



titan Since 11/5/14 12:55 am	eos Since 11/4/14 07:05 pm	rhea Since 11/4/14 02:40 pm	hpss Since 11/18/14 11:05 am
--	--------------------------------------	---------------------------------------	--

OLCF User Assistance Center

 9am to 5pm M-F

 help@nccs.gov

 (865) 241-6536

[Send a Help Ticket](#)

The center's normal support hours are 9 a.m. until 5 p.m. (Eastern time) Monday through Friday, exclusive of holidays. Outside of normal business hours, calls are directed to the ORNL Computer Operations staff. If you require immediate assistance outside of normal business hours, you may contact them at the phone number listed above. If your request is not urgent, you may send an email to help@nccs.gov, where it will be answered by a NCCS User Assistance member the next business day.

Support Overview
Getting Started
System User Guides
KnowledgeBase
Tutorials
Training Events
My OLCF
Software
Documents & Forms
Known Issues
OLCF Policies

[Home](#) › [User Support](#) › [System User Guides](#) › Titan User Guide

Titan User Guide

Contents

- 1.Titan System Overview
- 2.Requesting Access to OLCF Resources
 - 2.1.Project Allocation Requests
 - 2.2.User Account Requests
- 3.OLCF Help and Policies
 - 3.1.User Assistance Center
 - 3.2.Communications to Users
 - 3.3.My OLCF Site
 - 3.4.Special Requests and Policy Exemptions
 - 3.5.OLCF Acknowledgement
- 4.Accessing OLCF Systems
 - 4.1.OLCF System Hostnames
 - 4.2.General-Purpose Systems
 - 4.3.X11 Forwarding
 - 4.4.RSA Key Fingerprints
 - 4.5.Authenticating to OLCF Systems
- 5.Data Management
 - 5.1.User-Centric Data Storage
 - 5.1.1.User Home Directories (NFS)
 - 5.1.2.User Archive Directories (HPSS)
 - 5.2.Project-Centric Data Storage
 - 5.2.1.Project Home Directories (NFS)
 - 5.2.2.Project-Centric Work Directories
 - 5.2.3.Project Archive Directories (HPSS)
 - 5.3.Transferring Data
 - 5.4.Employing Data Transfer Nodes
 - 5.5.Data Management Policy Summary
- 6.Software and Shell Environments
 - 6.1.Default Shell
 - 6.2.Using Modules
 - 6.3.Installed Software
- 7.Compiling On Titan

7.1.Cray Compiler Wrappers
7.2.Compiling and Node Types
7.3.Controlling the Programming Environment
7.4.Compiling Threaded Codes
8.Running Jobs on Titan
8.1.Login vs. Service vs. Compute Nodes
8.2.Filesystems Available to Compute Nodes
8.3.Writing Batch Scripts
8.4.Submitting Batch Scripts
8.5.Interactive Batch Jobs
8.6.Common Batch Options to PBS
8.7.Batch Environment Variables
8.8.Modifying Batch Jobs
8.9.Monitoring Batch Jobs
8.10.Titan Batch Queues
8.11.Job Execution on Titan
8.11.1.Using the <code>aprun</code> command
8.11.2.XK7 CPU Description
8.11.3.Controlling MPI Task Layout Within a Physical Node
8.11.4.Controlling MPI Task Layout Across Many Physical Nodes
8.11.5.Controlling Thread Layout Within a Physical Node
8.12.Enabling Workflows through Cross-System Batch Submission
8.13.Job Resource Accounting
8.14.Titan Scheduling Policy
8.15.Aprun Tips
9.Development Tools
9.1.GPU Accelerated Libraries
9.2.Accelerator Compiler Directives
9.3.GPU Languages/Frameworks
10.Debugging and Optimizing Code on Titan
10.1.Accelerator Performance Tools

1. Titan System Overview

• [\(Back to Top\)](#)

With a theoretical peak performance of more than 20 petaflops, *Titan*, a Cray XK7 supercomputer located at the Oak Ridge Leadership Computing Facility (OLCF), gives computational scientists unprecedented resolution for studying a whole range of natural phenomena, from climate change to energy assurance, to nanotechnology and nuclear energy.

Compute Partition

Titan contains 18,688 physical compute nodes, each with a processor, physical memory, and a connection to the Cray custom high-speed interconnect. Each compute node contains (1) 16-core 2.2GHz AMD Opteron™ 6274 (Interlagos) processor and (32) GB of RAM. Two nodes share (1) *Gemini*™ high-speed interconnect router. The resulting partition contains 299,008 traditional processor cores, and (598) TB of memory.

Specialized NVIDIA Accelerators

The OLCF provides a comprehensive [Accelerated Computing Guide](#) that covers the use of Titan's NVIDIA Kepler™ accelerators.

In addition to the Opteron CPU, all of Titan's 18,688 physical compute nodes contain an NVIDIA Kepler™ accelerator (GPU) with 6 GB of DDR5 memory.

External Login Nodes

Upon login, users are placed onto login nodes by default. Each Titan login node houses an 8-core AMD Opteron™ 6140-series CPU and (256) GB of RAM.

Network Topology

Nodes within the compute partition are connected in a three-dimensional torus. This provides a very scalable network with low latency and high bandwidth.

File Systems

The OLCF's center-wide Lustre® file system, named [Spider](#), is available on Titan for computational work. Spider contains over 26,000 clients and (32) PB of disk space. A separate, NFS-based file system provides [\\$HOME storage areas](#), and an [HPSS-based file system](#) provides Titan users with archival spaces.

Operating System

Titan employs the Cray Linux Environment as its OS. This consists of a full-featured version of Linux on the login nodes, and a Compute Node Linux microkernel on compute nodes. The microkernel is designed to minimize partition overhead allowing scalable, low-latency global communications.

2. Requesting Access to OLCF Resources

• [\(Back to Top\)](#)

Access to the computational resources of the Oak Ridge Leadership Facility (OLCF) is limited to approved users via project allocations. There are different kinds of projects, and the type of project request will determine the application and review procedure. Approved projects will be granted an allocation of hours for a period of time on one or more systems. Every user account at the OLCF must be associated with at least one allocation. Once an allocation has been approved and established, users can request to be added to the project allocation so they may run jobs against it.

2.1. Project Allocation Requests

• [\(Back to Top\)](#)

The OLCF grants (3) different types of project allocations. The type of allocation you should request depends on a few different factors. The table below outlines the types of project allocations available at the OLCF and the some general policies that apply to each:

	INCITE	Director's Discretion	ALCC
Allocations	Large	Small	Large
Call for Proposals	Once per year	At any time	Once per year
Closeout Report	Required	Required	Required
Duration	1 year	1 year	1 year
Job Priority	High	Medium	High
Quarterly Reports	Required	Required	Required
	Apply for INCITE	Apply for DD	Apply for ALCC

Project Type Details

INCITE – The Novel Computational Impact on Theory and Experiment (INCITE) program invites proposals for large-scale, computationally intensive research projects to run at the OLCF. The INCITE program awards sizeable allocations (typically, millions of processor-hours per project) on some of the world’s most powerful supercomputers to address grand challenges in science and engineering. There is an annual call for INCITE proposals and awards are made on an annual basis. For more information or to apply for an INCITE project, please visit the [DOE INCITE](#) page. **ALCC** – The ASCR Leadership Computing Challenge (ALCC) is open to scientists from the research community in national laboratories,

academia and industry. The ALCC program allocates computational resources at the OLCF for special situations of interest to the Department with an emphasis on high-risk, high-payoff simulations in areas directly related to the Department's energy mission in areas such as advancing the clean energy agenda and understanding the Earth's climate, for national emergencies, or for broadening the community of researchers capable of using leadership computing resources. For more information or to submit a proposal, please visit the [DOE ALCC](#) page. **DD** – Director's Discretion (DD) projects are dedicated to leadership computing preparation, INCITE and ALCC scaling, and application performance to maximize scientific application efficiency and productivity on leadership computing platforms. The OLCF Resource Utilization Council, as well as independent referees, review and approve all DD requests. Applications are accepted year round via the [OLCF Director's Discretion Project Application](#) page.

After Project Approval

Once a project is approved, an OLCF Accounts Manager will notify the PI, outlining the steps (listed below) necessary to create the project. If you have any questions, please feel free to contact the OLCF Accounts Team at accounts@ccs.ornl.gov. Steps for Activating a Project Once the Allocation is Approved

1. A signed Principal Investigator's PI Agreement must be submitted with the project application.
2. Export Control: The project request will be reviewed by ORNL Export Control to determine whether sensitive or proprietary data will be generated or used. The results of this review will be forwarded to the PI. If the project request is deemed sensitive and/or proprietary, the OLCF Security Team will schedule a conference call with the PI to discuss the data protection needs.
3. ORNL Personnel Access System (PAS): All PI's are required to be entered into the ORNL PAS system. An OLCF Accounts Manager will send the PI a PAS invitation to submit all the pertinent information. Please note that processing a PAS request may take 15 or more days.
4. User Agreement/Appendix A or Subcontract: A User Agreement/Appendix A or Subcontract must be executed between UT-Battelle and the PI's institution. If our records indicate this requirement has not been met, all necessary documents will be provided to the applicant by an OLCF Accounts Manager.

Upon completion of the above steps, the PI will be notified that the project has been created and provided with the Project ID and system allocation. At this time, project participants may apply for an account via the [OLCF User Account Application](#) page.

2.2. User Account Requests

• ([Back to Top](#))

Users can apply for an account on existing projects. There are several steps in applying for an account; OLCF User Assistance can help you through the process. If you have any questions, please feel free to contact the Accounts Team at accounts@ccs.ornl.gov.

Steps to Obtain a User Account

1. Apply for an account using the Account Request Form.
2. The principal investigator (PI) of the project must approve your account and system access. The Accounts Team will contact the PI for this approval.
3. If you have or will receive a RSA SecurID from our facility, additional paperwork will be sent to you via email to complete for identity proofing.
4. Foreign national participants will be sent an Oak Ridge National Lab (ORNL) Personnel Access System (PAS) request specific for the facility and cyber-only access. After receiving your response, it takes between (2) to (5) weeks for approval.
5. Fully-executed User Agreements with each institution having participants are required. If our records indicate your institution needs to sign either a User Agreement and/or Appendix A, the form(s) along with instructions will be sent via email.
6. If you are processing sensitive or proprietary data, additional paperwork is required and will be sent to you.

Your account will be created and you will be notified via email when all of the steps above are complete. To begin the process, visit the [OLCF User Account Application](#) page.

3. OLCF Help and Policies

• [\(Back to Top\)](#)

The OLCF provides many tools to assist users, including direct hands-on assistance by trained consultants. Means of assistance at the OLCF include:

- The **OLCF User Assistance Center (UAC)**, where consultants answer your questions directly via email or phone.
- Various OLCF communications, which provide status updates of relevance to end-users.
- The [My OLCF](#) site, which provides a mechanism for viewing project allocation reports.
- The [OLCF Policy Guide](#), which details accepted use of our computational resources.
- Upcoming and historical [OLCF Training Events](#), both in-person and web-based, that cover topics of interest to end-users.

3.1. User Assistance Center

• [\(Back to Top\)](#)

The OLCF User Assistance Center (UAC) provides direct support to users of our computational resources.

Hours

The center’s normal support hours are 9am EST to 5pm EST Monday through Friday, exclusive of holidays.

Contact Us

Email	help@olcf.ornl.gov
Phone:	865-241-6536
Fax:	865-241-4011
Address:	1 Bethel Valley Road, Oak Ridge, TN 37831

The OLCF UAC is located at the Oak Ridge National Laboratory (ORNL) in Building 5600, Room C103.

After Hours

Outside of normal business hours, calls are directed to the ORNL Computer Operations staff. If you require immediate assistance, you may contact them at the phone number listed above. If your request is not urgent, you may send an email to help@olcf.ornl.gov, where it will be answered by a OLCF User Assistance member the next business day.

Ticket Submission Webform

In lieu of sending email, you can also use the [Ticket Submission Web Form](#) to submit a request directly to OLCF User Assistance.

3.2. Communications to Users

• [\(Back to Top\)](#)

The OLCF provides users with several ways of staying informed.

OLCF Announcements Mailing Lists

These mailing lists provides users with email messages of general interest (system upgrades, long-term outages, etc.) Since the mailing frequency is low and the information sent is important to all users, users are automatically subscribed to these lists as applicable when an account is set up.

OLCF "Notice" Mailing Lists

The OLCF also provides opt-in email lists that provide automated notices about the status of OLCF systems as well as other notable system events. Since the mailing frequency of these lists are high, they are offered on an opt-in basis. More information on the lists can be found at the [OLCF Notifications List](#) page.

Weekly Update

Each week, typically on Friday afternoon, an email announcing the next week's scheduled outages is sent to all users. This message also includes meeting announcements and other items of interest to all OLCF users. If you are an OLCF user but are not receiving this weekly message, please contact the [OLCF User Assistance Center](#).

System Status Pages

The [OLCF Main Support](#) page shows the current up/down status of selected OLCF systems at the top.

Twitter

The OLCF posts messages of interest on the [OLCF Twitter Feed](#). We also post tweets specific to system outages on the [OLCF Status Twitter Feed](#).

Message of the Day

In addition to other methods of notification, the system "Message of the Day" (MOTD) that is echoed upon login shows recent system outages. Important announcements are also posted to the MOTD. Users are encouraged to take a look at the MOTD upon login to see if there are any important notices.

3.3. My OLCF Site

• [\(Back to Top\)](#)

To assist users in managing project allocations, we provide end-users with *My OLCF*, a web application with valuable information about OLCF projects and allocations on a per user basis. Users must login to the site with their OLCF username and SecurID fob: <https://users.nccs.gov> Detailed metrics for users and projects can be found in each project's usage section:

- YTD usage by system, subproject, and project member
- Monthly usage by system, subproject, and project member
- YTD usage by job size groupings for each system, subproject, and project member
- Weekly usage by job size groupings for each system, and subproject
- Batch system priorities by project and subproject
- Project members

3.4. Special Requests and Policy Exemptions

• [\(Back to Top\)](#)

Users can request policy exemptions by submitting the appropriate web form available on the OLCF [Documents and Forms](#) page. Special requests forms allow a user to:

- Request Software installations
- Request relaxed queue limits for a job
- Request a system reservation
- Request a disk quota increase
- Request a User Work area purge exemption

Special requests are reviewed weekly and approved or denied by management via the OLCF Resource Utilization Council.

3.5. OLCF Acknowledgement

• [\(Back to Top\)](#)

Users should acknowledge the OLCF in all publications and presentations that speak to work performed on OLCF resources:

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

4. Accessing OLCF Systems

• [\(Back to Top\)](#)

This section covers the basic procedures for accessing OLCF computational resources. To avoid risks associated with using plain-text communication, the only supported remote client on OLCF systems is a secure shell (SSH) client, which encrypts the entire session between OLCF systems and the client system.

Note: To access OLCF systems, your SSH client must support SSH protocol version 2 (this is common) and allow keyboard-interactive authentication.

For UNIX-based SSH clients, the following line should be in either the default `ssh_config` file or your `$HOME/.ssh/config` file:

```
PreferredAuthentications keyboard-interactive,password
```

The line may also contain other authentication methods, but `keyboard-interactive` must be included. SSH clients are also available for Windows-based systems, such as *SecureCRT* published by Van Dyke Software. For recent SecureCRT versions, the preferred authentications change above can be made through the "connection properties" menu.

4.1. OLCF System Hostnames

• [\(Back to Top\)](#)

Each OLCF system has a single, designated hostname for general user-initiated user connections. Sometimes this is a load-balancing mechanism that will send users to other hosts as needed. In any case, the designated OLCF host names for general user connections are as follows:

System Name	Hostname	RSA Key Fingerprint
Titan	<code>titan.ccs.ornl.gov</code>	<code>77:dd:c9:2c:65:2f:c3:89:d6:24:a6:57:26:b5:9b:b7</code>
Rhea	<code>rhea.ccs.ornl.gov</code>	<code>9a:72:79:cf:9e:47:33:d1:91:dd:4d:4e:e4:de:25:33</code>
Eos	<code>eos.ccs.ornl.gov</code>	<code>e3:ae:eb:12:0d:b1:4c:0b:6e:53:40:5c:e7:8a:0d:19</code>
Everest	<code>everest.ccs.ornl.gov</code>	<code>cc:6e:ef:84:7e:7c:dc:72:71:7b:76:7f:f3:46:57:2b</code>
Smoky	<code>smoky.ccs.ornl.gov</code>	<code>e3:88:b9:ba:fe:3a:fd:99:00:24:fc:e6:9d:5c:69:2b</code>
Sith	<code>sith.ccs.ornl.gov</code>	<code>28:63:5e:41:32:39:c2:ec:9b:63:e0:86:16:2f:e4:bd</code>
Data Transfer Nodes	<code>dtn.ccs.ornl.gov</code>	<code>b3:31:ac:44:83:2b:ce:37:cc:23:f4:be:7a:40:83:85</code>
Home (machine)	<code>home.ccs.ornl.gov</code>	<code>12:9b:10:f7:b9:c7:1b:a2:b0:52:5e:13:e2:b9:b2:8c</code>

For example, to connect to Titan from a UNIX-based system, use the following:

```
$ ssh userid@titan.ccs.ornl.gov
```

4.2. General-Purpose Systems

• [\(Back to Top\)](#)

After a user account has been approved and created, the requesting user will be sent an email listing the system(s) to

which the user requested and been given access. In addition to the system(s) listed in the email, all users also have access to the following general-purpose systems:

home.ccs.ornl.gov

Home is a general purpose system that can be used to log into other OLCF systems that are not directly accessible from outside the OLCF network. For example, running the `screen` or `tmux` utility is one common use of Home. Compiling, data transfer, or executing long-running or memory-intensive tasks should never be performed on Home. More information can be found on the [The Home Login Host](#) page.

dtn.ccs.ornl.gov

The *Data Transfer Nodes* are hosts specifically designed to provide optimized data transfer between OLCF systems and systems outside of the OLCF network. More information can be found on the [Employing Data Transfer Nodes](#) page.

HPSS

The *High Performance Storage System (HPSS)* provides tape storage for large amounts of data created on OLCF systems. The HPSS can be accessed from any OLCF system through the `hsi` utility. More information can be found on the [HPSS](#) page.

4.3. X11 Forwarding

• [\(Back to Top\)](#)

Automatic forwarding of the X11 display to a remote computer is possible with the use of SSH and a local X server. To set up automatic X11 forwarding within SSH, you can do (1) of the following:

- Invoke `ssh` on the command line with:

```
$ ssh -X hostname
```

Note that use of the `-x` option (lowercase) will disable X11 forwarding.

- Edit (or create) your `$HOME/.ssh/config` file to include the following line:

```
ForwardX11 yes
```

All X11 data will go through an encrypted channel. The `$DISPLAY` environment variable set by SSH will point to the remote machine with a port number greater than zero. This is normal, and happens because SSH creates a proxy X server on the remote machine for forwarding the connections over an encrypted channel. The connection to the real X server will be made from the local machine.

Warning: Users should not manually set the `$DISPLAY` environment variable for X11 forwarding; a non-encrypted channel may be used in this case.

4.4. RSA Key Fingerprints

• [\(Back to Top\)](#)

Occasionally, you may receive an error message upon logging in to a system such as the following:

```
@@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
```

This can be a result of normal system maintenance that results in a changed RSA public key, or could be an actual security incident. If the RSA fingerprint displayed by your SSH client does not match the OLCF-authorized RSA fingerprint for the machine you are accessing, do not continue authentication; instead, contact help@olcf.ornl.gov.

4.5. Authenticating to OLCF Systems

All OLCF systems currently employ two-factor authentication only. To login to OLCF systems, an RSA SecurID® key fob is required.



Activating a new SecurID® fob

1. Initiate an SSH connection to `username@home.ccs.ornl.gov`.
2. When prompted for a **PASSCODE**, enter the 6-digit code shown on the fob.
3. You will be asked if you are ready to set your **PIN**. Answer with "Y".
4. You will be prompted to enter a PIN. Enter a (4) to (6) digit number you can remember. You will then be prompted to re-enter your PIN.
5. You will then be prompted to wait until the next code appears on your fob and to enter your PASSCODE. When the (6) digits on your fob change, enter your PIN digits followed immediately by the new (6) digits displayed on your fob. Note that any set of (6) digits on the fob can only be "used" once.
6. Your PIN is now set, and your fob is activated and ready for use.

