray Compiling Environment. With PrgEnv-cray, ftn, cc, CC invokes the Cray compiler drivers crayftn, craycc, crayCC. See crayftn, craycc, crayCC man pages or use the --help option for available Cray compiler flags.
- 10) cray-mpich
  is a Cray-developed high-performance, optimized implementation of open-source MPICH

  Note: Loading a cray-mpich module provides access to scripts such as mpifort, mpif77, mpif90, mpicc, mpicxx, mpic++ as well as various libmpi* for building MPI applications. Read the intro_mpi man page for more information.

  During runtime, a cray-pals module is needed to access the mpiexec executable in Cray's parallel application launch service (PALS).

- 2) cray-pals
  is a part of PALS that handles the startup and coordination of parallel applications across compute nodes; loading this module automatically loads the cray-libpals module
- 1) cray-libpals
  is a library of PALS
- 5) craype-network-ofi
  enables MPI applications to use the OpenFabrics Interface (OFI) or Libfabric network transport; works in conjunction with cray-mpich to implement MPI communication over the OFI framework, offering performance characteristics that can be compared with the alternative transports like UCX.
- 4) libfabric
  provides a standardized, high-performance communication library for applications running on Cray systems, abstracting the underlying Slingshot network
- 11) cray-libsci
  is a collection of highly optimized numerical routines, such as BLAS, LAPACK, and ScaLAPACK, designed to enhance the performance of scientific and HPC applications on Cray's systems
- 9) cray-dsmml (Distributed Symmetric Memory Management Library)
  is a proprietary library that manages distributed memory heaps for PGAS (Partitioned Global Address Space) languages like Coarray Fortran, UPC, and OpenSHMEM
- 3) craype-x86-turin
  is a module that adds reference to libraries for the purpose of building AMD Turin target apps; For Fortran, it turns on the Cray Fortran compiler flag -h cpu=x86-turin; for C/C++, it sets -target-cpu to znver5.
- 6) pertools-base
  is a foundational component for accessing the suite of Cray performance analysis tools (CrayPat, Apprentice2/3) without altering a program's behavior

If you purge your modules using the module purge command, reloading the PrgEnv-cray/8.6.0 module afterwards will load all of the above except the cray-pals, cray-libpals, craype-x86-turin and the perftools-base modules.

WARNING: If your system startup script (such as .cshrc or .profile) explicitly sources /usr/share/Modules/init/{csh,tcsh,bash,ksh} or /usr/local/lib/init/global.{cshrc,profile} or equivalent, you will encounter errors when you try module load PrgEnv-xxx .

For example:

```
% module purge
% module load PrgEnv-cray
Loading craype/2.7.35
  ERROR: Conflicting PrgEnv-cray/8.6.0 is loading

Loading PrgEnv-cray/8.6.0
  ERROR: Load of requirement craype failed
```

The solution is to (1) comment out sourcing those files, and (2) source /opt/cray/pe/modules/3.2.11.7/init/csh (for csh) or /opt/cray/pe/modules/3.2.11.7/init/bash (for bash) in your system startup scripts (for Athena) and job scripts.

Choosing the Intel compiler integrated in PrgEnv-intel

If you prefer to use the Intel compiler, loading the PrgEnv-intel/8.6.0 module after the module purge command will load the following modules:

1) intel/2023.2.1 4) libfabric/12.0 7) cray-libsci/25.09.0
2) craype/2.7.35 5) craype-network-ofi 8) PrgEnv-intel/8.6.0
3) cray-dsmml/0.3.1 6) cray-mpich/8.1.33

You can switch module from intel/2023.2.1 to intel-classic/2023.2.1 or intel-oneapi/2023.2.1. The Intel compilers invoked by ftn, cc, and CC when each of these intel modules is loaded are:

module loaded	intel/2023.2.1	intel-classic/2023.2.1	intel-oneapi/2023.2.1
ftn	ifort	ifort	ifx
cc	icx	icc	icx
CC	icpx	icpc	icpx

Note: When you choose to use PrgEnv-intel, the wrappers ftn, cc, CC will use the Intel compilers and link to Cray's libraries, not Intel's libraries. To link to Intel's library, for example, MKL library, add -qmkl={sequential,parallel,cluster} to the ftn, cc, CC command. During runtime, you may have to set the LD_LIBRARY_PATH to include the path of the MKL library (/nasa/intel/oneapi/mkl/2023.2.0/lib/intel64).

Choosing the AMD AOCC compiler integrated in PrgEnv-aocc

If you want to use the AMD AOCC compiler, loading PrgEnv-aocc/8.6.0 will load the aocc/5.0.0 module together with the craype, cray-libsci, cray-mpich, cray-network-ofi, libfabric, cray-dsmml modules.

Choosing the GNU compiler integrated in PrgEnv-gnu

Loading the PrgEnv-gnu/8.6.0 module will load gcc-native/14.2. In addition to the compiler module, the craype, cray-libsci, cray-mpich, cray-network-ofi, libfabric, cray-dsmml modules are also loaded.

