TITAN USER GUIDE

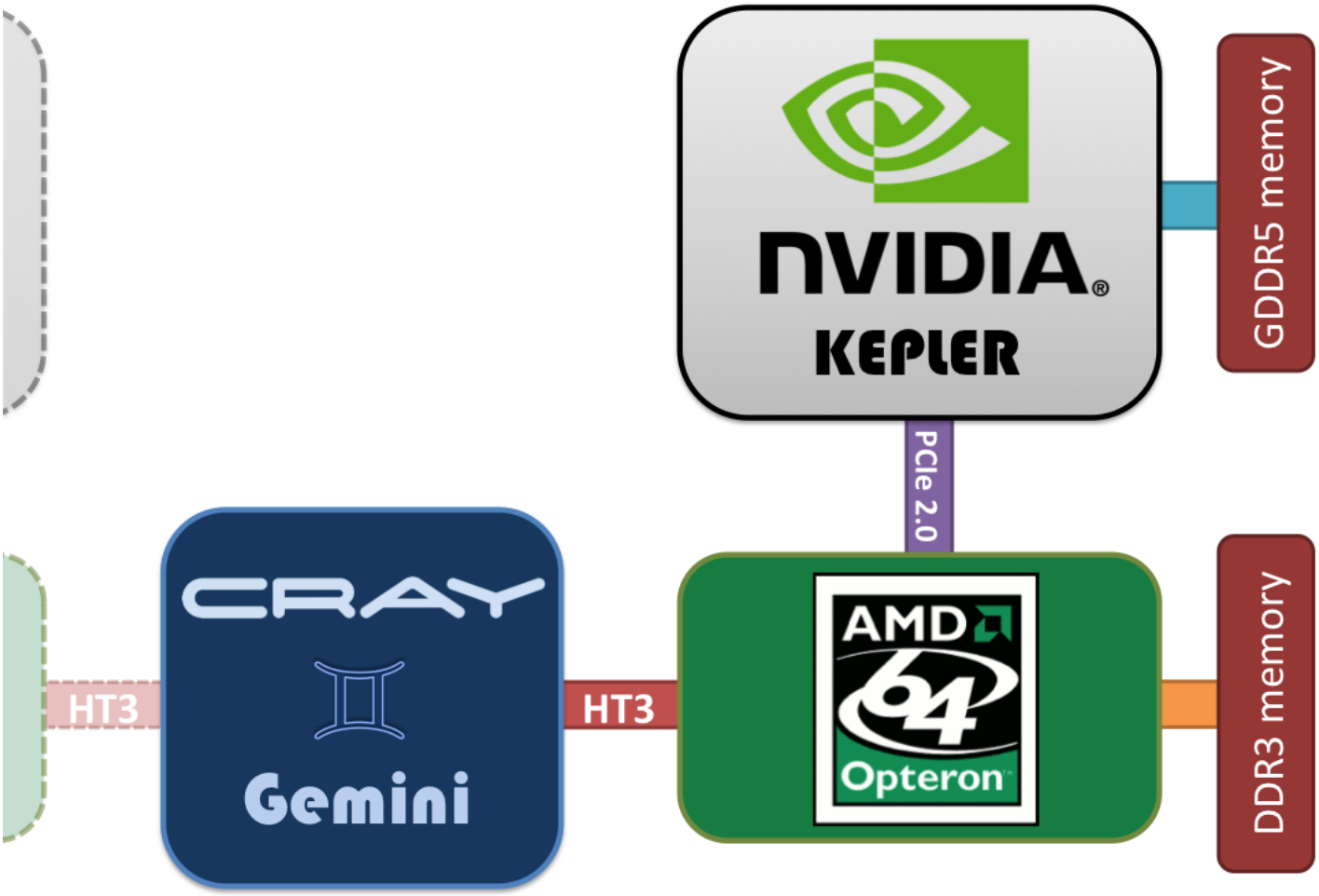
System Overview

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 Nodes

Titan contains 18,688 physical compute nodes, each with a processor, physical memory, and a connection to the custom Cray high-speed interconnect. Each compute node contains (1) 16-core 2.2GHz AMD Opteron™ 6274 (Interlagos) processor (/web/20190219213055/https://www.olcf.ornl.gov/running-jobs/#xk7-cpu-description) 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. In addition to the Opteron CPU, all of

below.



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

File Systems

The OLCF's center-wide Lustre® file system, named Spider (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#spider-the-centerwide-lustre-file-system), is available on Titan for computational work. With over 26,000 clients and (32) PB of disk space, it is one of the largest-scale Lustre® file systems in the world. A NFS-based file system provides User Home storage areas (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#user-home-directories-nfs) and Project Home storage areas (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#project-home-directories-nfs). Additionally, the OLCF's High Performance Storage System (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#hpss-high-performance-storage-system) (HPSS) provides archival spaces.