Using a SecurID® fob

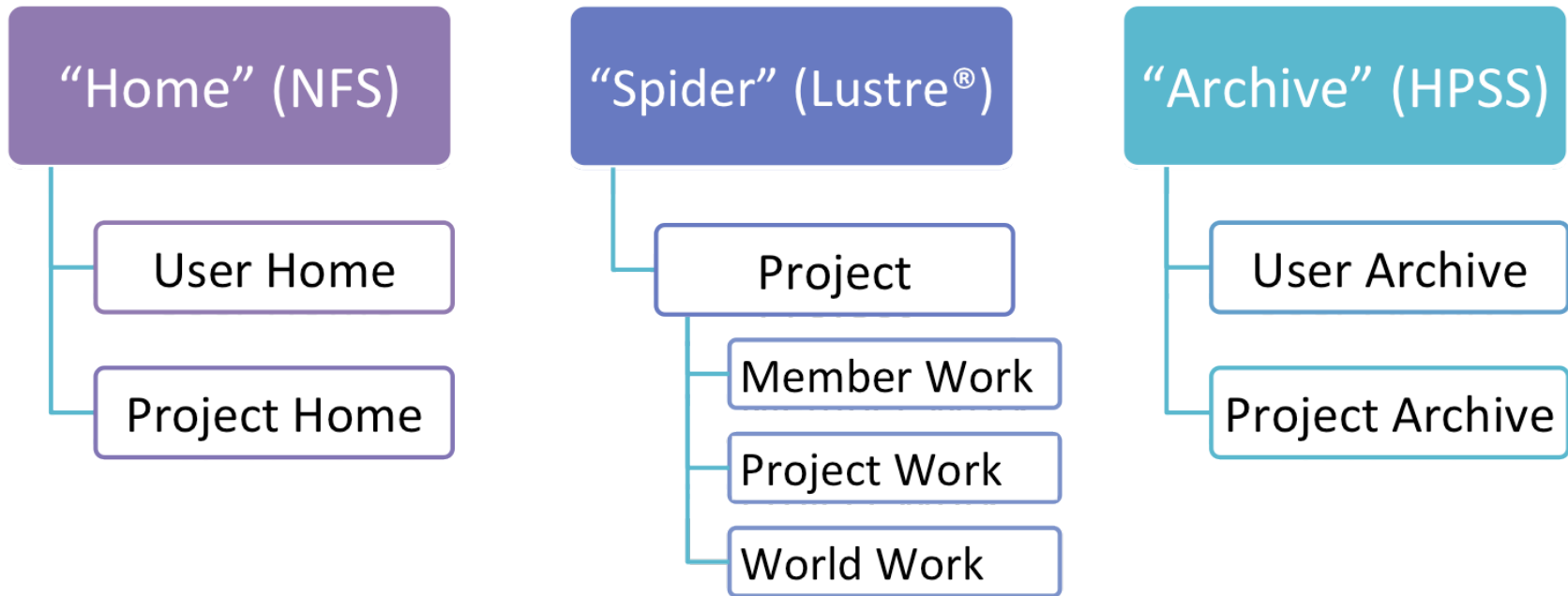
When prompted for your PASSCODE, enter your PIN digits followed immediately by the (6) digits shown on your SecurID® fob. For example, if your pin is `1234` and the (6) digits on the fob are `000987`, enter `1234000987` when you are prompted for a PASSCODE.

5. Data Management

OLCF users have many options for data storage. Each user has a series of user-affiliated storage spaces, and each project has a series of project-affiliated storage spaces where data can be shared for collaboration. The storage areas are mounted across all OLCF systems, making your data available to you from multiple locations.

A Storage Area for Every Activity

The storage area to use in any given situation depends upon the activity you wish to carry out. Each User has a *User Home* area on a Network File System (NFS) and a *User Archive* area on the archival High Performance Storage System (HPSS). User storage areas are intended to house user-specific files. Individual Projects have a *Project Home* area on NFS, multiple *Project Work* areas on Lustre, and a *Project Archive* area on HPSS. Project storage areas are intended to house project-centric files.



Simple Guidelines

The following sections contain a description of all available storage areas and relevant details for each. If you're the impatient type, you can probably get right to work by adhering to the following simple guidelines:

If you need to store...	then use...	at path...
Long-term data for routine access that is unrelated to a project	<i>User Home</i>	<code>\$HOME</code>
Long-term data for archival access that is unrelated to a project	<i>User Archive</i>	<code>/home/\$USER</code>
Long-term project data for routine access that's shared with other project members	<i>Project Home</i>	<code>/ccs/proj/[projid]</code>
Short-term project data for fast, batch-job access that you don't want to share	<i>Member Work</i>	<code>\$MEMBERWORK/[projid]</code>
Short-term project data for fast, batch-job access that's shared with other project members	<i>Project Work</i>	<code>\$PROJWORK/[projid]</code>
Short-term project data for fast, batch-job access that's shared with those outside your project	<i>World Work</i>	<code>\$WORLDWORK/[projid]</code>
Long-term project data for archival access that's shared with other project members	<i>Project Archive</i>	<code>/proj/[projid]</code>

5.1. User-Centric Data Storage

• [\(Back to Top\)](#)

Users are provided with several storage areas, each of which serve different purposes. These areas are intended for storage of data for a particular user and not for storage of project data. The following table summarizes user-centric storage areas available on OLCF resources and lists relevant polices.

User-Centric Storage Areas							
Area	Path	Type	Permissions	Quota	Backups	Purged	Retention
User Home	<code>\$HOME</code>	NFS	User-controlled	10 GB	Yes	No	90 days
User Archive	<code>/home/\$USER</code>	HPSS	User-controlled	2 TB [1]	No	No	90 days

[1] In addition, there is a quota/limit of 2,000 files on this directory.

5.1.1. User Home Directories (NFS)

• [\(Back to Top\)](#)

Each user is provided a home directory to store frequently used items such as source code, binaries, and scripts.

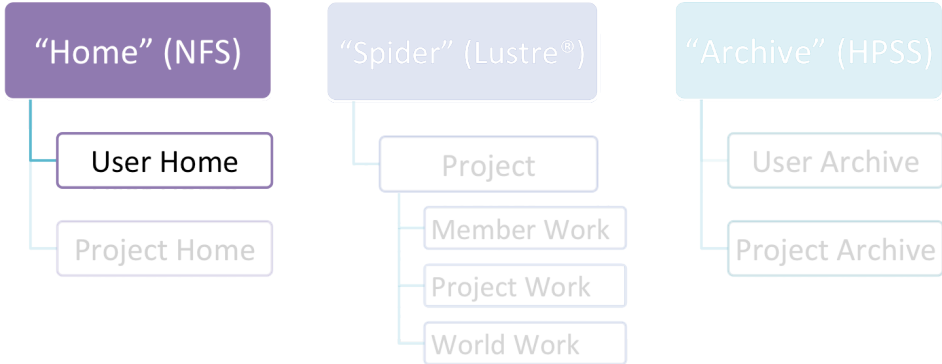
User Home Path

Home directories are located in a Network File Service (NFS) that is accessible from all OLCF resources as `/ccs/home/$USER`. The environment variable `$HOME` will always point to your current home directory. It is recommended, where possible, that you use this variable to reference your home directory. In cases in which using `$HOME` is not feasible, it is recommended that you use `/ccs/home/$USER`. Users should note that since this is an NFS-mounted filesystem, its performance will not be as high as other filesystems.

User Home Quotas

Quotas are enforced on user home directories. To request an increased quota, contact the OLCF User Assistance Center. To view your current quota and usage, use the `quota` command:

```
$ quota -Qs
Disk quotas for user usrid (uid 12345):
    Filesystem blocks quota limit grace files quota limit grace
```



nccsfiler1a.ccs.ornl.gov:/vol/home					
	4858M	5000M	5000M	29379	4295m 4295m

User Home Backups

If you accidentally delete files from your home directory, you may be able to retrieve them. Online backups performed hourly and nightly, with the most recent 6 hours and the most recent 2 nights available. These are available in `/ccs/home/$USER/.snapshot/hourly.*` and `~/.snapshot/nightly.*`. It is possible the files that were deleted will be available in one of those directories. Note that in the directory name, lower numbers represent more recent backups. Thus, `/ccs/home/$USER/.snapshot/hourly.0` is a more recent backup than `/ccs/home/$USER/.snapshot/hourly.1`.

User Home Permissions

The default permissions for user home directories are `0750` (full access to the user, read and execute for the group). Users have the ability to change permissions on their home directories, although it is recommended that permissions be set to as restrictive as possible (without interfering with your work).

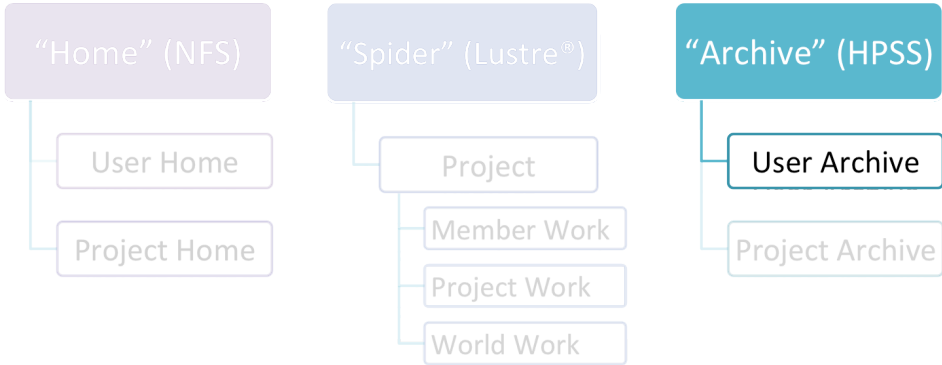
Special User Website Directory

User Home spaces may contain a directory named `/www`. If this directory exists, and if appropriate permissions exist, files in that directory will be accessible via the World Wide Web at `http://users.nccs.gov/~user` (where `user` is your userid).

5.1.2. User Archive Directories (HPSS)

• [\(Back to Top\)](#)

Users are also provided with user-centric archival space on the High Performance Storage System (HPSS). User archive areas on HPSS are intended for storage of data not immediately needed in either User Home directories (NFS) or User Work directories (Lustre®). User Archive areas also serve as a location for users to store backup copies of user files. User Archive directories should not be used to store project-related data. Rather, Project Archive directories should be used for project data.



User Archive Path

User archive directories are located at `/home/$USER`.

User Archive Access

User archive directories may be accessed only via specialized tools called HSI and HTAR. For more information on using HSI or HTAR, see the [HSI and HTAR](#) page.

User Archive Accounting

Each file and directory on HPSS is associated with an HPSS storage allocation. For information on storage allocation, please visit the [Understanding HPSS Storage Allocations](#) page.

5.2. Project-Centric Data Storage

• [\(Back to Top\)](#)

Projects are provided with several storage areas for the data they need. Project directories provide members of a project with a common place to store code, data files, documentation, and other files related to their project. While this information could be stored in one or more user directories, storing in a project directory provides a common location to gather all files.

The following table summarizes project-centric storage areas available on OLCF resources and lists relevant policies.

Project-Centric Storage Areas

Area	Path	Type	Permissions	Quota	Backups	Purged	Retention
Project Home	/ccs/proj/[projid]	NFS	770	50 GB	Yes	No	90 days
Member Work	\$MEMBERWORK/[projid]	Lustre®	700 [1]	10 TB	No	14 days	14 days
Project Work	\$PROJWORK/[projid]	Lustre®	770	100 TB	No	90 days	90 days
World Work	\$WORLDWORK/[projid]	Lustre®	775	10 TB	No	14 days	14 days
Project Archive	/proj/[projid]	HPSS	770	100 TB [2]	No	No	90 days

Important! Files within "Work" directories (i.e., Member Work, Project Work, World Work) are *not* backed up and are *purged* on a regular basis according to the timeframes listed above.

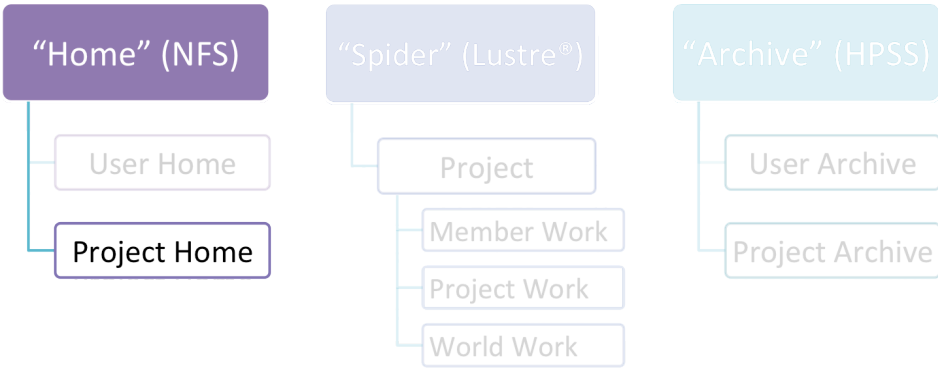
[1] Permissions on Member Work directories can be controlled to an extent by project members. By default, only the project member has any accesses, but accesses can be granted to other project members by setting group permissions accordingly on the Member Work directory. The parent directory of the Member Work directory prevents accesses by "UNIX-others" and cannot be changed (security measures).

[2] In addition, there is a quota/limit of 100,000 files on this directory.

5.2.1. Project Home Directories (NFS)

• (Back to Top)

Projects are provided with a Project Home storage area in the Network File Service (NFS) mounted filesystem. This area is intended for storage of data, code, and other files that are of interest to all members of a project. Since Project Home is an NFS-mounted filesystem, its performance will not be as high as other filesystems.



Project Home Path

Project Home area is accessible at `/ccs/proj/abc123` (where `abc123` is your project ID).

Project Home Quotas

To check your project's current usage, run `df -h /ccs/proj/abc123` (where `abc123` is your project ID). Quotas are enforced on project home directories. The current limit is shown on the [Storage Policy](#) page. To request an increased quota, contact the User Assistance Center.

Project Home Backups

If you accidentally delete one or more files from your project home directory, you may be able to retrieve it/them. Online backups performed hourly and nightly, with the most recent 6 hours and the most recent 2 nights available. These are available in: `/ccs/proj/.snapshot/hourly.*` and `/ccs/proj/.snapshot/nightly.*`. It is possible the files that were deleted will be available in one of those directories. Note that in the directory name, lower numbers represent more recent backups. Thus, `/ccs/proj/.snapshot/hourly.0` is a more recent backup than `/ccs/proj/.snapshot/hourly.1`.

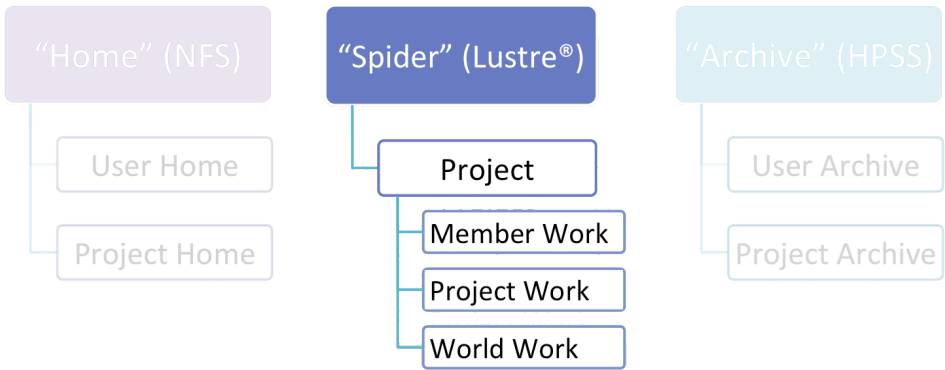
Project Home Permissions

The default permissions for project home directories are `0770` (full access to the user and group). The directory is owned by root and the group is the project's group. All members of a project should also be members of that group-specific project. For example, all members of project "ABC123" should be members of the "abc123" UNIX group.

5.2.2. Project–Centric Work Directories

• [\(Back to Top\)](#)

To provide projects and project members with high-performance storage areas that are accessible to batch jobs, projects are given (3) distinct project-centric work (i.e., scratch) storage areas within [Spider](#), the OLCF's center-wide Lustre® filesystem.



Three Project Work Areas to Facilitate Collaboration

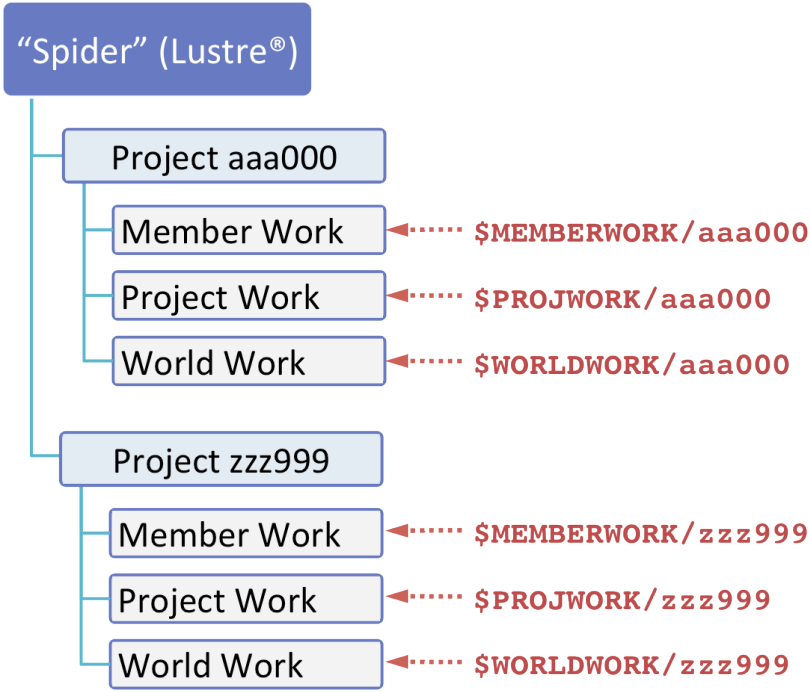
To facilitate collaboration among researchers, the OLCF provides (3) distinct types of project-centric work storage areas: *Member Work* directories, *Project Work* directories, and *World Work* directories. Each directory should be used for storing files generated by computationally-intensive HPC jobs related to a project. The difference between the three lies in the accessibility of the data to project members and to researchers outside of the project. Member Work directories are accessible only by an individual project member by default. Project Work directories are accessible by all project members. World Work directories are readable by any user on the system.

Paths

Paths to the various project-centric work storage areas are simplified by the use of environment variables that point to the proper directory on a per-user basis:

- Member Work Directory: `$MEMBERWORK/[projid]`
- Project Work Directory: `$PROJWORK/[projid]`
- World Work Directory: `$WORLDWORK/[projid]`

Environment variables provide operational staff (aka "us") flexibility in the exact implementation of underlying directory paths, and provide researchers (aka "you") with consistency over the long-term. For these reasons, we highly recommend the use of these environment variables for all scripted commands involving work directories.



Permissions

UNIX Permissions on each project-centric work storage area differ according to the area's intended collaborative use. Under this setup, the process of sharing data with other researchers amounts to simply ensuring that the data resides in the proper work directory.

- Member Work Directory: `700`
- Project Work Directory: `770`
- World Work Directory: `775`

For example, if you have data that must be restricted only to yourself, keep them in your Member Work directory for that project (and leave the default permissions unchanged). If you have data that you intend to share with researchers within your project, keep them in the project's Project Work directory. If you have data that you intend to share with researchers outside of a project, keep them in the project's World Work directory.

Quotas

Soft quotas are enforced on project-centric work directories. The current limit is shown on the [Storage Policy](#) page. To request an increased quota, contact the User Assistance Center.