Building and Running MPI Applications

Your existing executables built with HPE/SGI MPT or OpenMPI library will not run on the Athena Turin nodes. Rebuild your application using the recommended Cray compiler wrappers (ftn, cc, CC) and MPI library.

Building with PrgEnv-cray

athfe01% module load PrgEnv-cray

This should have also loaded a cray-mpich module. If not, load it explictly.

athfe01% module load cray-mpich

Method 1: Use ftn, cc, CC wrappers

When the modules PrgEnv-cray and cray-mpich are loaded, the Cray compiler wrappers ftn, cc, CC will automatically link the Cray MPI library. This is the preferred way to build an MPI application as the Cray compiler wrappers are designed to work specifically with the Cray MPICH implementation and other Cray system libraries, ensuring compatibility and optimized performance.

athfe01% ftn <FFLAGS> source.f90
athfe01% cc <CFLAGS> source.c
athfe01% CC <CXXFLAGS> source.cpp

TIP: Use the ftn, cc, CC flag -craype-verbose to print the full include and link information used by the compiler wrapper. For even more detailed information, use crayftn flag -v (accepted by ftn) to see what the compiler, lister (which generates a detailed, annotated listing of source code) and linker is doing and what it is calling. The Cray C and C++ compiler option -### acts as a dryrun (i.e., it prints what the compiler, linker, etc. will be doing for your code but does not actually run the command).

Method 2: Use mpifort, mpicc, mpicxx wrappers

Instead of using ftn, cc, CC, you can also use the various mpixxx scripts to build your MPI applications. Furthermore, this is the way to build your MPI application if cray-mpich is loaded but not PrgEnv-cray.

athfe01% mpifort <FFLAGS> source.f90
athfe01% mpicc <CFLAGS> source.c
athfe01% mpicxx <CXXFLAGS> source.cpp

Note: mpif77, mpif90 are soft links to mpifort. mpic++ is a soft link to mpicxx.

Building with PrgEnv-intel

athfe01% module switch PrgEnv-cray PrgEnv-intel

Method 1: Use ftn, cc, CC wrappers

With PrgEnv-intel, the default compiler module loaded is intel/2023.2.1. You can switch from intel/2023.2.1 to intel-classic/2023.2.1 or intel-oneapi/2023.2.1 depending on which combination of Fortran (ifort, ifx), C (icc, icx), and C++ compilers (icpc, icpx) you want. The ftn, cc, CC wrappers will correctly use the compilers associated with each of these three Intel compiler modules and link with the Cray MPICH library.

TIP: You can pass in the Intel compiler flag -dryrun to the ftn, cc, CC wrappers to show what the Intel driver tool commands to be used.

Method 2: Use mpifort, mpicc, mpicxx wrappers

If you prefer to use the mpifort, mpicc, mpicxx from the cray-mpich module to build your MPI applications, be aware that mpifort, mpicc, mpicxx default to using ifx, icx, icpx regardless of which of intel, intel-classic, intel-oneapi module you have loaded. Therefore, it is recommended to set the following environment variables to use the compiler you want before building with mpifort, mpicc, mpicxx from Cray MPICH. For example, to make sure ifort, icc, icpc are used by the mpifort, mpicc, mpicxx wrappers, do the following before invoking mpifort, mpicc, mpicxx:

With csh:

```
setenv MPICH_FC ifort
setenv MPICH_CC icc
setenv MPICH_CXX icpc
```

With bash:

```
export MPICH_FC=ifort
export MPICH_CC=icc
export MPICH_CXX=icpc
```

Method 3: Explictly invoke compilers and link to MPI libraries

If you prefer not to use any wrappers, you will need to explictly provide the paths to the appropriate include files and libraries. For example, if you are building with ifort and linking with the Cray MPICH 8.1.33 library, do:

```
ifort [options] program.f90 \
-I/opt/cray/pe/mpich/8.1.33/ofi/intel/2022.1/include \
-L/opt/cray/pe/mpich/8.1.33/ofi/intel/2022.1/lib \
-lmpi -lmpifort
```

Running MPI applications

To run your MPI executable, submit a PBS job as described in the Running PBS Jobs section. In addition, you will need to load the cray-pals module in order to access the mpiexec command.

```
% module use /opt/cray/pals/modulefiles
% module load cray-pals
Loading cray-pals/1.7.1
Loading requirement: cray-libpals/1.7.1

% which mpiexec
/opt/cray/pals/1.7/bin/mpiexec
```

Read man mpiexec for all options and environment variables.

Note: On Slingshot 11 systems, Cray MPICH sets the variable MPICH_OFI_USE_PROVIDER to cxi (the Cassini interfaces) by default. This variable controls the selection of a provider to use from a list of already-loaded providers. The variable FI_PROVIDER controls what providers to load. Setting FI_PROVIDER to just cxi (to be selected for use by MPICH_OFI_USER_PROVIDER) willl speed up the MPI startup.