Operating System

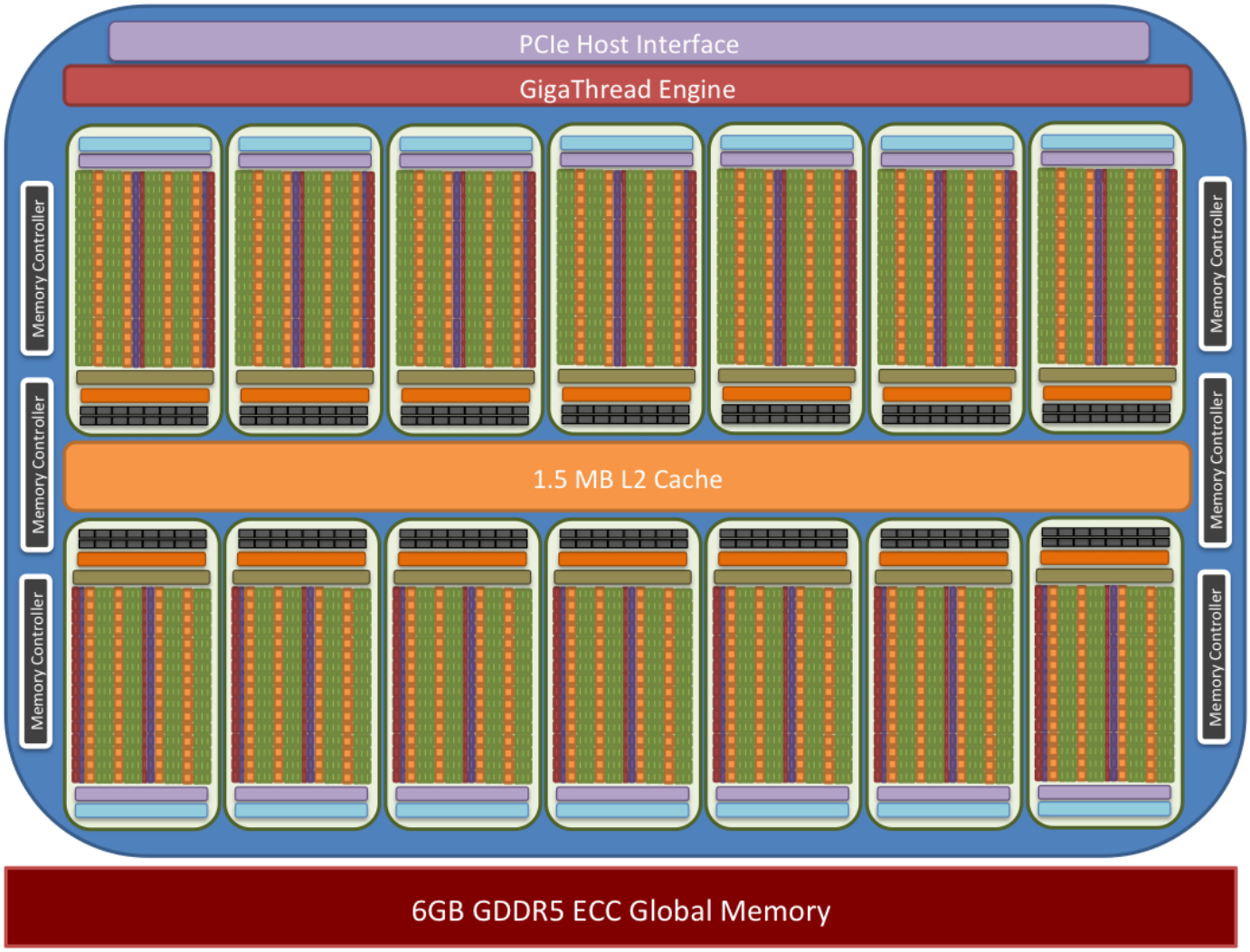
Titan employs the Cray Linux Environment as its OS. This consists of a full-featured version of Linux on the login nodes, and a Compute Node Linux microkernel on compute nodes. The microkernel is designed to minimize partition overhead allowing scalable, low-latency global communications.

NVIDIA K20X GPUs

K20X Accelerator Description

through the PCI express(PCIe) interface. The GigaThread engine is responsible for distributing work between each SM and (6) 64-bit memory controllers control access to 6 gigabytes of GDDR5 device(global) memory. A shared 1.5 megabyte L2 cache sits between global memory and each SM.