Backups

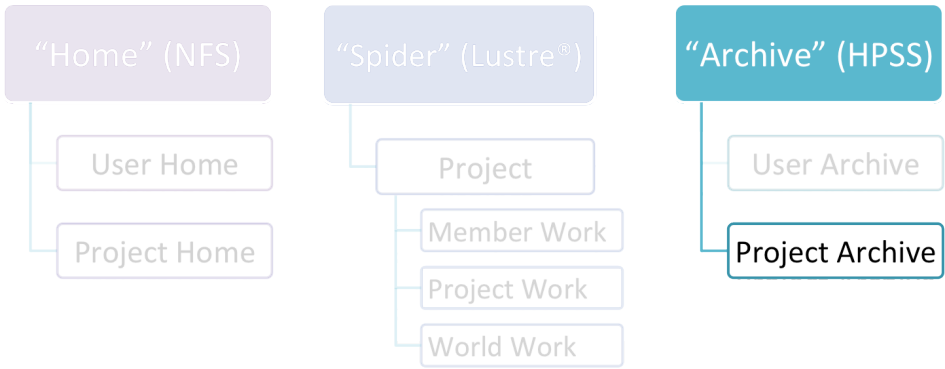
Member Work, Project Work, and World Work directories **are not backed up**. Project members are responsible for backing up these files, either to Project Archive areas (HPSS) or to an off-site location.

5.2.3. Project Archive Directories (HPSS)

• [\(Back to Top\)](#)

Projects are also allocated project-specific archival space on the High Performance Storage System (HPSS). The default quota is shown on the [Storage Policy](#) page. If a higher quota is needed, contact the User Assistance Center. The Project Archive space on HPSS is intended for storage of data not immediately needed in either Project Home (NFS) areas nor

Project Work (Lustre®) areas, and to serve as a location to store backup copies of project-related files.



Project Archive Path

The project archive directories are located at `/proj/pjt000` (where `pjt000` is your Project ID).

Project Archive Access

Project Archive directories may only be accessed via utilities called HSI and HTAR. For more information on using HSI or HTAR, see the [HSI and HTAR](#) page.

Project Archive Accounting

Each file and directory on HPSS is associated with an HPSS storage allocation. For information on HPSS storage allocations, please visit the [Understanding HPSS Storage Allocations](#) page.

5.3. Transferring Data

• [\(Back to Top\)](#)

OLCF users are provided with several options for transferring data among systems at the OLCF as well as between the OLCF and other sites.

Data Transfer Nodes

Dedicated data transfer nodes are provided to OLCF users and are accessible via the load-balancing hostname `dtn.ccs.ornl.gov`. The nodes have been tuned specifically for wide area data transfers, and also perform well on the local area. They are recommended for data transfers as they will, in most cases, improve transfer speed and help decrease load on computational systems’ login nodes. More information on these nodes can be found on the [Data Transfer Nodes](#) page.

Local Transfers

The OLCF provides a shared-storage environment, so transferring data between our machines is largely unnecessary. However, we provide tools both to move large amounts of data between scratch and archival storage and from one scratch area to another. More information can be found on the [local transfers](#) page.

Remote Transfers

The OLCF provides several tools for moving data between computing centers or between OLCF machines and local user workstations. The following tools are primarily designed for transfers over the internet, and aren't recommended for use transferring data between OLCF machines. The following table summarizes options for remote data transfers:

	GridFTP + GridCert	GridFTP + SSH	SFTP/SCP	BBCP
Data Security	insecure (default) / secure (w/configuration)	insecure (default) / secure (w/configuration)	secure	insecure (unsuited for sensitive projects)
Authentication	GridCert	Passcode	Passcode	Passcode
Transfer speed	Fast	Fast	Slow	Fast
Required Infrastructure	GridFTP server at remote site + user OSG Cert	GridFTP server at remote site	Comes standard with SSH install	BBCP installed at remote site

GridFTP

GridFTP is a high-performance data transfer protocol based on FTP and optimized for high-bandwidth wide-area networks. It is typically used to move large amounts of data between the OLCF and other majors centers. More information can be found on the [GridFTP](#) page.

SFTP and SCP

The SSH-based **SFTP** and **SCP** utilities can be used to transfer files to and from OLCF systems. Because these utilities can be slow, we recommend using them only to transfer limited numbers of small files. More information on these utilities can be found on the [SFTP and SCP](#) page.

BBCP

For larger files, the multi-streaming transfer utility **BBCP** is recommended. The **BBCP** utility is capable of breaking up your transfer into multiple simultaneously transferring streams, thereby transferring data much faster than single-streaming utilities such as **SCP** and **SFTP**. Note: **BBCP** is not secure, but is much faster than **SFTP**. More information can be found on the [BBCP](#) page.

5.4. Employing Data Transfer Nodes

• [\(Back to Top\)](#)

The OLCF provides nodes dedicated to data transfer that are available via `dtm.ccs.ornl.gov`. These nodes have been tuned specifically for wide-area data transfers, and also perform well on local-area transfers. The OLCF recommends that users employ these nodes for data transfers, since in most cases transfer speed improves and load on computational systems' login and service nodes is reduced.

Filesystems Accessible from DTNs

All OLCF filesystems -- the NFS-backed User Home and Project Home areas, the Lustre®-backed User Work and Project Work areas, and the HPSS-backed User Archive and Project Archive areas -- are accessible to users via the DTNs. For more information on available filesystems at the OLCF see the [Data Management Overview](#) page.

Interactive DTN Access

Members of allocated projects are automatically given access to the data transfer nodes. There are currently four interactive nodes named, dtm01-dtm04 each with the `.css.ornl.gov` suffix. Information on establishing a connection to `dtm.ccs.ornl.gov`, and other OLCF systems can be found on the [OLCF Host Names](#) page. Only dtm03 and dtm04 are setup to use Office of Science Grid (OSG) Certificate authentication and only dtm01 and dtm02 are set up to use DOE grid certificate authentication. For more on data transfer with grid certificate authentication see the [Transferring Data with Grid FTP](#) page.

Batch DTN Access

Data transfer nodes can be accessed from Titan's batch system by remote job submission. This is accomplished by the command `qsub -q host script.pbs` which will submit the file `script.pbs` to the batch queue on the specified host. This command can be inserted at the end of an existing batch script in order to automatically trigger work on another OLCF resource.

Note: DTNs can help you manage your allocation hours efficiently by preventing billable compute resources from sitting idle.

The following scripts show how this technique could be employed. Note that only the first script, `retrieve.pbs`, would need to be manually submitted; the others will trigger automatically from within the respective batch scripts.

Example Workflow Using DTNs in Batch Mode

The first batch script, `retrieve.pbs`, retrieves data needed by a compute job. Once the data has been retrieved, the script submits a different batch script, `compute.pbs`, to run computations on the retrieved data. **To run this script start on Titan or Rhea.**

```
qsub -q dtm retrieve.pbs
```

```
$ cat retrieve.pbs
```

```
# Batch script to retrieve data from HPSS via DTNs

# PBS directives
#PBS -A PROJ123
#PBS -l walltime=8:00:00

# Retrieve required data
cd $MEMBERWORK/proj123
hsi get largedatfileA
hsi get largedatafileB

# Verification code could go here

# Submit other batch script to execute calculations on retrieved data
qsub -q titan compute.pbs

$
```

The second batch script is submitted from the first to carry out computational work on the data. When the computational work is finished, the batch script `backup.pbs` is submitted to archive the resulting data.

```
$ cat compute.pbs

# Batch script to carry out computation on retrieved data

# PBS directives
#PBS -l walltime=24:00:00
#PBS -l nodes=10000
#PBS -A PROJ123
#PBS -l gres=widow3

# Launch executable
cd $MEMBERWORK/proj123
aprun -n 160000 ./a.out

# Submit other batch script to transfer resulting data to HPSS
qsub -q dtn backup.pbs

$
```

The final batch script is submitted from the second to archive the resulting data soon after creation to HPSS via the `hsi` utility.

```
$ cat backup.pbs

# Batch script to back-up resulting data

# PBS directives
#PBS -A PROJ123
#PBS -l walltime=8:00:00

# Store resulting data
cd $MEMBERWORK/proj123
hsi put largedatfileC
hsi put largedatafileD

$
```

Some items to note:

- Batch jobs submitted to the `dtn` partition will be executed on a DTN that is accessible exclusively via batch submissions. These batch-accessible DTNs have identical configurations to those DTNs that are accessible interactively; the only difference between the two is accessibility.
- The DTNs are not currently a billable resource, i.e., the project specified in a batch job targeting the `dtn` partition will not be charged for time spent executing in the `dtn` partition.
- The walltime limit for jobs submitted to the `dtn` partition is 12 hours.

5.5. Data Management Policy Summary

• [\(Back to Top\)](#)

Users must agree to the full [Data Management Policy](#) as part of their account application. The "Data Retention, Purge, & Quotas" section is useful and is summarized below.

Data Retention, Purge, & Quota Summary

User-Centric Storage Areas							
Area	Path	Type	Permissions	Quota	Backups	Purged	Retention
User Home	\$HOME	NFS	User-controlled	10 GB	Yes	No	90 days
User Archive	/home/\$USER	HPSS	User-controlled	2 TB [1]	No	No	90 days
Project-Centric Storage Areas							
Area	Path	Type	Permissions	Quota	Backups	Purged	Retention
Project Home	/ccs/proj/[projid]	NFS	770	50 GB	Yes	No	90 days
Member Work	\$MEMBERWORK/[projid]	Lustre®	700 [2]	10 TB	No	14 days	14 days
Project Work	\$PROJWORK/[projid]	Lustre®	770	100 TB	No	90 days	90 days
World Work	\$WORLDWORK/[projid]	Lustre®	775	10 TB	No	14 days	14 days
Project Archive	/proj/[projid]	HPSS	770	100 TB [3]	No	No	90 days

Area	The general name of storage area.
Path	The path (symlink) to the storage area's directory.
Type	The underlying software technology supporting the storage area.
Permissions	UNIX Permissions enforced on the storage area's top-level directory.
Quota	The limits placed on total number of bytes and/or files in the storage area.
Backups	States if the data is automatically duplicated for disaster recovery purposes.
Purged	Period of time, post-file-creation, after which a file will be marked as eligible for permanent deletion.
Retention	Period of time, post-account-deactivation or post-project-end, after which data will be marked as eligible for permanent deletion.

Important! Files within "Work" directories (i.e., Member Work, Project Work, World Work) are *not* backed up and are *purged* on a regular basis according to the timeframes listed above.

[1] In addition, there is a quota/limit of 2,000 files on this directory.

[2] Permissions on Member Work directories can be controlled to an extent by project members. By default, only the project member has any accesses, but accesses can be granted to other project members by setting group permissions accordingly on the Member Work directory. The parent directory of the Member Work directory prevents accesses by "UNIX-others" and cannot be changed (security measures).

[3] In addition, there is a quota/limit of 100,000 files on this directory.

6. Software and Shell Environments

• [\(Back to Top\)](#)

The OLCF provides hundreds of pre-installed software packages and scientific libraries for your use, in addition to taking software requests. Due to the large number of software packages and versions on OLCF resources, environment management tools are needed to handle changes to your shell environment. This chapter discusses how to manage your shell and software environment on OLCF systems.

6.1. Default Shell

• [\(Back to Top\)](#)

Users request their preferred shell on their initial user account request form. The default shell is enforced across all OLCF resources. The OLCF currently supports the following shells:

- bash
- tsch
- csh
- ksh

Please contact the OLCF User Assistance Center to request a different default shell.

6.2. Using Modules

• [\(Back to Top\)](#)

The *modules* software package allows you to dynamically modify your user environment by using pre-written *modulefiles*.

Modules Overview

Each modulefile contains the information needed to configure the shell for an application. After the modules software package is initialized, the environment can be modified on a per-module basis using the `module` command, which interprets a modulefile. Typically, a modulefile instructs the module command to alter or set shell environment variables such as `PATH` or `MANPATH`. Modulefiles can be shared by many users on a system, and users can have their own personal collection to supplement and/or replace the shared modulefiles. As a user, you can add and remove modulefiles from your current shell environment. The environment changes performed by a modulefile can be viewed by using the `module` command as well. More information on modules can be found by running `man module` on OLCF systems.

Summary of Module Commands

Command	Description
<code>module list</code>	Lists modules currently loaded in a user’s environment
<code>module avail</code>	Lists all available modules on a system in condensed format
<code>module avail -l</code>	Lists all available modules on a system in long format
<code>module display</code>	Shows environment changes that will be made by loading a given module
<code>module load</code>	Loads a module
<code>module unload</code>	Unloads a module
<code>module help</code>	Shows help for a module
<code>module swap</code>	Swaps a currently loaded module for an unloaded module

Re-initializing the Module Command

Modules software functionality is highly dependent upon the shell environment being used. Sometimes when switching between shells, modules must be re-initialized. For example, you might see an error such as the following:

```
$ module list
-bash: module: command not found
```

To fix this, just re-initialize your modules environment:

```
$ source $MODULESHOME/init/myshell
```

Where `myshell` is the name of the shell you are using and need to re-initialize.

Examples of Module Use

To show all available modules on a system:

```
$ module avail
```



```
----- /opt/cray/modulefiles -----
atp/1.3.0                                netcdf/4.1.3                                tpsl/1.0.01
atp/1.4.0(default)                      netcdf-hdf5parallel/4.1.2(default) tpsl/1.1.01(default)
atp/1.4.1                                netcdf-hdf5parallel/4.1.3
trilinos/10.6.4.0(default)
...
```

To search for availability of a module by name:

```
$ module avail -l netcdf
- Package -----+- Versions +- Last mod. -----
/opt/modulefiles:
netcdf/3.6.2                                2009/09/29 16:38:25
/sw/xk6/modulefiles:
netcdf/3.6.2                                2011/12/09 18:07:31
netcdf/4.1.3                                default    2011/12/12 20:43:37
...
```

To show the modulefiles currently in use (loaded) by the user:

```
$ module list
Currently Loaded Modulefiles:
  1) modules/3.2.6.6                                12) pmi/3.0.0-1.0000.8661.28.2807.gem
  2) xe-sysroot/4.0.30.securitypatch.20110928      13) ugni/2.3-1.0400.3912.4.29.gem
  3) xtpe-network-gemini                          14) udreg/2.3.1-1.0400.3911.5.6.gem
```

To show detailed help info on a modulefile:

```
$ module help netcdf/4.1.3
----- Module Specific Help for 'netcdf/4.1.3' -----
Purpose:
  New version of hdf5 1.8.7 and netcdf 4.1.3
Product and OS Dependencies:
  hdf5_netcdf 2.1 requires SLES 11 systems and was tested on Cray XE and
...
```

To show what a modulefile will do to the shell environment if loaded:

```
$ module display netcdf/4.1.3
-----
/opt/cray/modulefiles/netcdf/4.1.3:
setenv          CRAY_NETCDF_VERSION 4.1.3
prepend-path    PATH /opt/cray/netcdf/4.1.3/gnu/45/bin
...
```

To load or unload a modulefile

```
$ module load netcdf/4.1.3
$ module unload netcdf/4.1.3
```

To unload a modulefile and load a different one:

```
$ module swap netcdf/4.1.3 netcdf/4.1.2
```

6.3. Installed Software

• [\(Back to Top\)](#)

The OLCF provides hundreds of pre-installed software packages and scientific libraries for your use, in addition to taking software installation requests. See the [software section](#) for complete details on existing installs. To request a new software install, use the [software installation request](#) form.

7. Compiling On Titan

• [\(Back to Top\)](#)

For GPU specific compilation details please see the [Accelerated Computing Guide](#).

Compiling code on Titan (and other Cray machines) is different than compiling code for commodity or beowulf-style HPC linux clusters. Among the most prominent differences:

- Cray provides a sophisticated set of compiler wrappers to ensure that the compile environment is setup correctly. Their use is highly encouraged.
- In general, linking/using shared object libraries on compute partitions is not supported.
- Cray systems include many different types of nodes, so some compiles are, in fact, cross-compiles.

Available Compilers

The following compilers are available on Titan:

- [PGI](#), the Portland Group Compiler Suite (default)
- [GCC](#), the GNU Compiler Collection
- [CCE](#), the Cray Compiling Environment
- [Intel](#), Intel Composer XE

7.1. Cray Compiler Wrappers

• [\(Back to Top\)](#)

Cray provides a number of compiler wrappers that substitute for the traditional compiler invocation commands. The wrappers call the appropriate compiler, add the appropriate header files, and link against the appropriate libraries based on the currently loaded programming environment module. To build codes for the compute nodes, you should invoke the Cray wrappers via:

- `cc` To use the C compiler
- `CC` To use the C++ compiler
- `ftn` To use the FORTRAN 90 compiler

7.2. Compiling and Node Types

• [\(Back to Top\)](#)

Titan is comprised of different types of nodes:

- *Login* nodes running traditional Linux
- *Service* nodes running traditional Linux
- *Compute* nodes running the Cray Node Linux (CNL) microkernel

The type of work you are performing will dictate the type of node for which you build your code.

Compiling for Compute Nodes

Titan compute nodes are the nodes that carry out the vast majority of computation on the system. Compute nodes are running the CNL microkernel, which is markedly different than the OS running on the login and service nodes. Most code that runs on Titan will be built targeting the compute nodes. All parallel codes should run on the compute nodes. Compute nodes are accessible only by invoking `aprun` within a batch job. To build codes for the compute nodes, you should use the Cray compiler wrappers.

Note: The OLCF *highly* recommends that the Cray-provided `cc`, `CC`, and `ftn` compiler wrappers be used when compiling and linking source code for use on Titan compute nodes.

Support for Shared Object Libraries

On Titan, and Cray machines in general, statically linked executables will perform better and are easier to launch. Depending on the module files you load, certain Cray-provided modules and libraries (such as `mpich2` and `cudart`) may employ and configure dynamic linking automatically; the following warnings do not apply to them.

Warning: In general, the use of shared object libraries is *strongly discouraged* on Cray compute nodes. This excludes certain Cray-provided modules and libraries such as `mpich2` and `cudart`. Any necessary

dynamic linking of these libraries will be configured automatically by the Cray compiler wrappers.

If you must use shared object libraries, you will need to copy all necessary libraries to `$MEMBERWORK/[projid]` (or your `$PROJWORK` project scratch area) and then update your `LD_LIBRARY_PATH` environment variable to include this directory. For example, the following command will append your user scratch area to the `LD_LIBRARY_PATH` in bash: `$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MEMBERWORK[projid]` Compiling with shared libraries can be further complicated by the fact that Titan's login nodes do not run the same operating system as the compute nodes, and thus many shared libraries are available on the login nodes which are not available on the compute nodes. This means that an executable may appear to compile correctly on a login node, but will fail to start on a compute node because it will be unable to locate the shared libraries it needs. It may appear that this could be resolved by locating the shared libraries on the login node and copying them to Lustre for use on the compute nodes. This is inadvisable because these shared libraries were not compiled for the compute nodes, and may perform erratically. Also, referring directly to these libraries circumvents the `module` system, and may jeopardize your deployment environment in the event of system upgrades. For performance considerations, it is important to bear in mind that each node in your job will need to search through `$LD_LIBRARY_PATH` for each missing dynamic library, which could cause a bottleneck with the Lustre Metadata Server. Lastly, calls to functions in dynamic libraries will not benefit from compiler optimizations that are available when using static libraries.

Compiling for Login or Service Nodes

When you log into Titan you are placed on a login node. When you submit a job for execution, your job script is initially launched on one of a small number of shared service nodes. All tasks not launched through `aprun` will run on the service node. Users should note that there are a small number of these login and service nodes, and they are shared by all users. Because of this, long-running or memory-intensive work should not be performed on login nodes nor service nodes.

Warning: Long-running or memory-intensive codes should not be compiled for use on login nodes nor service nodes.

When using `cc`, `CC`, or `ftn` your code will be built for the compute nodes by default. If you wish to build code for the Titan login nodes or service nodes, you must do *one* of the following:

1. `module swap craype-interlagos craype-target-native` (this is the preferred option, especially when trying to use a configure script)
2. Add the `-target=native` flag to your `cc`, `CC`, or `ftn` command
3. Call the underlying compilers directly (e.g. `pgf90`, `ifort`, `gcc`)

XK7 Service/Compute Node Incompatibilities

On the Cray XK7 architecture, service nodes differ greatly from the compute nodes. The difference between XK7 compute and service nodes may cause cross compiling issues that did not exist on Cray XT5 systems and prior. For XK7, login and service nodes use AMD's Istanbul-based processor, while compute nodes use the newer Interlagos-based processor. Interlagos-based processors include instructions not found on Istanbul-based processors, so executables compiled for the compute nodes will not run on the login nodes nor service nodes; typically crashing with an illegal instruction error. Additionally, codes compiled specifically for the login or service nodes will not run *optimally* on the compute nodes.