Note: When running more than 1 MPI rank per node, both Cassini interfaces on a node will be used by default. There are environment variables (MPI_OFI_NIC_VERBOSE, MPI_OFI_NUM_NICS, MPICH_OFI_NIC_POLICY, MPICH_OFI_NIC_MAPPING, etc.) to display NIC selection, choose the number of NICs to use per node, select the rank-to-NIC assignment policy, specify the precise rank-to-NIC mapping on each node.

Building and Running OpenMP or MPI/OpenMP Hybrid Codes

With the Cray compiler, use the compiler flag -fopenmp or -homp to enable support for OpenMP directives. Without these options, the OpenMP directives are ignored.

Be aware that on a compute node, the default value of the variable OMP_NUM_THREADS is set to the value of ncpus in your PBS resource request. Remember to reset OMP_NUM_THREADS to a value of your choice before running.

TIP: If your OpenMP application is built with Cray OpenMP, set the variable CRAY_OMP_CHECK_AFFINITY to true to see how the OpenMP threads are placed on the cores.

For MPI/OpenMP hybrid code, it is critical to apply process and thread binding to avoid multiple threads running on the same core. The Cray PALS mpiexec includes some options to help with binding. For example, when running on 4 nodes with a total of 256 MPI processes where each node runs 64 MPI processes and 4 OpenMP threads, the following command provides one way for process/thread binding:

```
mpiexec -n 256 --ppn 64 --depth 4 --cpu-bind depth a.out
```

Another way to do process/threading binding is to use NAS's inhouse tool mbind.x, which is available at /u/scicon/tools/bin. You can also access mbind.x by loading the module scicon/cli_tools. To eliminate interference between Cray MPI's own binding scheme with mbind.x, one must add --cpu-bind none to mpiexec when using mbind.x. In addition, mbind.x takes the thread number either from the value of its -t option (higher precedence) or $OMP_NUM_THREADS. When the -t option is not specified and therefore $OMP_NUM_THREADS is used by mbind.x, be aware that the default value of $OMP_NUM_THREADS in a PBS job may be set to the value of NCPUS, which likely is not what you want. The default value of $OMP_NUM_THREADS can also cause surprising binding behavior when using mbind.x to bind MPI processes of pure MPI code. Thus, it is recommended to set $OMP_NUM_THREADS to 1 explicitly for PBS jobs running pure MPI applications.

With mbind.x, to run on 4 nodes with a total of 256 MPI processes where each node runs 64 MPI processes and 4 OpenMP threads, use:

```
mpiexec -n 256 --ppn 64 --cpu-bind none /u/scicon/tools/bin/mbind.x \
 -cs -t4 -v a.out

or

export OMP_NUM_THREADS=4 (bash)
#setenv OMP_NUM_THRREADS 4 (csh)
mpiexec -n 256 --ppn 64 --cpu-bind none /u/scicon/tools/bin/mbind.x \
 -cs -v a.out
```

Running PBS Jobs on Athena Turin Compute Nodes

The Athena Turin compute nodes are accessible only through PBS jobs submitted from athfe[01-04] served by the PBS server pbs06a.

To request Athena Turin compute nodes, you must use :model=tur_ath in your PBS script or the qsub command line, as shown in the example below.

TIP: The use of -lplace=scatter:excl is recommended if you plan to run across nodes with less than 512 processes on each node when SMT is enabled. This placement should be set as default. However, it is a good practice to double-check the setting.

Sample PBS Script For Athena Turin Nodes

```
#PBS -l select=2:ncpus=256:mpiprocs=256:model=tur_ath
# If mpiprocs is not set, it defaults to 1
#PBS -l walltime=8:00:00
#PBS -l place=scatter:excl
#PBS -q normal

# If using ProEnv-intel instead of PrgEnv-cray, uncomment the line below
# module swap PrgEnv-cray PrgEnv-intel
# If the module cray-pals is not loaded by default, load it manually
module use /opt/cray/pals/modulefiles
module load cray-pals

#set FI_PROVIDER may reduce MPI startup time
#with bash
FI_PROVIDER=cxi
#with csh
#setenv FI_PROVIDER cxi

cd $PBS_O_WORKDIR

# Run a total of 512 MPI processes with 256 processes per node
mpiexec -np 512 --ppn 256 ./a.out

# If wanting to use mbind.x for process/thread binding,
# reset OMP_NUM_THREADS from the default value to 1 for pure MPI code
# and you must disable Cray MPI's binding with --cpu-bind none

#with bash
OMP_NUM_THREADS=1
#with csh
#setenv OMP_NUM_THREADS 1
mpiexec -np 512 --ppn 256 --cpu-bind none /u/scicon/tools/bin/mbind.x \
 -cs -v ./a.out
```

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

- Cray Programming Environment 25.09 Software for HPCM
- Cray Compiling Environment (25.09)
- Cray Compiler Guide: C and C++ Reference Manual (24.07)
- Cray Compiler Guide: Fortran Reference Manual (24.07)

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.