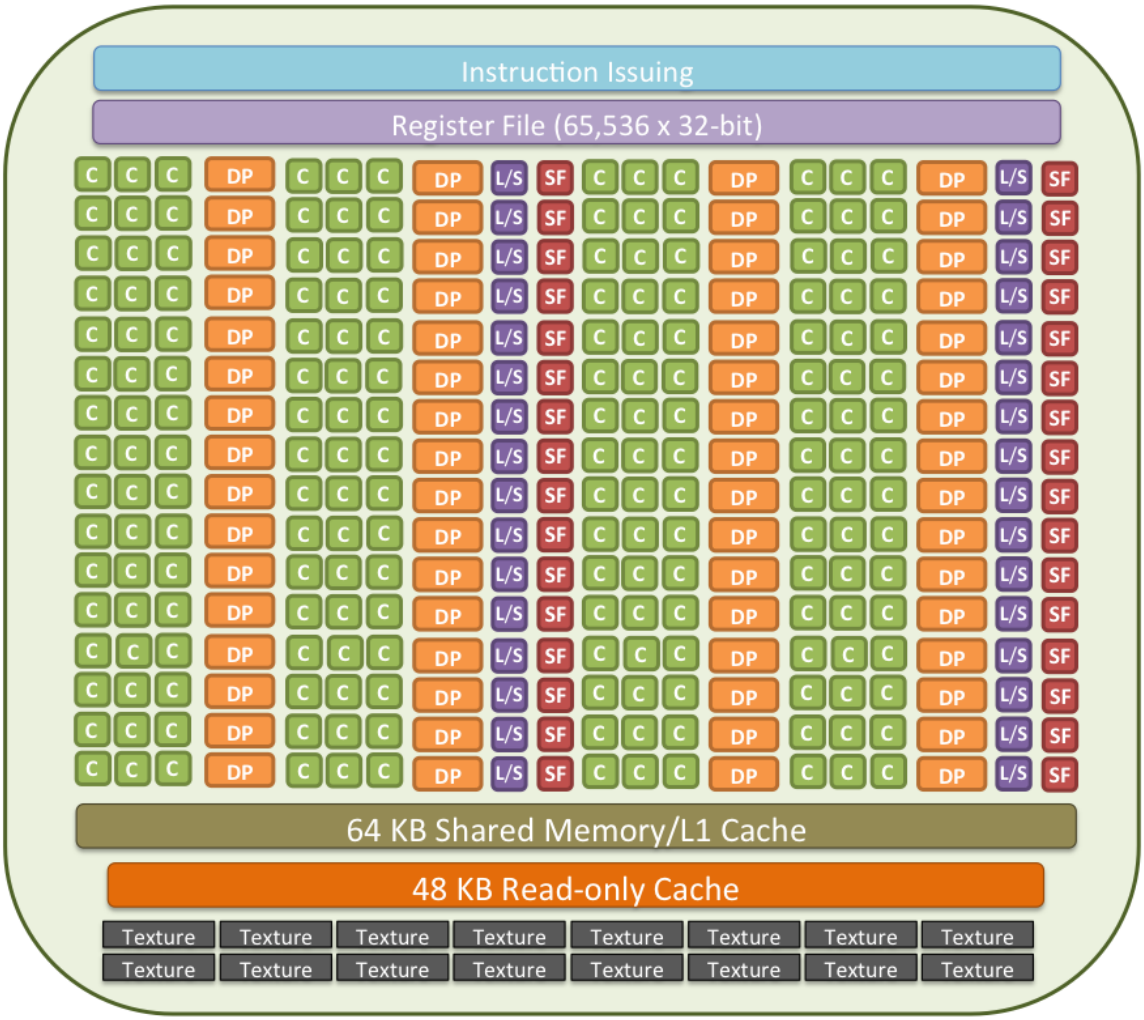
NVIDIA K20X accelerator



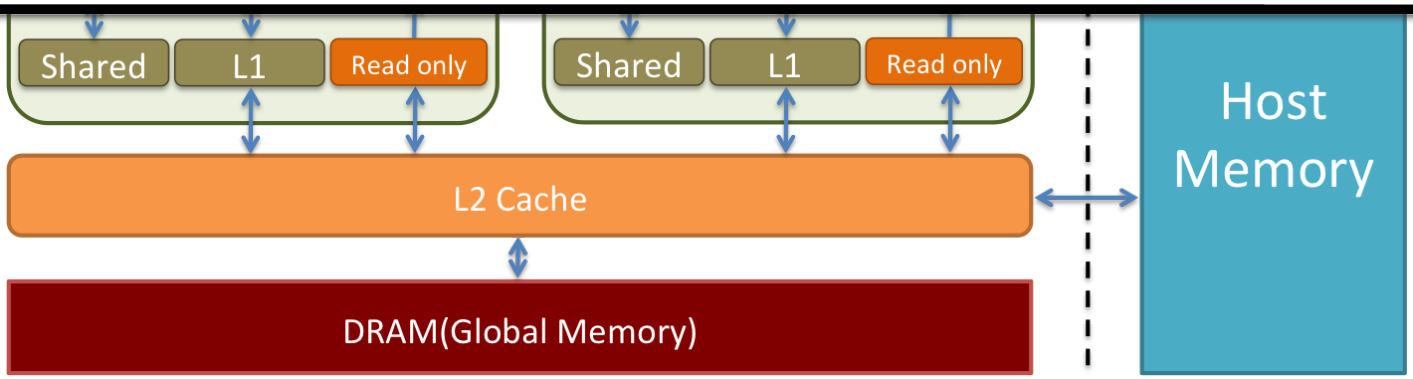
For arithmetic operations each SM contains (192) CUDA cores for handling single precision floating point and simple integer arithmetic, (64) double precision floating point cores, and (32) special function units capable of handling transcendental instructions. Servicing these arithmetic/logic units are (32)

Additionally, each SM has 48 kilobytes of on-chip read-only, user or compiler manageable, cache.

NVIDIA K20X SM



K20X Memory Hierarchy



Registers:

Each thread has access to private register storage. Registers are the lowest latency and highest bandwidth memory available to each thread but are a scarce resource. Each SM has access to (65,536) 32-bit registers that must be shared by all resident threads. An upper limit of 255 registers per thread is imposed by the hardware, if more thread private storage is required an area of device memory known as local memory will be used.

Shared memory/L1 cache:

Each SM has a 64 kilobyte area of memory that is split between shared memory and L1 cache. The programmer specifies the ratio between shared and L1.