Warning: Executables compiled for the XK7 compute nodes will not run on the XK7 login nodes nor XK7 service nodes.

Optimization Target Warning

Because of the difference between the login/service nodes (on which code is built) and the compute nodes (on which code is run), a software package's build process may inject optimization flags incorrectly targeting the login/service nodes. Users are strongly urged to check makefiles for CPU-specific optimization flags (ex: `-tp`, `-hcpu`, `-march`). Users should not need to set such flags; the Cray compiler wrappers will automatically add CPU-specific flags to the build. Choosing the incorrect processor optimization target can negatively impact code performance.

7.3. Controlling the Programming Environment

• [\(Back to Top\)](#)

Upon login, the default versions of the PGI compiler and associated Message Passing Interface (MPI) libraries are added to each user's environment through a *programming environment* module. Users do not need to make any environment changes to use the default version of PGI and MPI.

Changing Compilers

If a different compiler is required, it is important to use the correct environment for each compiler. To aid users in pairing the correct compiler and environment, programming environment modules are provided. The programming environment modules will load the correct pairing of compiler version, message passing libraries, and other items required to build and run. We highly recommend that the programming environment modules be used when changing compiler vendors. The following programming environment modules are available on Titan:

- PrgEnv-pgi
- PrgEnv-gnu
- PrgEnv-cray
- PrgEnv-intel

To change the default loaded PGI environment to the default GCC environment use:

```
$ module unload PrgEnv-pgi
$ module load PrgEnv-gnu
```

Or alternatively:

```
$ module swap PrgEnv-pgi PrgEnv-gnu
```

Changing Versions of the Same Compiler

To use a specific compiler *version*, you must first ensure the compiler's PrgEnv module is loaded, and *then* swap to the correct compiler version. For example, the following will configure the environment to use the GCC compilers, then load a non-default GCC compiler version:

```
$ module swap PrgEnv-pgi PrgEnv-gnu
$ module swap gcc gcc/4.6.1
```

General Programming Environment Guidelines

We recommend the following general guidelines for using the programming environment modules:

- Do not purge all modules; rather, use the default module environment provided at the time of login, and modify it.
- Do not swap or unload any of the Cray provided modules (those with names like `xt-*`, `xe-*`, `xk-*`, or `cray-*`).
- Do not swap moab, torque, or MySQL modules after loading a programming environment modulefile.

7.4. Compiling Threaded Codes

• [\(Back to Top\)](#)

When building threaded codes on Cray machines, you may need to take additional steps to ensure a proper build.

OpenMP

For PGI, add "-mp" to the build line.

```
$ cc -mp test.c -o test.x
$ setenv OMP_NUM_THREADS 2
$ aprun -n2 -d2 ./test.x
```

For GNU, add "-fopenmp" to the build line.

```
$ cc -fopenmp test.c -o test.x
$ setenv OMP_NUM_THREADS 2
```

```
$ aprun -n2 -d2 ./test.x
```

For Cray, no additional flags are required.

```
$ module swap PrgEnv-pgi PrgEnv-cray
$ cc test.c -o test.x
$ setenv OMP_NUM_THREADS 2
$ aprun -n2 -d2 ./test.x
```

For Intel, -fopenmp

```
$ module swap PrgEnv-pgi PrgEnv-intel
$ cc test.c -o test.x
$ setenv OMP_NUM_THREADS 2
$ aprun -n2 -d2 ./test.x
```

SHMEM

For SHMEM codes, users must load the `xt-shmem` module before compiling:

```
$ module load xt-shmem
```

8. Running Jobs on Titan

• [\(Back to Top\)](#)

In High Performance Computing (HPC), computational work is performed by *jobs*. Individual jobs produce data that lend relevant insight into grand challenges in science and engineering. As such, the timely, efficient execution of jobs is the primary concern in the operation of any HPC system. A job on Titan typically comprises a few different components:

- A batch submission script.
- A statically-linked binary executable.
- A set of input files for the executable.
- A set of output files created by the executable.

And the process for running a job, in general, is to:

1. Prepare executables and input files.
2. Write a batch script.
3. Submit the batch script to the batch scheduler.
4. Optionally monitor the job before and during execution.

The following sections describe in detail how to create, submit, and manage jobs for execution on Titan.

8.1. Login vs. Service vs. Compute Nodes

• [\(Back to Top\)](#)

Cray Supercomputers are complex collections of different types of physical nodes/machines. For simplicity, we can think of Titan nodes as existing in one of three categories: *login* nodes, *service* nodes, or *compute* nodes.

Login Nodes

Login nodes are designed to facilitate ssh access into the overall system, and to handle simple tasks. When you first log in, you are placed on a login node. Login nodes are shared by all users of a system, and should only be used for basic tasks such as file editing, code compilation, data backup, and job submission. Login nodes should not be used for memory-intensive nor processing-intensive tasks. Users should also limit the number of simultaneous tasks performed on login nodes. For example, a user should not run ten simultaneous tar processes.

Warning: Processor-intensive, memory-intensive, or otherwise disruptive processes running on login nodes may be killed without warning.

Service Nodes

Memory-intensive tasks, processor-intensive tasks, and any production-type work should be submitted to the machine's batch system (e.g. to Torque/MOAB via `qsub`). When a job is submitted to the batch system, the job submission script is first executed on a *service node*. Any job submitted to the batch system is handled in this way, including interactive batch jobs (e.g. via `qsub -I`). Often users are under the (false) impression that they are executing commands on compute nodes while typing commands in an interactive batch job. On Cray machines, this is not the case.

Compute Nodes

On Cray machines, when the `aprun` command is issued within a job script (or on the command line within an interactive batch job), the binary passed to `aprun` is copied to and executed in parallel on a set of compute nodes. Compute nodes run a Linux microkernel for reduced overhead and improved performance.

Note: On Cray machines, the **only** way to access the compute nodes is via the `aprun` command.

8.2. Filesystems Available to Compute Nodes

• [\(Back to Top\)](#)

Only User Work (Lustre®) and Project Work (Lustre®) storage areas are available to compute nodes on OLCF Cray systems. Other storage spaces (User Home, User Archive, Project Home, and Project Archive) are not mounted on compute nodes.

Warning: Only User Work (Lustre) and Project Work (Lustre) storage areas are available to compute nodes.

As a result, job executable binaries and job input files must reside within a Lustre Work space., e.g. `$MEMBERWORK/[projid]`. Job output must also be sent to a Lustre Work space. Batch jobs can be submitted from User Home or Project Home, but additional steps are required to ensure the job runs successfully. Jobs submitted from Home areas should `cd` into a Lustre Work directory prior to invoking `aprun`. An error like the following may be returned if this is not done:

```
aprun: [NID 94]Exec /lustre/atlas/scratch/userid/a.out failed: chdir /autofs/na1_home/userid
No such file or directory
```

8.3. Writing Batch Scripts

• [\(Back to Top\)](#)

Batch scripts, or job submission scripts, are the mechanism by which a user submits and configures a job for eventual execution. A batch script is simply a shell script which contains:

- Commands that can be interpreted by batch scheduling software (e.g. *PBS*)
- Commands that can be interpreted by a shell

The batch script is submitted to the batch scheduler where it is parsed. Based on the parsed data, the batch scheduler places the script in the scheduler queue as a batch job. Once the batch job makes its way through the queue, the script will be executed on a service node within the set of allocated computational resources.

Sections of a Batch Script

Batch scripts are parsed into the following three sections:

1. The Interpreter Line

The first line of a script can be used to specify the script's interpreter. This line is optional. If not used, the submitter's default shell will be used. The line uses the "hash-bang-shell" syntax: `#!/path/to/shell`

2. The Scheduler Options Section

The batch scheduler options are preceded by `#PBS`, making them appear as comments to a shell. PBS will look for `#PBS` options in a batch script from the script's first line through the first non-comment line. A comment line begins with `#`. `#PBS` options entered after the first non-comment line will not be read by PBS.

Note: All batch scheduler options must appear at the beginning of the batch script.

3. The Executable Commands Section

The shell commands follow the last `#PBS` option and represent the main content of the batch job. If any `#PBS` lines follow executable statements, they will be ignored as comments.

The execution section of a script will be interpreted by a shell and can contain multiple lines of executable invocations, shell commands, and comments. When the job's queue wait time is finished, commands within this section will be executed on a service node (sometimes called a "head node") from the set of the job's allocated resources. Under normal circumstances, the batch job will exit the queue after the last line of the script is executed.

An Example Batch Script

```
1: #!/bin/bash
2: #      Begin PBS directives
3: #PBS -A pjt000
4: #PBS -N test
5: #PBS -j oe
6: #PBS -l walltime=1:00:00,nodes=1500
7: #PBS -l gres=widow2%widow3
8: #      End PBS directives and begin shell commands
9: cd $MEMBERWORK/pjt000
10: date
11: aprun -n 24000 ./a.out
```

The lines of this batch script do the following:

Line	Option	Description
1	Optional	Specifies that the script should be interpreted by the bash shell.
2	Optional	Comments do nothing.
3	Required	The job will be charged to the "pjt000" project.
4	Optional	The job will be named "test".
5	Optional	The job's standard output and error will be combined.
6	Required	The job will request 1,500 compute nodes for 1 hour.
7	Optional	The job will be associated with the widow2 and widow3 Lustre® filesystems only.
8	Optional	Comments do nothing.
9	--	This shell command will the change to the user's \$MEMBERWORK/pjt000 directory.
10	--	This shell command will run the <code>date</code> command.
11	--	This invocation will run 24,000 MPI instances of the executable <code>a.out</code> on the compute nodes allocated by the batch system.

Note: For more batch script examples, please see the [Batch Script Examples](#) page.

Additional Example Batch Scripts

For more batch script examples, please see the [Batch Script Examples](#) page.

Batch Scheduler node Requests

A node's cores cannot be allocated to multiple jobs. Because the OLCF charges based upon the computational resources a job makes *unavailable* to others, a job is charged for an entire node even if the job uses only one processor core. To simplify the process, users are required to request an entire node through PBS.

Note: Whole nodes must be requested at the time of job submission, and allocations are reduced by core-hour amounts corresponding to whole nodes, regardless of actual core utilization.

8.4. Submitting Batch Scripts

• [\(Back to Top\)](#)

Once written, a batch script is submitted to the batch scheduler via the `qsub` command.

```
$ cd /path/to/batch/script
$ qsub ./script.pbs
```

If successfully submitted, a PBS job ID will be returned. This ID is needed to monitor the job's status with various job monitoring utilities. It is also necessary information when troubleshooting a failed job, or when asking the OLCF User Assistance Center for help.

Note: Always make a note of the returned job ID upon job submission, and include it in help requests to the OLCF User Assistance Center.

Options to the `qsub` command allow the specification of attributes which affect the behavior of the job. In general, options to `qsub` on the command line can also be placed in the batch scheduler options section of the batch script via `#PBS`. For more information on submitting batch jobs, see the [Batch Script Examples](#) page.

8.5. Interactive Batch Jobs

• [\(Back to Top\)](#)

Batch scripts are useful for submitting a group of commands, allowing them to run through the queue, then viewing the results at a later time. However, it is sometimes necessary to run tasks within a job interactively. Users are not permitted to access compute nodes nor run `aprun` directly from login nodes. Instead, users must use an interactive batch job to allocate and gain access to compute resources interactively. This is done by using the `-I` option to `qsub`.

Interactive Batch Example

For interactive batch jobs, PBS options are passed through `qsub` on the command line.

```
$ qsub -I -A pj000 -q debug -X -l nodes=3,walltime=30:00
```

This request will:

<code>-I</code>	Start an interactive session
<code>-A</code>	Charge to the "pj000" project
<code>-X</code>	Enables X11 forwarding. The DISPLAY environment variable must be set.
<code>-q debug</code>	Run in the debug queue
<code>-l nodes=3,walltime=30:00</code>	Request 3 compute nodes for 30 minutes (you get all cores per node)

After running this command, you will have to wait until enough compute nodes are available, just as in any other batch job. However, once the job starts, you will be given an interactive prompt on the head node of your allocated resource. From here commands may be executed directly instead of through a batch script.

Debugging via Interactive Jobs

A common use of interactive batch is to aid in debugging efforts. Interactive access to compute resources allows the ability to run a process to the point of failure; however, unlike a batch job, the process can be restarted after brief changes are made without losing the compute resource allocation. This may help speed the debugging effort because a user does not have to wait in the queue in between each run attempts.

Note: To tunnel a GUI from an interactive batch job, the `-X` PBS option should be used to enable X11 forwarding.

Choosing an Interactive Job's nodes Value

Because interactive jobs must sit in the queue until enough resources become available to allocate, to shorten the queue wait time, it is useful to base nodes selection on the number of unallocated nodes. The showbf command (i.e "show backfill") to see resource limits that would allow your job to be immediately back-filled (and thus started) by the scheduler. For example, the snapshot below shows that 802 nodes are currently free.

\$ showbf					
Partition	Tasks	Nodes	StartOffset	Duration	StartDate
-----	-----	-----	-----	-----	-----
ALL	4744	802	INFINITY	00:00:00	HH:MM:SS_MM/DD

See showbf -help for additional options.

8.6. Common Batch Options to PBS

• [\(Back to Top\)](#)

The following table summarizes frequently-used options to PBS:

Option	Use	Description
-A	#PBS -A <account>	Causes the job time to be charged to <account>. The account string, e.g. pjt000, is typically composed of three letters followed by three digits and optionally followed by a subproject identifier. The utility showproj can be used to list your valid assigned project ID(s). This option is required by all jobs.
-l	#PBS -l nodes=<value>	Maximum number of compute nodes. Jobs cannot request partial nodes.
	#PBS -l walltime=<time>	Maximum wall-clock time. <time> is in the format HH:MM:SS.
-o	#PBS -o <filename>	Writes standard output to <name> instead of <job script>.o\$PBS_JOBID. \$PBS_JOBID is an environment variable created by PBS that contains the PBS job identifier.
-e	#PBS -e <filename>	Writes standard error to <name> instead of <job script>.e\$PBS_JOBID.
-j	#PBS -j {oe, eo}	Combines standard output and standard error into the standard error file (eo) or the standard out file (oe).
-m	#PBS -m a	Sends email to the submitter when the job aborts.
	#PBS -m b	Sends email to the submitter when the job begins.
	#PBS -m e	Sends email to the submitter when the job ends.
-M	#PBS -M <address>	Specifies email address to use for -m options.
-N	#PBS -N <name>	Sets the job name to <name> instead of the name of the job script.
-S	#PBS -S <shell>	Sets the shell to interpret the job script.
-q	#PBS -q <queue>	Directs the job to the specified queue.This option is not required to run in the default queue on any given system.
-V	#PBS -V	Exports all environment variables from the submitting shell into the batch job shell. Not Recommended Because the login nodes differ from the service nodes, using the '-V' option is not recommended. Users should create the needed environment within the batch job.
-X	#PBS -X	Enables X11 forwarding. The -X PBS option should be used to tunnel a GUI from an interactive batch job.

Note: Because the login nodes differ from the service nodes, using the '-V' option is not recommended.

Users should create the needed environment within the batch job.

Further details and other PBS options may be found through the `qsub` man page.

8.7. Batch Environment Variables

• [\(Back to Top\)](#)

PBS sets multiple environment variables at submission time. The following PBS variables are useful within batch scripts:

Variable	Description
<code>\$PBS_O_WORKDIR</code>	The directory from which the batch job was <i>submitted</i> . By default, a new job starts in your home directory. You can get back to the directory of job submission with <code>cd \$PBS_O_WORKDIR</code> . Note that this is not necessarily the same directory in which the batch script resides.
<code>\$PBS_JOBID</code>	The job's full identifier. A common use for <code>PBS_JOBID</code> is to append the job's ID to the standard output and error files.
<code>\$PBS_NUM_NODES</code>	The number of nodes requested.
<code>\$PBS_JOBNAME</code>	The job name supplied by the user.
<code>\$PBS_NODEFILE</code>	The name of the file containing the list of nodes assigned to the job. Used sometimes on non-Cray clusters.

8.8. Modifying Batch Jobs

• [\(Back to Top\)](#)

The batch scheduler provides a number of utility commands for managing submitted jobs. See each utilities' man page for more information.

Removing and Holding Jobs

`qdel`

Jobs in the queue in any state can be stopped and removed from the queue using the command `qdel`.

```
$ qdel 1234
```

`qhold`

Jobs in the queue in a non-running state may be placed on hold using the `qhold` command. Jobs placed on hold will not be removed from the queue, but they will not be eligible for execution.

```
$ qhold 1234
```

`qrls`

Once on hold the job will not be eligible to run until it is released to return to a queued state. The `qrls` command can be used to remove a job from the held state.

```
$ qrls 1234
```

Modifying Job Attributes

`qalter`

Non-running jobs in the queue can be modified with the PBS `qalter` command. The `qalter` utility can be used to do the following (among others): Modify the job's name:

```
$ qalter -N newname 130494
```


Modify the number of requested cores:

```
$ qalter -l nodes=12 130494
```

Modify the job's walltime:

```
$ qalter -l walltime=01:00:00 130494
```

Note: Once a batch job moves into a running state, the job's walltime can not be increased.

8.9. Monitoring Batch Jobs

• [\(Back to Top\)](#)

PBS and Moab provide multiple tools to view queue, system, and job status. Below are the most common and useful of these tools.

Job Monitoring Commands

showq

The Moab utility `showq` can be used to view a more detailed description of the queue. The utility will display the queue in the following states:

- Active** These jobs are currently running.
- Eligible** These jobs are currently queued awaiting resources. Eligible jobs are shown in the order in which the scheduler will consider them for allocation.
- Blocked** These jobs are currently queued but are not eligible to run. A job may be in this state because the user has more jobs that are "eligible to run" than the system's queue policy allows.

To see all jobs currently in the queue:

```
$ showq
```

To see all jobs owned by userA currently in the queue:

```
$ showq -u userA
```

Note: To increase response time, the MOAB utilities (*showstart*, *checkjob*) will display a cached result. The cache updates every 30 seconds. But, because the cached result is displayed, you may see the following message:

```
-----  
NOTE: The following information has been cached by the remote server  
and may be slightly out of date.  
-----
```

checkjob

The Moab utility `checkjob` can be used to view details of a job in the queue. For example, if job 736 is a job currently in the queue in a blocked state, the following can be used to view why the job is in a blocked state:

```
$ checkjob 736
```

The return may contain a line similar to the following:

```
BlockMsg: job 736 violates idle HARD MAXJOB limit of X for user (Req: 1 InUse: X)
```

This line indicates the job is in the blocked state because the owning user has reached the limit for jobs in the "eligible to run" state.

qstat

The PBS utility `qstat` will poll PBS (Torque) for job information. However, `qstat` does not know of Moab's blocked and eligible states. Because of this, the `showq` Moab utility (see above) will provide a more accurate batch queue state. To show show all queued jobs:

```
$ qstat -a
```

To show details about job 1234:

```
$ qstat -f 1234
```

To show all currently queued jobs owned by userA:

```
$ qstat -u userA
```

8.10. Titan Batch Queues

• [\(Back to Top\)](#)

Queues are used by the batch scheduler to aid in the organization of jobs. Users typically have access to multiple queues, and each queue may allow different job limits and have different priorities. Unless otherwise notified, users have access to the following queues on Titan:

Name	Usage	Description	Limits
<code>batch</code>	No explicit request required	Default; most production work runs in this queue.	See the Titan Scheduling Policy for details.
<code>killable</code>	<code>#PBS -q killable</code>	Opportunistic; jobs start even if they will not complete before the onset of a scheduled outage.	
<code>debug</code>	<code>#PBS -q debug</code>	Quick-turnaround; short jobs for software development, testing, and debugging.	

The `batch` Queue

The `batch` queue is the default queue for production work on Titan. Most work on Titan is handled through this queue.

The `killable` Queue

At the start of a scheduled system outage, a *queue reservation* is used to ensure that no jobs are running. In the `batch` queue, the scheduler will not start a job if it expects that the job would not complete (based on the job's user-specified max walltime) before the reservation's start time. In contrast, the `killable` queue allows the scheduler to start a job even if it will *not* complete before a scheduled reservation.

Note: If your job can perform usable work in a (1) hour timeframe and is tolerant of abrupt termination, this queue may allow you to take advantage of idle resources available prior to a scheduled outage.