Shared memory is a programmer managed low latency high bandwidth memory. Variables declared in shared memory are available to all threads within the same threadblock.

L1 cache on the K20X handles local memory (register spillage, stack data, etc.). Global memory loads are not cached in L1.

Read Only cache:

Each SM has 48 kilobytes of read only cache populated from global device memory. Eligible variables are determined by the compiler along with guidance from the programmer.

broadcast. Constants reside in device memory and have an aggregate max size of 64 kilobytes.

Texture:

Texture memory provides multiple hardware accelerated boundary and filtering modes for 1D,2D, and 3D data. Textures reside in global memory but are cached in the on SM 48 kilobyte read-only cache.

L2 cache:

Each K20X contains 1.5 megabytes of L2 cache. All access to DRAM, including transfers to and from the host, go through this cache.

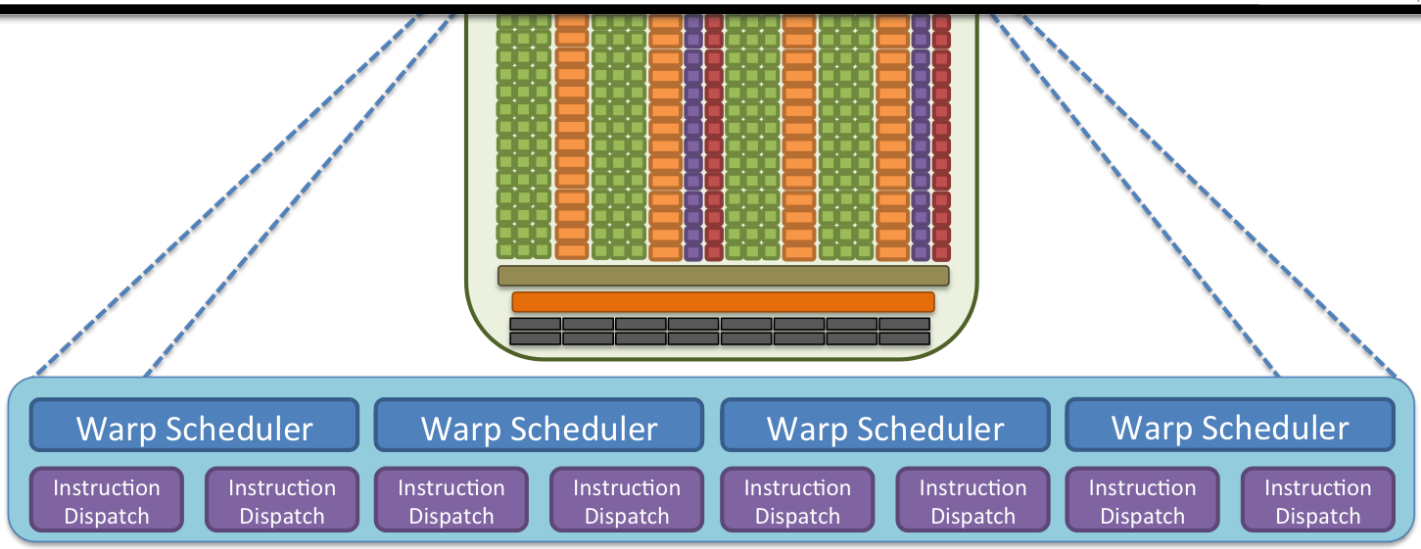
DRAM (Global memory):

Each K20X contains 6 gigabytes of GDDR5 DRAM. On titan ECC is enabled which reduces the available global memory by 12.5%; Approximately 5.25 gigabytes is available to the user. In addition to global memory DRAM is also used for local storage(in case of register spillage), constant, and texture memory.

K20X Instruction Issuing

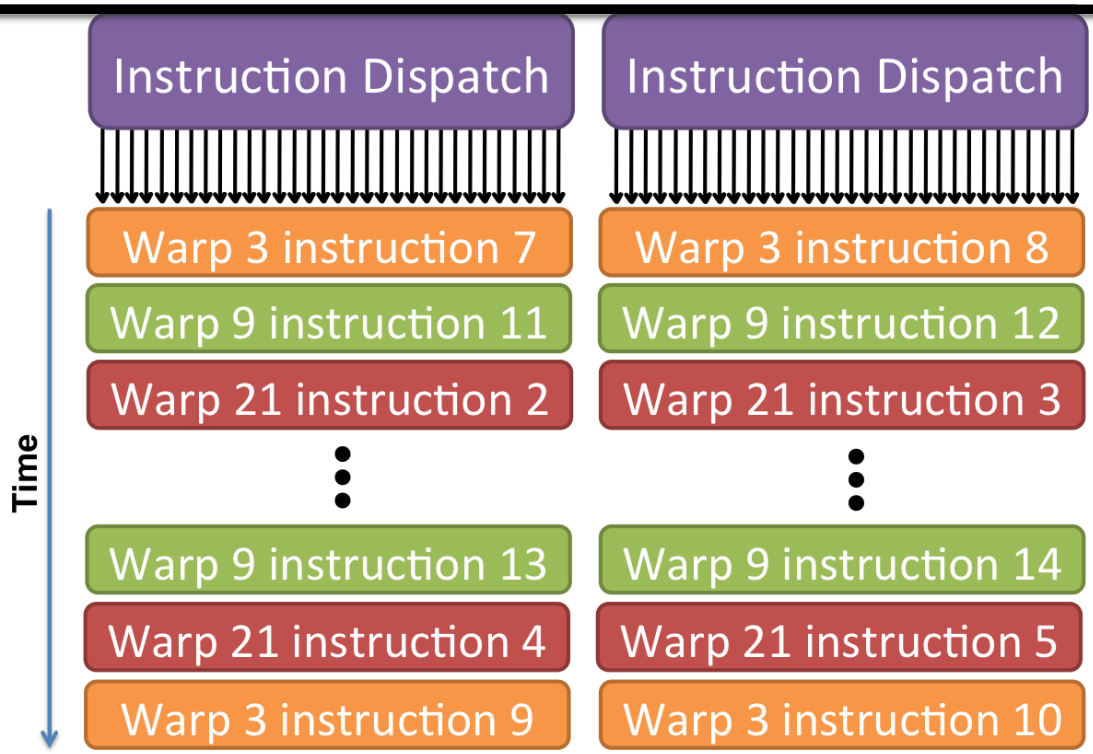
When a kernel is launched on the K20X, the GigaThread engine handles dispatching the enclosed thread blocks to available SMs. Each SM is capable of handling up to 2048 threads or 64 threadblocks, including blocks from concurrent kernels, if resources allow. All threads within a particular thread block must reside on a single SM.

Once a thread block is assigned to a particular SM, all threads contained in the block will execute entirely on that SM. At the hardware level on the SM, each thread block is broken down into chunks of 32 consecutive threads, referred to as warps. The K20X issues instructions at the warp level. That is to say an instruction is issued in vector like fashion to 32 consecutive threads at a time. This execution model is referred to as Single Instruction Multiple Thread, or SIMT.



Each Kepler SM has (4) warp schedulers. When a block is divided up into warps, each warp is assigned to a warp scheduler. Warps will stay on the assigned scheduler for the lifetime of the warp. The scheduler is able to switch between concurrent warps, originating from any block of any kernel, without overhead. When one warp stalls — that is, the next instruction can not be executed in the next cycle — the scheduler will switch to a warp that is able to execute an instruction. This low overhead warp swapping allows for instruction latency to be effectively hidden, assuming enough warps with issuable instructions reside on the SM.