The `debug` Queue

The `debug` queue is intended to provide faster turnaround times for the code development, testing, and debugging cycle. For example, interactive parallel work is an ideal use for the `debug` queue.

Warning: Users who misuse the `debug` queue may have further access to the queue denied.

More detailed information on any of the batch scheduler queues can be found on the [Titan Scheduling Policy](#) page.

8.11. Job Execution on Titan

• [\(Back to Top\)](#)

Once resources have been allocated through the batch system, users can:

- Run commands in serial on the resource pool's primary service node
- Run executables in parallel across compute nodes in the resource pool

Serial Execution

The executable portion of a batch script is interpreted by the shell specified on the first line of the script. If a shell is not specified, the submitting user's default shell will be used. This portion of the script may contain comments, shell commands, executable scripts, and compiled executables. These can be used in combination to, for example, navigate file systems, set up job execution, run executables, and even submit other batch jobs.

Parallel Execution

By default, commands in the job submission script will be executed on the job's primary service node. The `aprun` command is used to execute a binary on one or more compute nodes within a job's allocated resource pool.

Note: On Titan, the **only** way access a compute node is via the `aprun` command within a batch job.

8.11.1. Using the `aprun` command

• [\(Back to Top\)](#)

The `aprun` command is used to run a compiled application program across one or more compute nodes. You use the `aprun` command to specify application resource requirements, request application placement, and initiate application launch. The machine's physical [node layout](#) plays an important role in how `aprun` works. Each Titan compute node contains (2) 8-core NUMA nodes on a single socket (a total of 16 cores).

Note: The `aprun` command is the only mechanism for running an executable in parallel on compute nodes. To run jobs as efficiently as possible, a thorough understanding of how to use `aprun` and its various options is paramount.

OLCF uses a version of `aprun` with two extensions. One is used to identify which libraries are used by an executable to allow us to better track third party software that is being actively used on the system. The other analyzes the command line to identify cases where users might be able to optimize their application's performance by using slightly different job layout options. We highly recommend using both of these features; however, if there is a reason you wish to disable one or the other please contact the User Assistance Center for information on how to do that.

Shell Resource Limits

By default, `aprun` will *not* forward shell limits set by `ulimit` for sh/ksh/bash or by `limit` for csh/tcsh. To pass these settings to your batch job, you should set the environment variable `APRUN_XFER_LIMITS` to 1 via `export APRUN_XFER_LIMITS=1` for sh/ksh/bash or `setenv APRUN_XFER_LIMITS 1` for csh/tcsh.

Simultaneous `aprun` Limit

All `aprun` processes are launched from a small number of shared service nodes. Because large numbers of `aprun` processes can cause other users' `apruns` to fail, users are asked to limit the number of simultaneous `apruns` executed within a batch script. Users are limited to 100 `aprun` processes per batch job; attempts to launch `apruns` over the limit will result in the following error:

```
apsched: no more claims allowed for this reservation (max 100)
```

Warning: Users are limited to 100 `aprun` processes per batch job.

Common `aprun` Options

The following table lists commonly-used options to `aprun`. For a more detailed description of `aprun` options, see the `aprun`

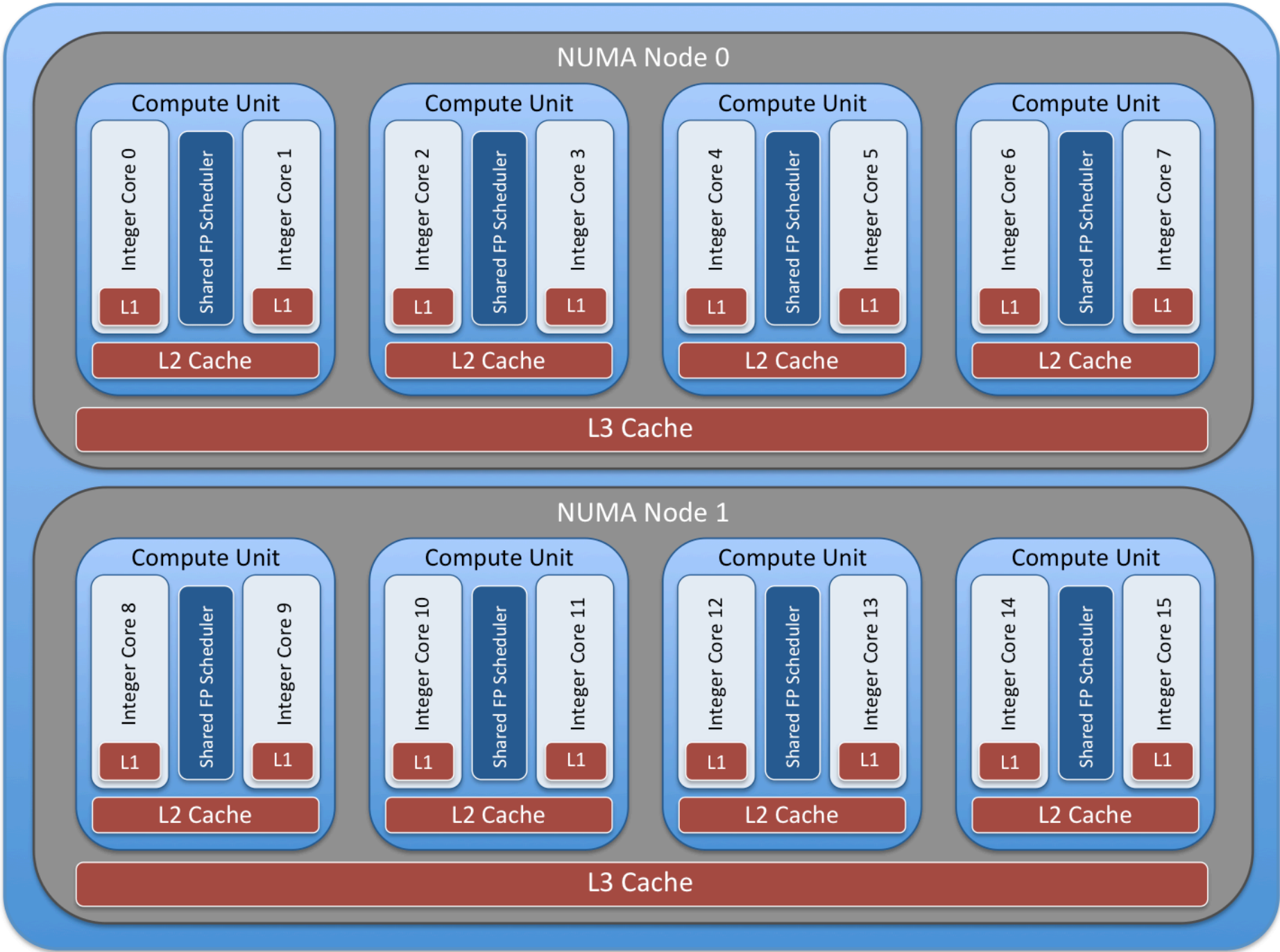
man page.

Option	Description
-D	Debug; shows the layout aprun will use
-n	Number of total MPI tasks (aka 'processing elements') for the executable. If you do not specify the number of tasks to aprun, the system will default to 1.
-N	Number of MPI tasks (aka 'processing elements') per physical node. <div>Warning: Because each node contains multiple processors/NUMA nodes, the -S option is likely a better option than -N to control layout within a node.</div>
-m	Memory required per MPI task. There is a maximum of 2GB per core, i.e. requesting 2.1GB will allocate two cores minimum per MPI task
-d	Number of threads per MPI task. <div>Warning: The default value for -d is 1. If you specify OMP_NUM_THREADS but do not give a -d option, aprun will allocate your threads to a single core. Use OMP_NUM_THREADS to specify to your code the number of threads per MPI task; use -d to tell aprun how to place those threads.</div>
-j	<div>For Titan: Number of CPUs to use per <i>paired-core compute unit</i>. The -j parameter specifies the number of CPUs to be allocated per <i>paired-core compute unit</i>. The valid values for -j are 0 (use the system default), 1 (use one integer core), and 2 (use both integer cores; this is the system default).</div> <div>For Eos: The -j parameter controls Hyper Threading. The valid values for -j are 0 (use the system default), 1 (turn Hyper Threading off; this is the system default), and 2 (turn Hyper Threading on).</div>
-cc	This is the cpu_list option. It binds MPI tasks or threads to the specified CPUs. The list is given as a set of comma-separated numbers (0 though 15) which each specify a compute unit (core) on the node. The list can also be given as hyphen-separated rages of numbers which each specify a range of compute units (cores) on the node. See man aprun.
-S	Number of MPI tasks (aka 'processing elements') per NUMA node. Can be 1, 2, 3, 4, 5, 6, 7, or 8.
-ss	Strict memory containment per NUMA node. The default is to allow remote NUMA node memory access. This option prevents memory access of the remote NUMA node.
-r	Assign system services associated with your application to a compute core. If you use less than 16 cores, you can request all of the system services to be placed on an unused core. This will reduce "jitter" (i.e. application variability) because the daemons will not cause the application to context switch unexpectedly. Should use -r 1 ensuring -N is less than 16 or -S is less than 8.

8.11.2. XK7 CPU Description

• (Back to Top)

Each Titan compute node contains (1) AMD Opteron™ 6274 (Interlagos) CPU. Each CPU contains (2) die. Each die contains (4) "bulldozer" compute units and a shared L3 cache. Each compute unit contains (2) integer cores (and their L1 cache), a shared floating point scheduler, and shared L2 cache. To aid in task placement, each die is organized into a NUMA node. Each compute node contains (2) NUMA nodes. Each NUMA node contains a die's L3 cache and its (4) compute units (8 cores). This configuration is shown graphically below.



8.11.3. Controlling MPI Task Layout Within a Physical Node

• [\(Back to Top\)](#)

Users have (2) ways to control MPI task layout:

- 1. Within a physical node
- 2. Across physical nodes

This article focuses on how to control MPI task layout within a physical node.

Understanding NUMA Nodes

Each physical node is organized into (2) 8-core *NUMA nodes*. *NUMA* is an acronym for "Non-Uniform Memory Access". You can think of a NUMA node as a division of a physical node that contains a subset of processor cores and their high-affinity memory. Applications may use resources from one or both NUMA nodes. The default MPI task layout is *SMP-style*. This means MPI will sequentially allocate *all* cores on one NUMA node before allocating tasks to another NUMA node.

Note: A brief description of how a physical XK7 node is organized can be found on the [XK7 node description page](#).

Spreading MPI Tasks Across NUMA Nodes

Each physical node contains (2) NUMA nodes. Users can control MPI task layout using the `aprun` NUMA node flags. For jobs that do not utilize all cores on a node, it may be beneficial to spread a physical node's MPI task load over the (2) available NUMA nodes via the `-S` option to `aprun`.

Note: Jobs that do not utilize all of a physical node's processor cores may see performance improvements by spreading MPI tasks across NUMA nodes within a physical node.

Example 1: Default NUMA Placement

Job requests (2) processor cores without a NUMA flag. Both tasks are placed on the first NUMA node.


```
$ aprun -n2 ./a.out
Rank 0, Node 0, NUMA 0, Core 0
Rank 1, Node 0, NUMA 0, Core 1
```

Example 2: Specific NUMA Placement

Job requests (2) processor cores with `aprun -S`. A task is placed on each of the (2) NUMA nodes:

```
$ aprun -n2 -S1 ./a.out
Rank 0, Node 0, NUMA 0, Core 0
Rank 1, Node 0, NUMA 1, Core 0
```

The following table summarizes common NUMA node options to `aprun`:

Option	Description
-S	Processing elements (essentially a processor core) per NUMA node. Specifies the number of PEs to allocate per NUMA node. Can be 1, 2, 3, 4, 5, 6, 7, or 8.
-ss	Strict memory containment per NUMA node. The default is to allow remote NUMA node memory access. This option prevents memory access of the remote NUMA node.

Advanced NUMA Node Placement

Example 1: Grouping MPI Tasks on a Single NUMA Node

Run a.out on (8) cores. Place (8) MPI tasks on (1) NUMA node. In this case the `aprun -S` option is optional:

```
$ aprun -n8 -S8 ./a.out
```

Compute Node															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0	1	2	3	4	5	6	7								

Example 2: Spreading MPI tasks across NUMA nodes

Run a.out on (8) cores. Place (4) MPI tasks on each of (2) NUMA nodes via `aprun -S`.

```
$ aprun -n8 -S4 ./a.out
```

Compute Node															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0	1	2	3					4	5	6	7				

Example 3: Spread Out MPI Tasks Across Paired-Core Compute Units

The `-j` option can be used for codes to allow one task per *paired-core compute unit*. Run a.out on (8) cores; (4) cores per NUMA node; but only (1) core on each paired-core compute unit:

```
$ aprun -n8 -S4 -j1 ./a.out
```

Compute Node							
NUMA 0				NUMA 1			

Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0		1		2		3		4		5		6		7	

To see MPI rank placement information on the nodes set the PMI_DEBUG environment variable to 1 For cshell:

```
$ setenv PMI_DEBUG 1
```

For bash:

```
$ export PMI_DEBUG=1
```

Example 4: Assign System Services to a Unused Compute Core

The `-r` option can be used to assign system services associated with your application to a compute core. If you use less than 16 cores, you can request all of the system services to be placed on an unused core. This will reduce "jitter" (i.e. application variability) because the daemons will not cause the application to context switch unexpectedly. You should use `-r 1` ensuring `-N` is less than 16 or `-S` is less than 8. The following example will place a task from a.out on cores 0-14; core 15 will be used only for system services: Run a.out on (8) cores; (4) cores per NUMA node; but only (1) core on each paired-core compute unit. All node services will be placed on the node's last core:

```
$ aprun -n8 -S4 -j1 -r1 ./a.out
```

Compute Node															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0		1		2		3		4		5		6		7	*

* *System Services*

8.11.4. Controlling MPI Task Layout Across Many Physical Nodes

• [\(Back to Top\)](#)

Users have (2) ways to control MPI task layout:

1. Within a physical node
2. Across physical nodes

This article focuses on how to control MPI task layout across physical nodes nodes. The default MPI task layout is SMP-style. This means MPI will sequentially allocate all cores on one physical node before allocating tasks to another physical node.

Viewing Multi-Node Layout Order

Task layout can be seen by setting `MPICH_RANK_REORDER_DISPLAY` to 1.

Changing Multi-Node Layout Order

For multi-node jobs, layout order can be changed using the environment variable `MPICH_RANK_REORDER_METHOD`. See `man intro_mpi` for more information.

Multi-Node Layout Order Examples

Example 1: Default Layout

The following will run a.out across (32) cores. This requires (2) physical compute nodes.

```
$ aprun -n 32 ./a.out
```

--

Compute Node 0															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Compute Node 1															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Example 2: Round-Robin Layout

The following will place tasks in a round robin fashion. This requires (2) physical compute nodes.

```
$ setenv MPICH_RANK_REORDER_METHOD 0
$ aprun -n 32 ./a.out
```

Compute Node 0															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30

Compute Node 1															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31

Example 3: Combining Inter-Node and Intra-Node Options

The following combines `MPICH_RANK_REORDER_METHOD` and `-S` to place tasks on three cores per processor within a node and in a round robin fashion across nodes.

```
$ setenv MPICH_RANK_REORDER_METHOD 0
$ aprun -n12 -S3 ./a.out
```

Compute Node 0															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
0	2	4						6	8	10					

Compute Node 1															
NUMA 0								NUMA 1							
Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7	Core 0	Core 1	Core 2	Core 3	Core 4	Core 5	Core 6	Core 7
1	3	5						7	9	11					

• [\(Back to Top\)](#)

Titan supports threaded programming within a compute node. Threads may span across both processors within a single compute node, but cannot span compute nodes. Users have a great deal of flexibility in thread placement. Several examples are shown below.

Note: Threaded codes must use the `-d` (depth) option to `aprun`.

The `-d` option to `aprun` specifies the number of threads per MPI task. Under previous CNL versions this option was not required. Under the current CNL version, the number of cores used is calculated by multiplying the value of `-d` by the value of `-n`.

Warning: Without the `-d` option, all threads will be started on the same processor core. This can lead to performance degradation for threaded codes.

Thread Layout Examples

The following examples are written for the bash shell. If using `csh/tcsh`, you should change `export OMP_NUM_THREADS=x` to `setenv OMP_NUM_THREADS x` wherever it appears.

Example 1: (2) MPI tasks, (16) Threads Each

This example will launch (2) MPI tasks, each with (16) threads. This requests (2) compute nodes and requires a `node` request of (2):

```
$ export OMP_NUM_THREADS=16
$ aprun -n2 -d16 a.out

Rank 0, Thread 0, Node 0, NUMA 0, Core 0 <-- MASTER
Rank 0, Thread 1, Node 0, NUMA 0, Core 1 <-- slave
Rank 0, Thread 2, Node 0, NUMA 0, Core 2 <-- slave
Rank 0, Thread 3, Node 0, NUMA 0, Core 3 <-- slave
```

Example 2: (2) MPI tasks, (6) Threads Each

This example will launch (2) MPI tasks, each with (6) threads. Place (1) MPI task per [NUMA node](#). This requests (1) physical compute nodes and requires a `nodes` request of (1):

```
$ export OMP_NUM_THREADS=6
$ aprun -n2 -d6 -S1 a.out
```

Compute Node															
NUMA 0								NUMA 1							
Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7	Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7
Rank0	Rank0	Rank0	Rank0	Rank0	Rank0			Rank1	Rank1	Rank1	Rank1	Rank1	Rank1		
Thread0	Thread1	Thread2	Thread3	Thread4	Thread5			Thread0	Thread1	Thread2	Thread3	Thread4	Thread5		

Example 3: (4) MPI tasks, (2) Threads Each

This example will launch (4) MPI tasks, each with (2) threads. Place only (1) MPI task [and its (2) threads] on each [NUMA node](#). This requests (2) physical compute nodes and requires a `nodes` request of (2), even though only (8) cores are actually being used:

```
$ export OMP_NUM_THREADS=2
$ aprun -n4 -d2 -S1 a.out

Rank 0, Thread 0, Node 0, NUMA 0, Core 0 <-- MASTER
Rank 0, Thread 1, Node 0, NUMA 0, Core 1 <-- slave
Rank 1, Thread 0, Node 0, NUMA 1, Core 0 <-- MASTER
Rank 1, Thread 1, Node 0, NUMA 1, Core 1 <-- slave
```

Example 4: (2) MPI tasks, (4) Threads Each, Using only (1) core per compute unit

The `-j` option can be used to allow use of only one core per *paired-core compute unit*. This example will launch (2) MPI

tasks, each with (4) threads. Place only (1) MPI task [and its (4) threads] on each [NUMA node](#). One core per *paired-core compute unit* sit idle. This requires (1) physical compute node and requires a nodes request of (1), even though only (8) cores are actually being used:

```
$ export OMP_NUM_THREADS=4
$ aprun -n2 -d4 -S1 -j1 a.out
```

Compute Node															
NUMA 0								NUMA 1							
Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7	Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7
Rank0 Thread0		Rank0 Thread1		Rank0 Thread2		Rank0 Thread3		Rank1 Thread0		Rank1 Thread1		Rank1 Thread2		Rank1 Thread3	

Example 5: (2) MPI tasks, (8) Threads Each, Using only (1) core per compute unit

The `-j` option can be used to allow use of only one core per *paired-core compute unit*. This example will launch (1) MPI tasks, each with (8) threads. One core per *paired-core compute unit* will sit idle. This requires (2) physical compute node and requires a size request of (32), even though only (16) cores are actually being used:

```
$ export OMP_NUM_THREADS=8
$ aprun -n2 -d8 -N1 -j1 a.out
```

Compute Node 0															
NUMA 0								NUMA 1							
Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7	Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7
Rank0 Thread0		Rank0 Thread1		Rank0 Thread2		Rank0 Thread3		Rank0 Thread4		Rank0 Thread5		Rank0 Thread6		Rank0 Thread7	
Compute Node 1															
NUMA 0								NUMA 1							
Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7	Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7
Rank1 Thread0		Rank1 Thread1		Rank1 Thread2		Rank1 Thread3		Rank1 Thread4		Rank1 Thread5		Rank1 Thread6		Rank1 Thread7	