Each warp scheduler has (2) instruction dispatch units. At each cycle the scheduler selects a warp, and if possible, two independent instructions will be issued to that warp. Two clock cycles are required to issue a double precision floating point instruction to a full warp.



K20X by the numbers

Model	K20X
Compute Capability	3.5
Peak double precision floating point performance	1.31 teraflops
Peak single precision floating point performance	3.95 teraflops
Single precision CUDA cores	2688
Double precision CUDA cores	896
CUDA core clock frequency	732 MHz

Memory size GDDR5(ECC on)	5.25 GB
L2 cache	1.5 MB
Shared memory/L1 configurable	64 KB per SM
Read-only cache	48 KB per SM
Constant memory	64 KB
32-bit Registers	65,536 per SM
Max registers per thread	255
Number of multiprocessors(SM)	14
Warp size	32 threads
Maximum resident warps per SM	64
Maximum resident blocks per SM	16
Maximum resident threads per SM	2048
Maximum threads per block	1024 threads
Maximum block dimensions	1024, 1024, 64
Maximum grid dimensions	2147483647, 65535, 65535

* ECC will reduce achievable bandwidth by 15+%

K20X Floating Point Considerations

between a particular algorithm run on the CPU and GPU since, in general, bit-for-bit agreement will not be possible.

The NVIDIA K20X provides full IEEE 754-2008 support for single and double precision floating point arithmetic. For simple floating point arithmetic (addition, subtraction, multiplication, division, square root, FMA) individual operations on the AMD Opteron and NVIDIA K20X should produce bit-for-bit identical values to the precision specified by the standard.

Transcendental operations (trigonometric, logarithmic, exponential, etc.) should not be expected to produce bit-for-bit identical values when compared against the CPU. Please refer to the CUDA programming guide (https://web.archive.org/web/20190219213055/http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#standard-functions) for single and double precision function ULP.

Floating point operations are not necessarily associative nor distributive. For a given algorithm, the number of threads, thread ordering, and instruction execution may differ between the CPU and GPU, resulting in potential discrepancies; as such it should not be expected that a series of arithmetic operations will produce bit-for-bit identical results.

Connecting

This section covers the basic procedures for accessing computational resources at the Oak Ridge Leadership Computing Facility.

Connect with SSH

with using plain-text communication.

> **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 (https://web.archive.org/web/20190219213055/https://www.vandyke.com/products/securecrt/) published by Van Dyke Software. For recent SecureCRT versions, the preferred authentications setting shown above can be made through the "connection properties" menu.

> **Note:** SSH multiplexing is disabled on all of the OLCF's user-facing systems. Users will receive an error message if they attempt to connect to an OLCF resource that tries to reuse an SSH control path. To ensure SSH connections will not attempt multiplexing, you will need to modify your `$HOME/.ssh/config` file by adding the following:

```
Host *.ccs.ornl.gov
  ControlMaster no
```

OLCF System Hostnames

designated OLCF hostnames for general user connections are as follows:

System Name	Hostname	RSA Key Fingerprints
Summit	summit.olcf.ornl.gov	MD5 08:d0:fe:3f:f3:41:96:9c:ae:73:73:a8:92:6c:79:34 SHA256 nA7X4qyPvtEpXWxG5MDeXEC8xfpmm0UMiLq/LkgM33I
Titan	titan.ccs.ornl.gov	MD5 77:dd:c9:2c:65:2f:c3:89:d6:24:a6:57:26:b5:9b:b7 SHA256 6Df2kqvj26HGadu3KDegPSeE/vbLYUjSIuot2AhsqL4
Rhea	rhea.ccs.ornl.gov	MD5 9a:72:79:cf:9e:47:33:d1:91:dd:4d:4e:e4:de:25:33 SHA256 AJIOXipN3Pgpcp/pFp8+g1zG09v0CiFpwQlc17OJ9S8
Eos	eos.ccs.ornl.gov	MD5 e3:ae:eb:12:0d:b1:4c:0b:6e:53:40:5c:e7:8a:0d:19 SHA256 LlznEESwYCT16sUty1ItSbO6n9FbqT0NNMVQMLQX3IY
Data Transfer Nodes	dtn.ccs.ornl.gov	MD5 b3:31:ac:44:83:2b:ce:37:cc:23:f4:be:7a:40:83:85 SHA256 GzKEIvoBKdeEH/yekGSOO8aSKibkl/KNTp9ZfYQq7VM
Home (machine)	home.ccs.ornl.gov	MD5 ba:12:46:8d:23:e7:4d:37:92:39:94:82:91:ea:3d:e9 SHA256 FjDs4sRAX8hglzA7TVkK22NzRKsjhDTTTdfeEAHwPEA
System Name	**Hostname**	**ECDSA Key Fingerprints**

Summit	summit.olcf.ornl.gov	MD5 cf:32:f9:35:fd:3f:2a:0f:ed:d3:84:b1:2d:f0:35:1b SHA256 m0iF9JJEoJu6jJGA8FFbSABlpKFYPGKbdmi25rFC1AI
Titan	titan.ccs.ornl.gov	MD5 54:2a:81:ed:75:14:d6:ec:fc:85:b8:4f:fb:b1:11:fa SHA256 afnEsujjMnIvC+1HFxnbsj4WmGa/Ka7tVn0nXHp2ebw
Rhea	rhea.ccs.ornl.gov	N/A
Eos	eos.ccs.ornl.gov	MD5 d7:bb:7d:a1:73:f7:92:42:43:e6:75:d6:31:29:87:8a SHA256 ddtmhprIkEcTt7OChHW6ITb0EjlCOdlP5DXMYC49Vog
Data Transfer Nodes	dtn.ccs.ornl.gov	MD5 04:35:86:9a:97:8a:19:74:32:21:29:5c:53:91:35:7a SHA256 owzOIoCC9VcYNXjAcOgHgIgRfmbtglfzhVtf8TM5qtQ
Home (machine)	home.ccs.ornl.gov	MD5 8a:92:0f:31:4d:38:2d:2c:ec:7d:53:ce:8b:46:73:d6 SHA256 0hc6SDou8vauFWgOaeXKUmhDSmKK8roj9jWpapV4qzc

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

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

RSA Key Fingerprints

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

```
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. (https://web.archive.org/web/20190219213055/mailto:help@olcf.ornl.gov)

Authenticating to OLCF Systems

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

Activating a new SecurID® Token (fob)

Follow the steps described below to set up your new SecurID Token (fob).

1. Initiate an SSH connection to home.ccs.ornl.gov using your OLCF username. (i.e., `ssh userid@home.ccs.ornl.gov`)
2. When prompted for a PASSCODE (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/connecting/#pins-passcodes-and-tokencodes), enter the 6 digits displayed on your token.

guides/titan/connecting/#pins-passcodes-and-tokencodes), answer with “y”.

4 You will then be prompted to enter a PIN. Enter a 4- to 8-character alphanumeric PIN you can remember. You will then be prompted to re-enter your PIN.

5 A message will appear stating that your PIN has been accepted. Press enter to continue.

6 Finally, you will be prompted again with “Enter PASSCODE”. This time enter both your PIN and the 6 digits displayed on your token before pressing enter.

7 Your PIN is now set and you are logged into home.ccs.ornl.gov.

Activating a New RSA Token

from OLCF

02:02

Using a SecurID® Token (fob)

enter `1234000987` when you are prompted for a PASSCODE.

> **Warning:** The 6-digit code displayed on the SecurID token can only be used once. If prompted for multiple PASSCODE entries, always allow the 6-digit code to change between entries. Re-using the 6-digit code can cause your account to be automatically disabled.

PINs, Passcodes, and Tokencodes

When users connect with RSA SecurID tokens, they are most often prompted for a PASSCODE. Sometimes, they are instead prompted for a PIN (typically only on initial setup) and other times they might be prompted to wait for the tokencode to change and enter the new tokencode. What do these terms mean?

The **TOKENCODE** is the 6-digit number generated by the RSA token.

The **PIN** is a (4) to (8)-digit number selected by the user when they initially set up their RSA token.

The **PASSCODE** is simply the user's PIN followed by the current tokencode.

These are relatively straightforward; however, there can be some confusion on initial setup. The first time a user connects with a new token (or, if for some reason the user requested that we clear the PIN associated with their token) they are prompted for a PASSCODE but in reality only enter a tokencode. This is because during this initial setup procedure a PIN does not exist. Since there is no PIN, the PASSCODE is the same as the tokencode in this rare case.

X11 Forwarding

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:

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.

Connecting to Internal OLCF Systems

Some OLCF systems are not directly accessible from outside the OLCF network. In order to access these systems, you must first log into *Home*.