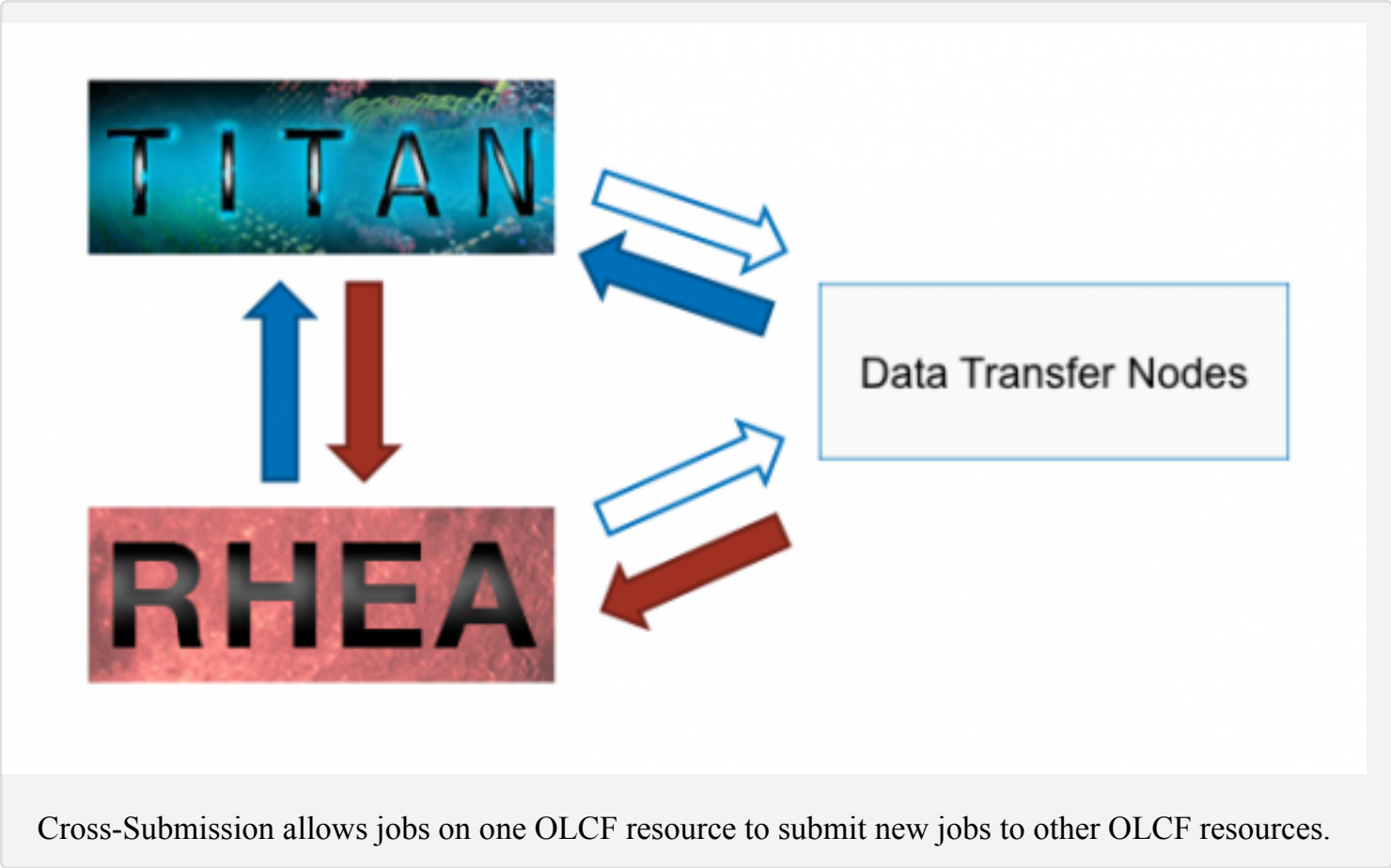
The `-cc` option can be used to control the placement of threads or tasks on specific processing units. To accomplish the same layout shown above with `-cc`

```
$ export OMP_NUM_THREADS=8
$ aprun -n2 -d8 -N1 -cc 0,2,4,6,8,10,12,14 ./a.out
```

8.12. Enabling Workflows through Cross-System Batch Submission

• [\(Back to Top\)](#)

The OLCF now supports submitting jobs between OLCF systems via batch scripts. This can be useful for automatically triggering analysis and storage of large data sets after a successful simulation job has ended, or for launching a simulation job automatically once the input deck has been retrieved from HPSS and pre-processed.

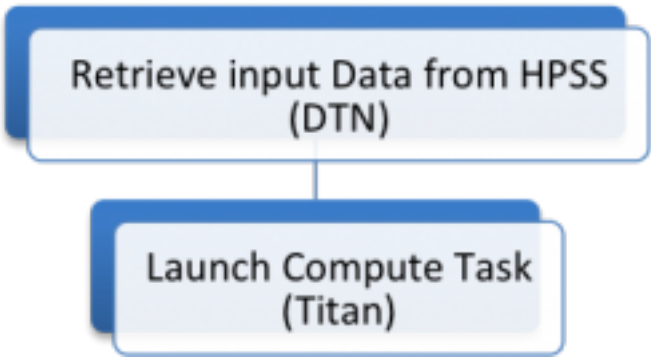


The key to remote job submission is the command `qsub -q host script.pbs` which will submit the file `script.pbs` to the batch queue on the specified host. This command can be inserted at the end of an existing batch script in order to automatically trigger work on another OLCF resource. This feature is supported on the following hosts:

Host	Remote Submission Command
Rhea	<code>qsub -q rhea visualization.pbs</code>
Titan	<code>qsub -q titan compute.pbs</code>
Data Transfer Nodes (DTNs)	<code>qsub -q dtn retrieve_data.pbs</code>

Example Workflow 1: Automatic Post-Processing

The simplest example of a remote submission workflow would be automatically triggering an analysis task on Rhea at the completion of a compute job on Titan. This workflow would require two batch scripts, one to be submitted on Titan, and a second to be submitted automatically to Rhea. Visually, this workflow may look something like the following:



The batch scripts for such a workflow could be implemented as follows: **Batch-script-1.pbs**

```
#PBS -l walltime=0:30:00
#PBS -l nodes=1
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Retrieve data from HPSS
cd $MEMBERWORK/prj123
htar -xf /proj/prj123/compute_data.htar compute_data/

# Submit compute job to Titan
qsub -q titan Batch-script-2.pbs
```

Batch-script-2.pbs

```
#PBS -l walltime=2:00:00
#PBS -l nodes=4096
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Launch executable
cd $MEMBERWORK/prj123
aprun -n 65536 ./analysis-task.exe
```

The key to this workflow is the `qsub -q batch@rhea-batch Batch-script-2.pbs` command, which tells `qsub` to submit the file `Batch-script-2.pbs` to the batch queue on Rhea.

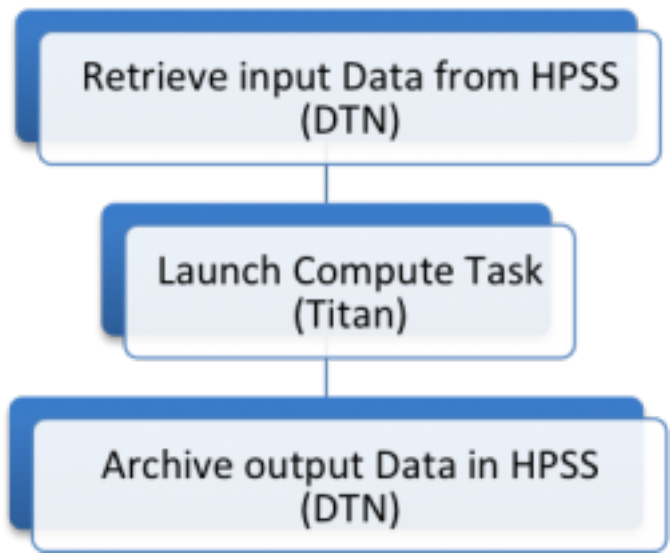
Initializing the Workflow

We can initialize this workflow in one of two ways:

- Log into `dtm.ccs.ornl.gov` and run `qsub Batch-script-1.pbs` OR
- From Titan or Rhea, run `qsub -q dtm Batch-script-1.pbs`

Example Workflow 2: Data Staging, Compute, and Archival

Now we give another example of a linear workflow. This example shows how to use the Data Transfer Nodes (DTNs) to retrieve data from HPSS and stage it to your project's scratch area before beginning. Once the computation is done, we will automatically archive the output.



Batch-script-1.pbs

```
#PBS -l walltime=0:30:00
#PBS -l nodes=1
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Retrieve Data from HPSS
cd $MEMBERWORK/prj123
htar -xf /proj/prj123/input_data.htar input_data/

# Launch compute job
qsub -q titan Batch-script-2.pbs
```

Batch-script-2.pbs

```
#PBS -l walltime=6:00:00
#PBS -l nodes=4096
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Launch exectuable
cd $MEMBERWORK/prj123
aprun -n 65536 ./analysis-task.exe

# Submit data archival job to DTNs
qsub -q dtm Batch-script-3.pbs
```

Batch-script-3.pbs

```
#PBS -l walltime=0:30:00
#PBS -l nodes=1
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Launch exectuable
cd $MEMBERWORK/prj123
htar -cf /proj/prj123/viz_output.htar viz_output/
htar -cf /proj/prj123/compute_data.htar compute_data/
```

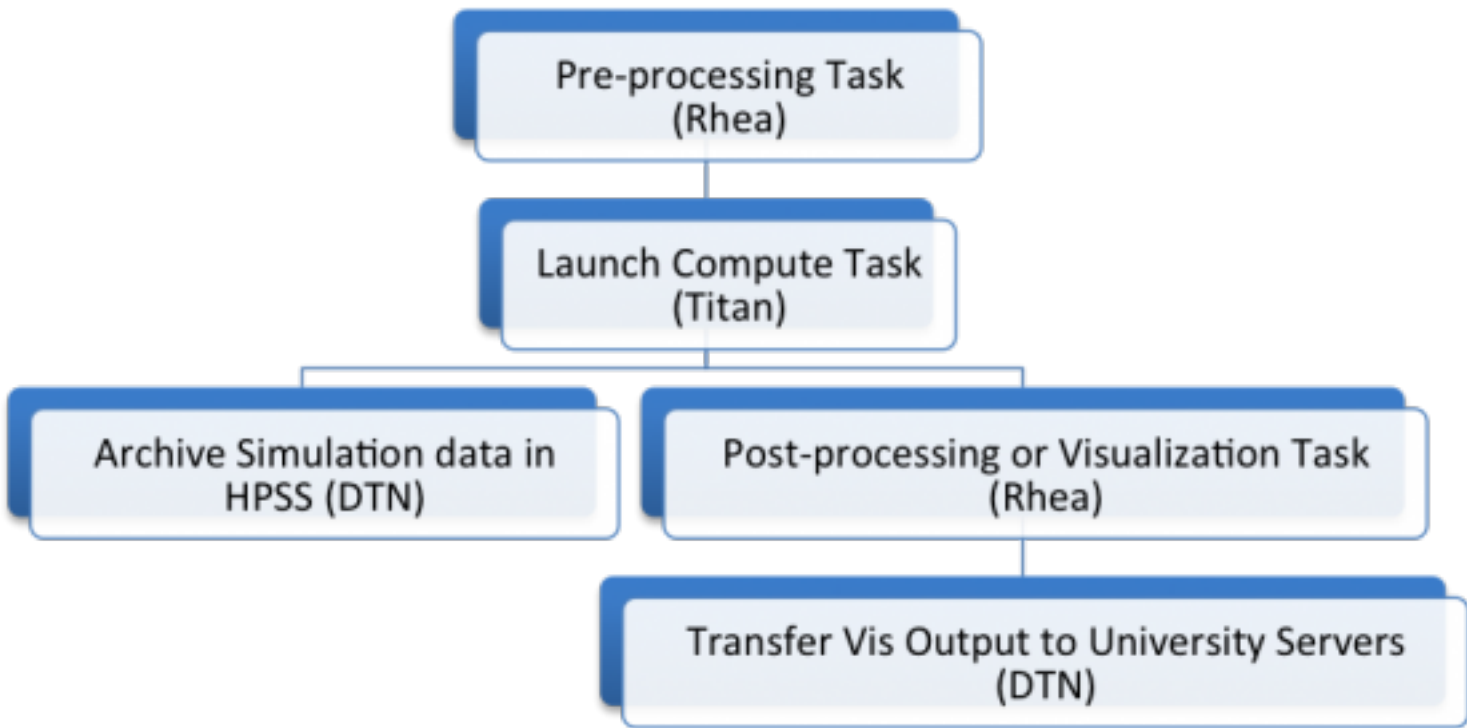
Initializing the Workflow

We can initialize this workflow in one of two ways:

- Log into dtn.ccs.ornl.gov and run qsub Batch-script-1.pbs OR
- From Titan or Rhea, run qsub -q dtn Batch-script-1.pbs

Example Workflow 3: Data Staging, Compute, Visualization, and Archival

This is an example of a "branching" workflow. What we will do is first use Rhea to prepare a mesh for our simulation on Titan. We will then launch the compute task on Titan, and once this has completed, our workflow will branch into two separate paths: one to archive the simulation output data, and one to visualize it. After the visualizations have finished, we will transfer them to a remote institution.



Step-1.prepare-data.pbs

```
#PBS -l walltime=0:30:00
#PBS -l nodes=10
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Prepare Mesh for Simulation
mpirun -n 160 ./prepare-mesh.exe

# Launch compute job
qsub -q titan Step-2.compute.pbs
```

Step-2.compute.pbs

```
#PBS -l walltime=6:00:00
#PBS -l nodes=4096
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Launch exectuable
cd $MEMBERWORK/prj123
aprun -n 65536 ./analysis-task.exe

# Workflow branches at this stage, launching 2 separate jobs

# - Launch Archival task on DTNs
qsub -q dtn@dtn-batch Step-3.archive-compute-data.pbs

# - Launch Visualization task on Rhea
qsub -q rhea Step-4.visualize-compute-data.pbs
```

Step-3.archive-compute-data.pbs

```
#PBS -l walltime=0:30:00
#PBS -l nodes=1
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Archive compute data in HPSS
cd $MEMBERWORK/prj123
htar -cf /proj/prj123/compute_data.htar compute_data/
```

Step-4.visualize-compute-data.pbs

```
#PBS -l walltime=2:00:00
#PBS -l nodes=64
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Visualize Compute data
cd $MEMBERWORK/prj123
mpirun -n 768 ./visualization-task.py

# Launch transfer task
qsub -q dtn Step-5.transfer-visualizations-to-campus.pbs
```

Step-5.transfer-visualizations-to-campus.pbs

```
#PBS -l walltime=2:00:00
#PBS -l nodes=1
#PBS -A PRJ123
#PBS -l gres=atlas1%atlas2

# Transfer visualizations to storage area at home institution
cd $MEMBERWORK/prj123
SOURCE=gsiftp://dtn03.ccs.ornl.gov/$MEMBERWORK/visualization.mpg
DEST=gsiftp://dtn.university-name.edu/userid/visualization.mpg
globus-url-copy -tcp-bs 12M -bs 12M -p 4 $SOURCE $DEST
```

Initializing the Workflow

We can initialize this workflow in one of two ways:

- Log into `rhea.ccs.ornl.gov` and run `qsub Step-1.prepare-data.pbs` OR
- From Titan or the DTNs, run `qsub -q rhea Step-1.prepare-data.pbs`

Checking Job Status

Host	Remote qstat	Remote showq
Rhea	<code>qstat -a @rhea-batch</code>	<code>showq --host=rhea-batch</code>
Titan	<code>qstat -a @titan-batch</code>	<code>showq --host=titan-batch</code>
Data Transfer Nodes (DTNs)	<code>qstat -a @dtn-batch</code>	<code>showq --host=dtn-batch</code>

Deleting Remote Jobs

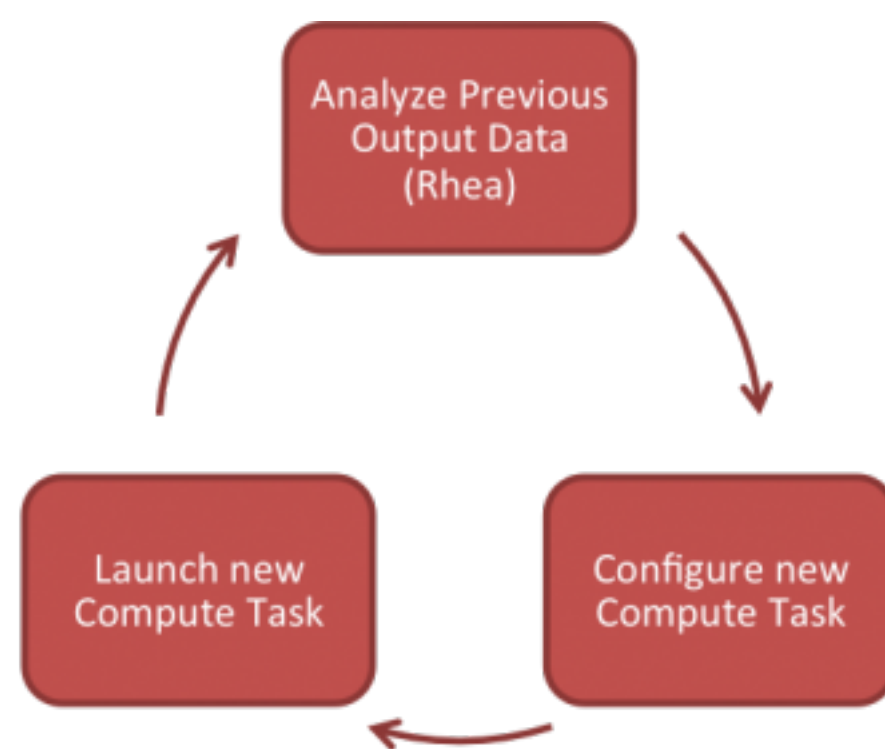
In order to delete a job (say, job number 18688) from a remote queue, you can do the following

Host	Remote qdel
Rhea	<code>qdel 18688@rhea-batch</code>
Titan	<code>qdel 18688@titan-batch</code>
Data Transfer Nodes (DTNs)	<code>qdel 18688@dtn-batch</code>

Potential Pitfalls

The OLCF advises users to keep their remote submission workflows simple, short, and mostly linear. Workflows that contain many layers of branches, or that trigger many jobs at once, may prove difficult to maintain and debug. Workflows that contain loops or recursion (jobs that can submit themselves again) may inadvertently waste allocation hours if a suitable exit condition is not reached.

Recursive workflows which do not exit will drain your project's allocation. Refunds will not be granted. Please be extremely cautious when designing workflows that cause jobs to re-submit themselves.



As always, users on multiple projects are strongly advised to double check that the `#PBS -A <PROJECTID>` field is set to the correct project prior to submission. This will ensure that resource usage is associated with the intended project.

8.13. Job Resource Accounting

• [\(Back to Top\)](#)

The hybrid nature of Titan's accelerated XK7 nodes mandated a new approach to its node allocation and job charge units. For the sake of resource accounting, each Titan XK7 node will be *defined* as possessing (30) total cores (e.g. (16) CPU cores + (14) GPU core equivalents). Jobs consume charge units in "Titan core-hours", and each Titan node consumes (30) of such units per hour. As in years past, jobs on the Titan system will be scheduled in full node increments; a node's cores cannot be allocated to multiple jobs. Because the OLCF charges based on what a job makes *unavailable* to other users, a job is charged for an entire node even if it uses only one core on a node. To simplify the process, users are required to request an entire node through PBS. Notably, codes that do not take advantage of GPUs will have only (16) CPU cores available per node; however, allocation requests—and units charged—will be based on (30) cores per node.

Note: Whole nodes must be requested at the time of job submission, and associated allocations are reduced by (30) core-hours per node, regardless of actual CPU or GPU core utilization.

Viewing Allocation Utilization

Projects are allocated time on Titan in units of "Titan core-hours". Other OLCF systems are allocated in units of "core-hours". This page describes how such units are calculated, and how users can access more detailed information on their relevant allocations.

Titan Core-Hour Calculation

The *Titan core-hour* charge for each batch job will be calculated as follows:

$$\text{Titan core-hours} = \text{nodes requested} * 30 * (\text{batch job endtime} - \text{batch job starttime})$$

Where *batch job starttime* is the time the job moves into a running state, and *batch job endtime* is the time the job exits a running state. A batch job's usage is calculated solely on requested nodes and the batch job's start and end time. The number of cores actually used within any particular node within the batch job is not used in the calculation. For example, if a job requests 64 nodes through the batch script, runs for an hour, uses only 2 CPU cores per node, and uses no GPU cores, the job will still be charged for $64 * 30 * 1 = 1,920$ *Titan core-hours*.

Viewing Usage

Utilization is calculated daily using batch jobs which complete between 00:00 and 23:59 of the previous day. For example, if a job moves into a run state on Tuesday and completes Wednesday, the job's utilization will be recorded Thursday. Only batch jobs which write an end record are used to calculate utilization. Batch jobs which do not write end records due to system failure or other reasons are not used when calculating utilization. Each user may view usage for projects on which

they are members from the command line tool `showusage` and the [My OLCF site](#).

On the Command Line via `showusage`

The `showusage` utility can be used to view your usage from January 01 through midnight of the previous day. For example:

```
$ showusage
Usage on titan:

Project      Allocation      Project Totals      <userid>
              Usage      Remaining      Usage
-----
<YourProj>    2000000      123456.78    1876543.22      1560.80
```

The `-h` option will list more usage details.

On the Web via My OLCF

More detailed metrics may be found on each project's usage section of the [My OLCF site](#). The following information is available for each project:

- YTD usage by system, subproject, and project member
- Monthly usage by system, subproject, and project member
- YTD usage by job size groupings for each system, subproject, and project member
- Weekly usage by job size groupings for each system, and subproject
- Batch system priorities by project and subproject
- Project members

The My OLCF site is provided to aid in the utilization and management of OLCF allocations. If you have any questions or have a request for additional data, please contact the OLCF User Assistance Center.

8.14. Titan Scheduling Policy

• [\(Back to Top\)](#)

Note: This details an official policy of the OLCF, and must be agreed to by the following persons as a condition of access to or use of OLCF computational resources:

- Principal Investigators (Non-Profit)
- Principal Investigators (Industry)
- All Users

Title: Titan Scheduling Policy **Version:** 13.02

In a simple batch queue system, jobs run in a first-in, first-out (FIFO) order. This often does not make effective use of the system. A large job may be next in line to run. If the system is using a strict FIFO queue, many processors sit idle while the large job waits to run. *Backfilling* would allow smaller, shorter jobs to use those otherwise idle resources, and with the proper algorithm, the start time of the large job would not be delayed. While this does make more effective use of the system, it indirectly encourages the submission of smaller jobs.

The DOE Leadership-Class Job Mandate

As a DOE Leadership Computing Facility, the OLCF has a mandate that a large portion of Titan's usage come from large, *leadership-class* (aka *capability*) jobs. To ensure the OLCF complies with DOE directives, we strongly encourage users to run jobs on Titan that are as large as their code will warrant. To that end, the OLCF implements queue policies that enable large jobs to run in a timely fashion.

Note: The OLCF implements queue policies that encourage the submission and timely execution of large, leadership-class jobs on Titan.

The basic priority-setting mechanism for jobs waiting in the queue is the time a job has been waiting relative to other jobs in the queue. However, several factors are applied by the batch system to modify the *apparent* time a job has been waiting. These factors include:

- The number of nodes requested by the job.
- The queue to which the job is submitted.
- The 8-week history of usage for the project associated with the job.

- The 8-week history of usage for the user associated with the job.

If your jobs require resources outside these queue policies, please complete the relevant request form on the [Special Requests](#) page. If you have any questions or comments on the queue policies below, please direct them to the User Assistance Center.

Job Priority by Processor Count

Jobs are *aged* according to the job's requested processor count (older age equals higher queue priority). Each job's requested processor count places it into a specific *bin*. Each bin has a different aging parameter, which all jobs in the bin receive.

Bin	Min Nodes	Max Nodes	Max Walltime (Hours)	Aging Boost (Days)
1	11,250	--	24.0	15
2	3,750	11,249	24.0	5
3	313	3,749	12.0	0
4	125	312	6.0	0
5	1	124	2.0	0

FairShare Scheduling Policy

FairShare, as its name suggests, tries to push each user and project towards their fair share of the system's utilization: in this case, 5% of the system's utilization per user and 10% of the system's utilization per project. To do this, the job scheduler adds (30) minutes priority aging per user and (1) hour of priority aging per project for every (1) percent the user or project is under its fair share value for the prior (8) weeks. Similarly, the job scheduler subtracts priority in the same way for users or projects that are over their fair share. For instance, a user who has personally used 0.0% of the system's utilization over the past (8) weeks who is on a project that has also used 0.0% of the system's utilization will get a (12.5) hour bonus (5 * 30 min for the user + 10 * 1 hour for the project). In contrast, a user who has personally used 0.0% of the system's utilization on a project that has used 12.5% of the system's utilization would get no bonus (5 * 30 min for the user - 2.5 * 1 hour for the project).

batch Queue Policy

The `batch` queue is the default queue for production work on Titan. Most work on Titan is handled through this queue. It enforces the following policies:

- Limit of (2) *eligible-to-run* jobs per user.
- Jobs in excess of the per user limit above will be placed into a *held* state, but will change to eligible-to-run at the appropriate time.
- Users may have only (2) jobs in bin 5 *running* at any time.

Note: The *eligible-to-run* state is not the *running* state. Eligible-to-run jobs have not started and are waiting for resources. Running jobs are actually executing.

killable Queue Policy

At the start of a scheduled system outage, a *queue reservation* is used to ensure that no jobs are running. In the `batch` queue, the scheduler will not start a job if it expects that the job would not complete (based on the job's user-specified max walltime) before the reservation's start time. In constrast, the `killable` queue allows the scheduler to start a job even if it will *not* complete before a scheduled reservation. It enforces the following policies:

- Jobs will be killed if still running when a system outage begins.
- The scheduler will stop scheduling jobs in the `killable` queue (1) hour before a scheduled outage.
- Maximum-job-per-user limits are the same (i.e., in conjunction with) the batch queue.
- Any killed jobs will be automatically re-queued after a system outage completes.

debug Queue Policy

The debug queue is intended to provide faster turnaround times for the code development, testing, and debugging cycle. For example, interactive parallel work is an ideal use for the debug queue. It enforces the following policies:

- Production jobs are not allowed.
- Maximum job walltime of (1) hour.
- Limit of (1) job per user *regardless of the job's state*.
- Jobs receive a (2)-day priority aging boost for scheduling.

Warning: Users who misuse the debug queue may have further access to the queue denied.

Allocation Overuse Policy

Projects that overrun their allocation are still allowed to run on OLCF systems, although at a reduced priority. Like the adjustment for the number of processors requested above, this is an adjustment to the apparent submit time of the job. However, this adjustment has the effect of making jobs appear much younger than jobs submitted under projects that have not exceeded their allocation. In addition to the priority change, these jobs are also limited in the amount of wall time that can be used. For example, consider that job1 is submitted at the same time as job2. The project associated with job1 is over its allocation, while the project for job2 is not. The batch system will consider job2 to have been waiting for a longer time than job1. The adjustment to the apparent submit time depends upon the percentage that the project is over its allocation, as shown in the table below:

% Of Allocation Used	Priority Reduction
< 100%	0 days
100% to 125%	30 days
> 125%	365 days

System Reservation Policy

Projects may request to reserve a set of processors for a period of time through the reservation request form, which can be found on the [Special Requests](#) page. If the reservation is granted, the reserved processors will be blocked from general use for a given period of time. Only users that have been authorized to use the reservation can utilize those resources. Since no other users can access the reserved resources, it is crucial that groups given reservations take care to ensure the utilization on those resources remains high. To prevent reserved resources from remaining idle for an extended period of time, reservations are monitored for inactivity. If activity falls below 50% of the reserved resources for more than (30) minutes, the reservation will be canceled and the system will be returned to normal scheduling. A new reservation must be requested if this occurs. Since a reservation makes resources unavailable to the general user population, projects that are granted reservations will be charged (regardless of their actual utilization) a CPU-time equivalent to (`# of cores reserved`) * (`length of reservation in hours`).

8.15. Aprun Tips

• [\(Back to Top\)](#)

The following tips may help diagnose errors and improve job runtime by providing

- example solutions to common aprun [errors](#)
- suggestions to improve aprun [layout issues](#)
- tips to work around [node failures](#) .

Layout Suggestion: Avoiding Floating-Point Contention

Note: Because the layout of tasks within a node may negatively impact performance, you may receive an aprun warning notice if we detect that the specified aprun layout does not spread the tasks evenly over the node.

An aprun wrapper will parse the given layout options returning a warning if tasks are not spread equally over a node's compute units and/or numa nodes. You may see a warning similar to the following if the wrapper detects a possible non-

optimal layout: `APRUN usage: requested less processes than cores (-N 2) without using -j 1 to avoid floating-point unit contention` Each Titan [compute node](#) contains (1) AMD Opteron™ 6274 (Interlagos) CPU. Each CPU contains (2) die. Each die contains (4) "bulldozer" compute units. Each compute unit contains (2) integer cores and a shared floating point scheduler. By default, aprun will place 16 processes on a node. In this manner, pairs of processes placed on the same compute unit will contend for the compute unit's floating point scheduler. If your code is floating point intensive, sharing the floating point scheduler may degraded performance. You can override this behavior using the aprun options `-j` and `-S` to control process layout. **Please note:** Batch jobs can not share nodes; a batch job will be [charged](#) for an entire node (30 core-hours per node) regardless of actual CPU or GPU core utilization. For more information on titan's aprun options, node description, and layout examples, see the [Job Execution](#) section of titan's user guide. The following examples do not use all cores on a node but share compute units' floating point schedule. The examples assume:

- 16 cores per node
- 4 nodes allocated to batch job: `#PBS -l nodes=4`

aprun -n16 -S2 ./a.out

Problem:

All cores on the node are not used, but the tasks will be placed on the first compute unit of each NUMA node. Taking the default layout, 3 compute units on each NUMA node will sit idle.

Suggestion:

Using the `-j1` aprun flag the job will be spread out out such that only one integer core on each compute unit is used. This will prevent contention for the each compute unit's floating point scheduler.

```
aprun -n16 -S2 -j1 ./a.out
```

Note: When using the `-j` flag, a portion of a node's integer cores will sit idle. Batch jobs can not share nodes; a batch job will be [charged](#) for an entire node (30 core-hours per node) regardless of actual CPU or GPU core utilization.

aprun -n16 -N4 ./a.out

Problem:

All cores on the node are not used, but the tasks will be placed on the first two compute units of the node's first NUMA node. Taking the default layout, the node's second NUMA node will sit idle.

Suggestion:

Using the `-S` and `-j1` aprun flags the job will be spread out out such that each both NUMA nodes on a node are used and only one integer core on each compute unit is used. This will prevent contention for the each compute unit's floating point scheduler.

```
aprun -n16 -S2 -j1 ./a.out
```

Error: Requesting more resources than have been allocated

It is possible to ask aprun to utilize more cores than have been allocated to the batch job. Attempts to over-allocate a batch jobs' reserved nodes may result in the following message:

```
claim exceeds reservation's CPUs
```

The following examples result in over-allocation attempts. The examples assume

- 16 cores per node
- 4 nodes allocated to batch job: `#PBS -l nodes=4`

aprun -n128 ./a.out

Problem:

There are not enough cores allocated to the batch job to fulfill the request. 4 nodes requested with 16 cores per node provides 64 cores.

Corrections:

Request more nodes:

```
#PBS -lnodes=8
aprun -n128 ./a.out
```

Request fewer tasks:

```
aprun -n64 ./a.out
```

aprun -n32 -N8 -S2 ./a.out

Problem:

There are enough cores allocated (64) to fulfill the task request (-n32). There are also enough nodes allocated to run 8 cores per node (-N8 * 4 nodes). But, -S2 requests that aprun run only 2 tasks per numa node. Since there are only 2 numa nodes per node, only 4 cores could be allocated per node (4 cores * 4 nodes < 32).

Corrections:

The -N is not needed when -S is used. You could remove the -N flag and increase the number of tasks per NUMA node by increasing -S from 2 to 4:

```
aprun -n32 -S4 -j1 ./a.out
```

You could remove the -N flag and increase the number of nodes allocated to the batch job:

```
#PBS -lnodes=8
aprun -n32 -S2 ./a.out
```

For more information on titan's aprun options and layout examples, see the [Job Execution](#) section of titan's user guide.

Working Around Node Failure

With the large number of nodes on titan, you may experience node failure on occasion. You may see node failures the following ways:

- If a node fails between batch job allocation and the first aprun, you may see the following error:

```
claim exceeds reservation's CPUs
```

Note: This most often occurs when attempting to run on more resources than were allocated to the batch job. See the [requesting more resources than have been allocated](#) section for more information on this message when not related to node failure.

- If a node fails during an aprun job, the aprun process should terminate.

The following steps may be useful when dealing with node failure:

1. Request more nodes than are required by aprun.
2. Add a loop around aprun to check for success (**this check is code specific**) and re-run the aprun process on the additional allocated nodes upon error.

The following is a pseudo code example:

```
#PBS -lnodes=(a few more than you need)

while (not successful)
```



```
aprun -n (exact number you need) ./a.out

sleep 30

end while
```

The loop's purpose is to re-run the aprun process on the extra nodes in the case that the aprun process does not succeed. Upon completion of aprun, unless the success check determines that aprun completed successfully, the aprun will be re-run. If the aprun does not succeed due to a node issue, the aprun process should be re-run allowing the system to place the tasks on one of the extra node(s) instead of the troubled node. This process may allow the job to work through a node issue without exiting the batch system and re-entering the batch queue. Its success is dependent on how well you can tailor the success test to you code.

9. Development Tools

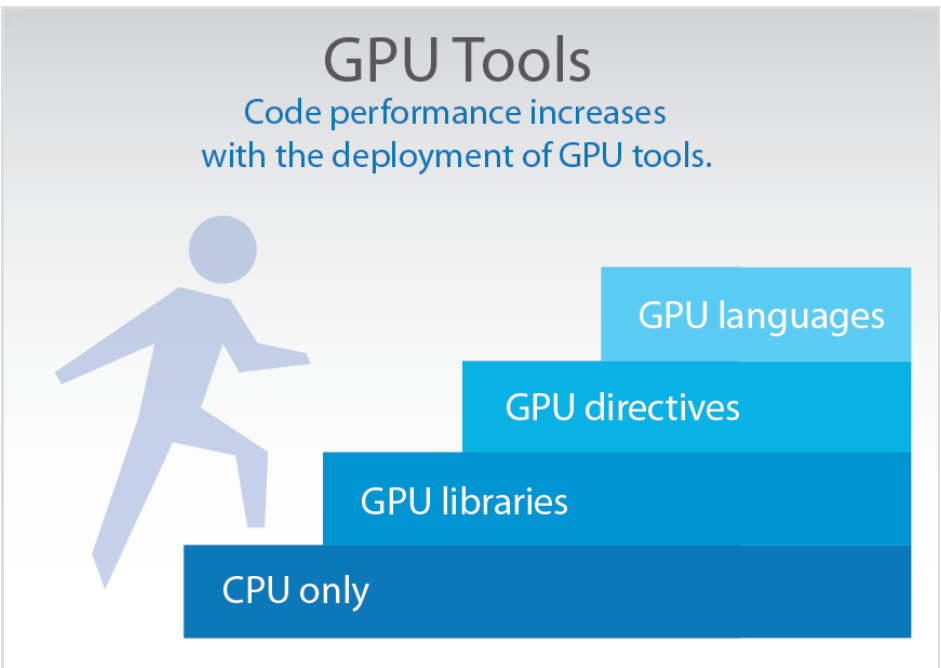
• [\(Back to Top\)](#)

The following section provides an overview of some of the information available in the [Accelerated Computing Guide](#).

Titan is configured with a broad set of tools to facilitate acceleration of both new and existing application codes. These tools can be broken down into (3) categories based on their implementation methodology:

- Accelerated Libraries
- Accelerator Compiler Directives
- Accelerator Languages/Frameworks

Each of these three methods has pros and cons that must be weighed for each program and are not mutually exclusive. In addition to these tools, Titan supports a wide variety of performance and debugging tools to ensure that, however you choose to implement acceleration into your program code, it is being done so efficiently.



9.1. GPU Accelerated Libraries

• [\(Back to Top\)](#)

Due to the performance benefits that come with GPU computing, many scientific libraries are now offering accelerated versions. If your program contains BLAS or LAPACK function calls, GPU-accelerated versions may be available. Magma, CULA, cuBLAS, cuSPARSE, and LibSciACC libraries provide optimized GPU linear algebra routines that require only minor changes to existing code. These libraries require little understanding of the underlying GPU hardware, and performance enhancements are transparent to the end developer.

For more general libraries, such as Trilinos and PETSc, you will want to visit the appropriate software development site to examine the current status of GPU integration. For Trilinos, please see the latest documentation at the [Sandia Trillinos](#) page. Similarly, [Argonne's PETSc Documentation](#) has a page containing the latest GPU integration information.

MAGMA

The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures. For C and Fortran code currently using LAPACK this should be a relatively simple port and does not require CUDA knowledge.

Use

This module is currently only compatible with the GNU programming environment:

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ module load cudatoolkit magma
```

To link in the MAGMA library:

```
cc -lcuda -lmagma -lmagmablas source.c
```

Resources

For comprehensive user manual please see the [MAGMA Documentation](#). A knowledgeable [MAGMA User Forum](#) is also available for personalized help. To see MAGMA in action see the following two PGI articles that include full example code of MAGMA usage with PGI Fortran: [Using MAGMA With PGI Fortran](#) and [Using GPU-enabled Math Libraries with PGI Fortran](#).

CULA

CULA is a GPU accelerated linear algebra library, mimicking LAPACK, that utilizes the NVIDIA CUDA. For C and Fortran code currently using LAPACK this should be a relatively simple port and does not require CUDA knowledge.

Use

CULA is accessed through the cula module, for linking it will be convenient to load the cuda module as well:

```
$ module load cula-dense cudatoolkit
```

To link in the CULA library:

```
$ cc -lcuda_core -lcuda_lapack source.c
```

Resources

A comprehensive [CULA Programmers Guide](#) is available for CULA that covers everything you need to know to use it. Once the module is loaded you can find up to date documentation in the **\$CULA_ROOT/doc** directory and examples in **\$CULA_ROOT/examples**. An example of using CULA with PGI Fortran is available in [Using GPU-enabled Math Libraries with PGI Fortran](#)

Running the examples:

Obtain an interactive job and load the appropriate modules:

```
$ qsub -I -A[projID] -lwalltime=00:30:00,nodes=1
$ module load cuda cula-dense
```

Copy the example files:

```
$ cd $MEMBERWORK/[projid]
$ cp -r $CULA_ROOT/examples .
$ cd examples
```

Now each example can be built and executed:

```
$ cd basicUsage
$ make build64
$ aprun basicUsage
```

cuBLAS/cuSPARSE

cuBLAS and cuSPARSE are NVIDIA provided BLAS GPU routines optimized for dense and sparse use respectively. If your program currently uses BLAS routines integration should be straight forward and minimal CUDA knowledge is needed. Although primarily designed for use in C/C++ code Fortran bindings are available.

Use

cuBLAS and cuSPARSE are accessed through the cublas header and need to be linked against the cublas library:

```
$ module load cudatoolkit
$ cc -lcublas source.c
```

Resources

The [CUBLAS](#) and [CUSPARSE](#) user guides are available to download from NVIDIA, these guides provide complete function listings as well as example code. The nvidia SDK provides sample code and can accessed using the instructions below. An example of using CUBLAS with PGI Fortran is available in [Using GPU-enabled Math Libraries with PGI Fortran](#).

Running the examples:

Obtain an interactive job and load the appropriate modules:

```
$ qsub -I -A[projID] -lwalltime=00:30:00,nodes=1
$ module switch PrgEnv-pgi PrgEnv-gnu
$ module load cudatoolkit nvidia-sdk
```

Copy the example files:

```
$ cd $MEMBERWORK/[projid]
$ cp -r $NVIDIA_SDK_PATH/CUDALibraries .
$ cd CUDALibraries
```

Now each example can be executed:

```
$ cd bin/linux/release
$ aprun simpleCUBLAS
```

LibSciACC

Cray's LibSciACC provides GPU enabled BLAS and LAPACK routines. LibSicACC provides two interfaces, automatic and manual. The Automatic interface is largely transparent to the programmer. LibSciACC will determine if the call is likely to benefit from GPU acceleration and if so will take care of accelerating the routine and the associated memory management. The manual interface provides an API to manage accelerator resources, providing more control to the programmer.

Use

It is recommended that the **craype-accel-nvidia35** module be used to manage LibSciAcc. The LibSciACC Automatic interface is currently compatible with the Cray and GNU programming environments:

```
$ module switch PrgEnv-pgi PrgEnv-cray
$ module load craype-accel-nvidia35
```

LibSciAcc will automatically be linked in when using the Cray provided compiler wrappers

```
$ cc source.c
```

```
$ ftn source.f90
```

Resources

The man page **intro_libsci_acc** provides detailed usage information. The environment variable

`$LIBSCI_ACC_EXAMPLES_DIR` specifies a directory containing several C and Fortran example codes.

cuFFT

CUFFT provides a set of optimized GPU fast Fourier routines that are provided by NVIDIA as part of the CUDA toolkit. The CUFFT library provides an API similar to FFTW for managing accelerated FFT's. The CUFFTW interface provides a FFTW3 interface to CUFFT to aid in porting existing applications.

Use

The cudatoolkit module will append the include and library directories required by CUFFT. When using NVCC or the GNU programming environment the library can then be added.

```
$ module load cudatoolkit
```

```
$ cc -lcufft source.c
```

```
$ nvcc -lcufft source.c
```

Resources

NVIDIA provides comprehensive documentation, including example code, available [Here](#). For an example of using the CUFFT with Fortran through the ISO_C_BINDING interface please see the following [example](#). The OLCF provides an [OpenACC and CUFFT interoperability tutorial](#).

cuRAND

CURAND is an NVIDIA provided random number generator library. CURAND provides both a host launched and device inalienable interface. Multiple pseudorandom and quasirandom algorithms are supported.

Use

The cudatoolkit module will append the include and library directories required by CURAND. When using NVCC or the GNU programming environment the library can then be added.

```
$ module load cudatoolkit
```

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ cc -lcurand source.c
```

```
$ nvcc -lcurand source.cu
```

Resources

NVIDIA provides comprehensive documentation, including example code, available [Here](#). For an example of using the CURAND host library please see the following OLCF [tutorial](#).

Thrust

Thrust is a CUDA accelerated C++ template library modeled after the Standard Template Library(STL). Thrust provides a high level host interface for GPU data management as well as an assortment of accelerated algorithms. Even if your application is not currently using the STL the easy access to many optimized accelerated algorithms Thrust provides is worth taking a look at.

Use

The cudatoolkit module will append the include and library directories required by Thrust. When using NVCC or the GNU programming environment the library can then be added.

```
$ module load cudatoolkit
```

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ CC source.cpp
```

```
$ nvcc source.cu
```

Resources

NVIDIA provides comprehensive documentation, including example code, available [Here](#). For an example of using Thrust please see the following OLCF [tutorial](#). The [Github page allows access to the Thrust source code, examples, and information on how to obtain help](#).

9.2. Accelerator Compiler Directives

• [\(Back to Top\)](#)

Accelerator compiler directives allow the compiler, guided by the programmer, to take care of low-level accelerator work. One of the main benefits of a directives-based approach is an easier and faster transition of existing code compared to low-level GPU languages. Additional benefits include performance enhancements that are transparent to the end developer and greater portability between current and future many-core architectures. Although initially several vendors provided their own set of proprietary directives OpenACC has now provided a unified specification for accelerator directives.

OpenACC

OpenACC aims to provide an open accelerator interface consisting primarily of compiler directives. Currently PGI, Cray, and CapsMC provide OpenACC implementations for C/C++ and Fortran. OpenACC aims to provide a portable cross platform solution for accelerator programming.

Using C/C++

PGI Compiler

```
$ module load cudatoolkit
$ cc -acc vecAdd.c -o vecAdd.out
```

Cray Compiler

```
$ module switch PrgEnv-pgi PrgEnv-cray
$ module load craype-accel-nvidia35
$ cc -h pragma=acc vecAdd.c -o vecAdd.out
```

CapsMC Compiler

```
$ module load cudatoolkit capsmc
$ cc vecAdd.c -o vecAdd.out
```

Using Fortran

PGI Compiler

```
$ module load cudatoolkit
$ ftn -acc vecAdd.f90 -o vecAdd.out
```

Cray Compiler

```
$ module switch PrgEnv-pgi PrgEnv-cray
```



```
$ module load craype-accel-nvidia35
$ ftn -h acc vecAdd.f90 -o vecAdd.out
```

CapsMC Compiler

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ module load cudatoolkit capsmc
$ ftn vecAdd.f90 -o vecAdd.out
```

Resources

The OpenACC specification provides the basis for all OpenACC implementations and is available [OpenACC specification](#) . In addition the implementation specific documentation may be of use. PGI has a site dedicated to collecting [OpenACC resources](#). Chapter 5 of the [Cray C and C++ Reference Manual](#) provides details on Crays implementation. CapsMC has provided an [OpenACC Reference Manual](#).

Tutorials

The OLCF provides [Vector Addition](#) and [Game of Life](#) example codes demonstrating the OpenACC accelerator directives.

Note: This section covers the PGI Accelerator directives, Not PGI's OpenACC implementation.

PGI Accelerator

The Portland Group provides accelerator directive support with their latest C and Fortran compilers. Performance and feature additions are still taking place at a rapid pace but it is currently stable and full featured enough to use in production code.

Use

To make use of the PGI accelerator directives the cuda module and pgi programming environment must be loaded:

```
$ module load cudatoolkit
$ module load PrgEnv-pgi
```

To specify the platform that the compiler directives should be applied to the **Target Accelerator** flag is used:

```
$ cc -ta=nvidia source.c
$ ftn -ta=nvidia source.f90
```

Resources

PGI provides a useful [web portal for Accelerator resources](#). The portal links to the **PGI Fortran & C Accelerator Programming Model** which provides a comprehensive overview of the framework and is an excellent starting point. In addition the ***PGI Accelerator Programming Model on NVIDIA GPUs article series by Michael Wolfe*** walks you through basic and advanced programming using the framework providing very helpful tips along the way. If you run into trouble PGI has a [user forum](#) where PGI staff regularly answer questions.

Examples

The OLCF provides [Vector Addition](#) and [Game of Life](#) example codes demonstrating the PGI accelerator directives.

Note: This section covers the CapsMC directives, Not Caps OpenACC implementation.

CapsMC

The core of CAPS Enterprises GPU directive framework is CapsMC. CapsMC is a compiler and runtime environment that interprets OpenHMPP and OpenACC directives and in conjunction with your traditional compiler (PGI, GNU, Cray or Intel C or Fortran compiler) creates GPU accelerated executables.

Use

To use CAPS accelerator framework you will need the cuda and capsmc modules loaded. Additionally a PGI, GNU, or Intel Programming environment must be enabled.

```
$ module load cudatoolkit
$ module load capsmc
$ module load PrgEnv-gnu
```

CapsMC modifies the Cray compiler wrappers, generating accelerator code and then linking it in without any additional flags.

```
$ cc source.c
$ ftn source.f90
```

Resources

CAPS provides several documents and code snippets to get you started with HMPP Workbench. It is recommended to start off with the [HMPP directives reference manual](#) and the [HMPPCG reference manual](#).

Examples

The OLCF provides [Vector Addition](#) and [Game of Life](#) example codes demonstrating the HMPP accelerator directives.

9.3. GPU Languages/Frameworks

• [\(Back to Top\)](#)

For complete control over the GPU, Titan supports CUDA C, CUDA Fortran, and OpenCL. These languages and language extensions, while allowing explicit control, are generally more cumbersome than directive-based approaches and must be maintained to stay up-to-date with the latest performance guidelines. Substantial code structure changes may be needed and an in-depth knowledge of the underlying hardware is often necessary for best performance.

NVIDIA CUDA C

NVIDIA's CUDA C is largely responsible for launching GPU computing to the forefront of HPC. With a few minimal additions to the C programming language, NVIDIA has allowed low-level control of the GPU without having to deal directly with a driver-level API.

Use

To setup the CUDA environment the `cudatoolkit` module must be loaded:

```
$ module load cudatoolkit
```

This module will provide access to NVIDIA supplied utilities such as the **nvcc** compiler, the CUDA visual profiler(**computeprof**), **cuda-gdb**, and **cuda-memcheck**. The environment variable **CUDAROOT** will also be set to provide easy access to NVIDIA GPU libraries such as **cuBLAS** and **cuFFT**.

To compile we use the NVIDIA CUDA compiler, **nvcc**.

```
$ nvcc source.cu
```

For a full usage walkthrough please see the supplied tutorials.

Resources

NVIDIA provides a comprehensive web portal for CUDA developer resources [here](#). The [developer documentation center](#)

contains the **CUDA C programming guide** which very thoroughly covers the CUDA architecture. The programming guide covers everything from the underlying hardware to performance tuning and is a must read for those interested in CUDA programming. Also available on the same downloads page are whitepapers covering topics such as **Fermi Tuning** and **CUDA C best practices**. The CUDA SDK is available for download as well and provides many samples to help illustrate C for CUDA programming technique. For personalized assistance NVIDIA has a very knowledgeable and active [developer forum](#).

Examples

The OLCF provides both a [Vector Addition](#) and [Game of Life](#) example code tutorial demonstrating CUDA C usage.

PGI CUDA Fortran

PGI's CUDA Fortran provides a well-integrated Fortran interface for low-level GPU programming, doing for Fortran what NVIDIA did for C. PGI worked closely with NVIDIA to ensure that the Fortran interface provides nearly all of the low-level capabilities of the CUDA C framework.

Usage

CUDA Fortran will be properly configured by loading the PGI programming environment:

```
$ module load PrgEnv-pgi
```

To compile a file with the **cuf** extension we use the PGI Fortran compiler as usual:

```
$ ftn source.cuf
```

For a full usage walkthrough please see the supplied tutorials.

Resources

PGI provides a comprehensive web portal for CUDA Fortran resources [here](#). The portal links to the [PGI Fortran & C Accelerator Programming Model](#) which provides a comprehensive overview of the framework and is an excellent starting point. The web portal also features a set of articles covering introductory material, device kernels, and memory management. If you run into trouble PGI has a [user forum](#) where PGI staff regularly answer questions.

Examples

The OLCF provides both a [Vector Addition](#) and [Game of Life](#) example code tutorial demonstrating CUDA Fortran usage.

OpenCL

The Khronos group, a non-profit industry consortium, currently maintains the OpenCL (Open Compute Language) standard. The OpenCL standard provides a common low-level interface for heterogeneous computing. At its core, OpenCL is composed of a kernel language extension to C (similar to CUDA C) and a C API to control data management and code execution.

Usage

The cuda module must be loaded for the OpenCL header files to be found and a PGI or GNU programming environment enabled:

```
$ module load PrgEnv-pgi
$ module load cudatoolkit
```

To use OpenCL you must include the OpenCL library and library path:

```
gcc -lOpenCL source.c
```

Resources

Khronos provides a [web portal for OpenCL](#). From here you can view the specification, browse the reference [pages](#), and get individual level help from the [OpenCL forums](#). A [developers page](#) is also of great use and includes tutorials and example code to get you started.

In addition to the general Khronos provided material users will want to check out the vendor-specific available information for capability and optimization details. Of main interest to OLCF users will be the [AMD](#) and [NVIDIA](#) OpenCL developer zones.

Examples

The OLCF provides both a [Vector Addition](#) and [Game of Life](#) example code tutorial demonstrating OpenCL usage.

10. Debugging and Optimizing Code on Titan

• ([Back to Top](#))

There are a number of code debugging and profiling tools available to users on Titan. This section will provide some general guidelines which should apply broadly to all applications, however, the focus will be on applications running on the Titan XK7 system.

DDT

DDT is the primary debugging tool available to users on Titan. DDT is a commercial debugger sold by Allinea Software, a leading provider of parallel software development tools for High Performance Computing. For more information on DDT, see the [DDT software](#) page, or Allinea's [DDT support](#) page.

Optimization Guide for AMD64 Processors

AMD offers guidelines specifically for serial code optimization on the AMD Opteron processors. Please see [AMD's Developer Documentation](#) site for whitepages and information on the latest generation of AMD processors.

CrayPAT

CrayPAT is a profiling tool that provides information on application performance. CrayPAT is used for basic profiling of serial, multiprocessor, multithreaded, and accelerated programs. More information can be found on the [CrayPAT software](#) page.

File I/O Tips

Spider, the OLCF's center-wide Lustre[®] file system, is configured for efficient, fast I/O across OLCF computational resources. You can find information about how to optimize your application's I/O on the [Spider](#) page.

10.1. Accelerator Performance Tools

• ([Back to Top](#))

To ensure efficient use of Titan a full suite of performance and analysis tools will be offered. These tools offer a wide variety of support from static code analysis to full runtime heterogeneous tracing.

NVPROF

NVIDIA's command line profiler, NVPROF, provides profiling for CUDA codes. No special steps are needed when compiling your code. The profiler includes tracing capability as well as the ability to provide many performance metrics, including FLOPS. The profiler data can be saved and imported into the NVIDIA visual profiler for easier analysis.

Running

To use NVPROF the cudatoolkit module must be loaded and PMI daemon forking disabled. To view the output in the NVIDIA Compute Visual Profiler [X11 forwarding must be enabled](#).

```
$ module load cudatoolkit
$ export PMI_NO_FORK=1
```

Although NVPROF doesn't provide MPI aggregated data the %h and %p output file modifiers can be used to create separate output files for each **host** and **process**.

```
$ aprun -n16 nvprof -o output.%h.%p ./gpu.out
```

A variety of metrics and events can be captured by the profiler. For example to output the number of double precision flops you may use the following:

```
$ aprun -n16 nvprof --metrics flops_dp -o output.%h.%p ./gpu.out
```

To see a list of all available metrics and events the following can be used:

```
$ aprun nvprof --query-metrics
$ aprun nvprof --query-events
```

To view the output in the NVIDIA visual profiler please see the following [NVIDIA documentation](#).

Resources

The **nvprof user guide** is available on the [NVIDIA Developer Documentation Site](#) and provides comprehensive coverage of the profiler's usage and features.

NVIDIA Command Line Profiler

NVIDIA's Command Line Profiler provides run-time profiling of CUDA and OpenCL code. No special steps are needed when compiling your code and any tool that utilizes CUDA or OpenCL code, including compiler directives and accelerator libraries, can be profiled. The profile data can be collected in .txt format or .csv format to be viewed with the NVIDIA visual profiler.

Running

To use the NVIDIA Command Line Profiler the cudatoolkit module must be loaded. To view the output in the NVIDIA Compute Visual Profiler [X11 forwarding must be enabled](#):

```
$ module load cudatoolkit
```

Although NVPROF doesn't provide MPI aggregated data the compute node HOSTNAME and %p output file modifiers can be used to create separate output files for each host and **process**. The %h modifier is not available in the command line profiler.

```
$ export COMPUTE_PROFILE=1
$ export COMPUTE_PROFILE_LOG=$MEMBERWORK/[projid]/output.$HOSTNAME.%p
$ aprun gpu.out
```

To view the output in the NVIDIA Visual Profiler please see the [NVIDIA Visual Profiler Users Guide](#).

Resources

The **NVIDIA Command Line Profiler User Guide** is available on [NVIDIA's Developer Documentation Site](#) and provides comprehensive coverage of the profilers usage and features.

VampirTrace

VampirTrace allows for temporal and spatial MPI enabled CPU/GPU runtime tracing. The produced OTF data can then be analyzed using the Vampir or Tau GUI visualizers.

Running

To run the vampirtrace module must be loaded, the cuda module must be loaded as well for GPU traces:

```
$ module load cudatoolkit
$ module load vampirtrace
```

Once loaded the program must be recompiled with one of the provided wrappers:

```
$ vtcc -vt:cc mpicc source.c
$ vtnvcc source.cu
```

To see the invoked compiler and its arguments:

```
$ vtcc -vt:verbose
```

For further help on wrapper compiler options:

```
$ vtcc -vt:help
```

Running the executable should then produce the trace files.

Resources

Additional information can be found on the [OLCF VampirTrace software page](#). The [VampirTrace User Manual](#) provides comprehensive coverage of VampirTrace usage.

Vampir

Vampir is a graphical user interface for analyzing OTF trace data. For small traces all analysis may be done on the users local machine running a local Vampir copy. For larger traces the GUI can be run from the users local machine while the analysis is done using VampirServer, running on the parallel machine.

Use

The easiest way to get started is to launch the Vampir GUI from an OLCF compute resource, however a slow network connection may limit usability.

```
$ module load vampir
$ vampir
```

Resources

Additional information may be found on the [OLCF VampirTrace software page](#). The [Vampir User Manual](#) provides comprehensive coverage of Vampir usage.

TAU

TAU provides profiling and tracing tools for C, C++, Fortran, and GPU hybrid programs. Generated traces can be viewed in the included Paraprof GUI or displayed in Vampir.

Use

A simple GPU profiling example could be preformed as follows:

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ module load tau cudatoolkit
$ nvcc source.cu -o gpu.out
```

Once the cuda code has been compiled **tau_exec -cuda** can be used to profile the code at runtime

```
$ aprun tau_exec -cuda ./gpu.out
```

The resulting trace file can then be viewed using paraprof

```
$ paraprof
```

Resources

Additional information may be found in the [OLCF TAU software page](#). The TAU [documentation website](#) contains a complete User Guide, Reference Guide, and even video tutorials.

CrayPAT

CrayPAT is a profiling tool that provides information on application performance. CrayPAT is used for basic profiling of serial, multiprocessor, multithreaded, and GPU accelerated programs.

Use

A simple GPU profiling example could be preformed as follows:

With PrgEnv-Cray:

```
$ module load craype-accel-nvidia35
$ module load perftools
```

With PrgEnv other than Cray:

```
$ module load cudatoolkit
$ module load perftools
```

Compiling:

```
$ nvcc -g -c cuda.cu
$ cc cuda.cu launcher.c -o gpu.out
$ pat_build -u gpu.out
$ export PAT_RT_ACC_STATS=all
$ pat_report gpu.out+*.xf
```

Resources

More information can be found on the [CrayPAT software](#) page. For more details on linking nvcc and compiler wrapper compiled code please see our tutorial on [Compiling Mixed GPU and CPU Code](#).

CAPS HMPP Wizard

The goal of the HMPP Wizard is to identify common compute kernels and provide domain specific advice on how to best optimize them. This is done statically through a GUI interface and works best when used in conjunction with other HMPP performance tools. Currently the HMPP Wizard is compatible with GNU and Intel compilers.

Running

To run either the GNU or Intel programming environment needs to be loaded along with the hmppwizard module. Additional setup is performed by a provided shell script.

```
$ module switch PrgEnv-pgi PrgEnv-gnu
$ module load cudatoolkit
$ module load hmpp-wizard-perfanalyzer
$ source hmppwizard-env.sh
```

Viewing a report is a multi-step process. The program is compiled, a report is generated, and an internet browser is launched to view the results.

```
$ hmppReport -o ./report gcc source.c
$ hmppReport --build -o ./report
$ firefox report/index.html
```

Resources

The [HMPP Wizard User Manual](#) provides a comprehensive overview of the use and features of the wizard.

Home	About OLCF	Leadership Science	Computing Resources	Center Projects	Support	Media Center	Summit SC14
	Overview	Biological Sciences	Titan Cray XK7	Adios	Getting Started	Center Reports	
	Leadership Team	Chemistry	Eos	CCI	System User Guides	Fact Sheets	
	Groups	Computer Science	EVEREST	eSiMon	KnowledgeBase	Media Kit	
	Org Chart	Earth Science	Rhea	File System Projects	Tutorials	Image Gallery	
	Careers	Materials Science	Sith	IOTA	Training Events		
	Visitor Information & Tours	Physics	Data Management	OpenSFS	My OLCF		
	Contact Us	2014 INCITE Projects	Visualization and Data Analysis	SWTools	Software		
		2014 ALCC Projects		XGAR	Known Issues		
					Documents & Forms		
					OLCF Policies		