```
ssh userid@home.ccs.ornl.gov
```

Once logged into Home, you can ssh into the desired (internal) system. Please see the Home (/web/20190219213055/https://www.olcf.ornl.gov/for-users/getting-started/#home) section on the Getting Started (/web/20190219213055/https://www.olcf.ornl.gov/for-users/getting-started/) page for more information (e.g. appropriate uses).

File Systems: Data Storage & Transfers

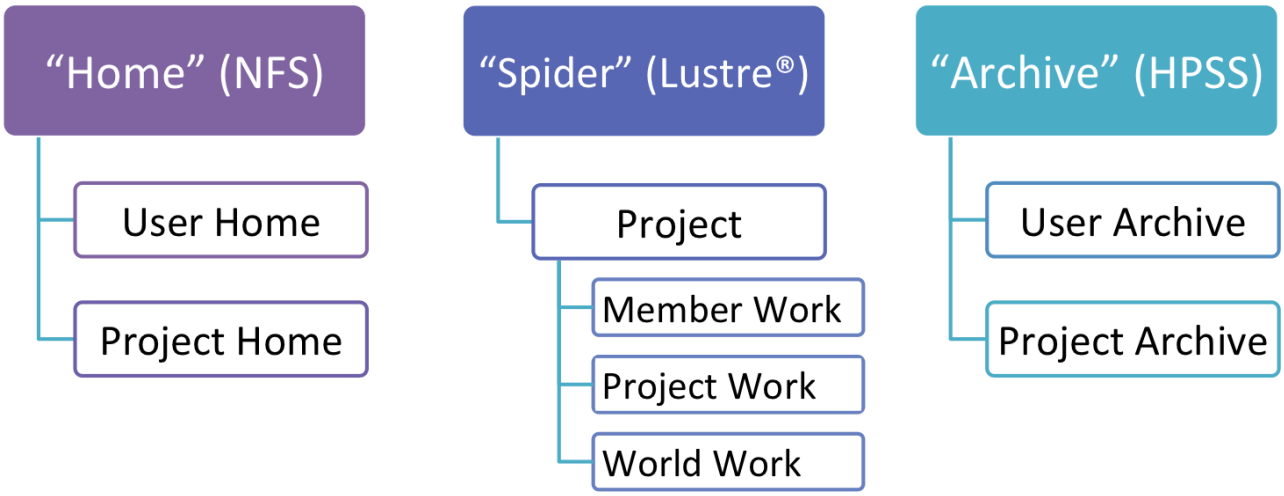
collaboration. The storage areas are mounted across all OLCF systems for maximum data availability.

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). These user storage areas are intended to house user-specific files.

Each project has a *Project Home* area on NFS, multiple *Project Work* areas on Lustre, and a *Project Archive* area on HPSS. These project storage areas are intended to house project-centric files.



Simple Guidelines

The following sections describe all available storage areas and detail intended use for each.
A brief description of each area and basic guidelines to follow are provided in the table below:

If you need to store…	then use…	at path…
Long-term data for routine access that is unrelated to a project	*User Home*	`$HOME`

Long-term data for archival access that is unrelated to a project	*User Archive*	`/home/$USER`
Long-term project data for routine access that's shared with other project members	*Project Home*	`/ccs/proj/[projid]`
Short-term project data for fast, batch-job access that you don't want to share	*Member Work*	`$MEMBERWORK/[projid]`
Short-term project data for fast, batch-job access that's shared with other project members	*Project Work*	`$PROJWORK/[projid]`
Short-term project data for fast, batch-job access that's shared with those outside your project	*World Work*	`$WORLDWORK/[projid]`
Long-term project data for archival access that's shared with other project members	*Project Archive*	`/proj/[projid]`

User-Centric Data Storage

Users are provided with several storage areas, each of which serve different purposes. These areas are intended for storage of user data, not for storage of project data.

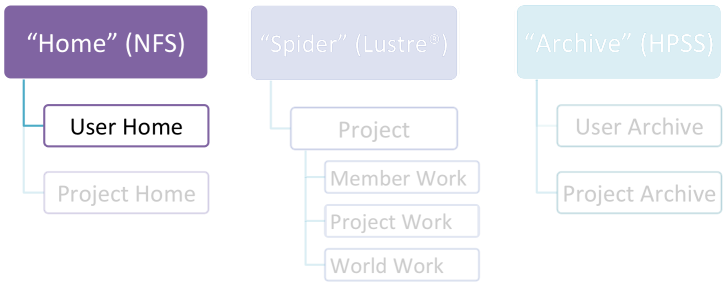
The following table summarizes user-centric storage areas available on OLCF resources and lists relevant polices.

Area	Path	Type	Permissions	Quota	Backups	Purged	Retention
User Home	$HOME	NFS	User-controlled	50 GB	Yes	No	90 days
User Archive	/home/$USER	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.

User Home Directories (NFS)

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 System (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:

```
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 are performed at regular intervals. Hourly backups for the past 24 hours, daily backups for the last 7 days, and 1 weekly backup are available. It is possible that the deleted files are available in one of those backups. The backup directories are named `hourly.*`, `daily.*` ,and `weekly.*` where `*` is the date/time stamp of the backup. For example, `hourly.2016-12-01-0905` is an hourly backup made on December 1, 2016 at 9:05 AM.

The backups are accessed via the `.snapshot` subdirectory. Note that if you do an `ls` (even with the `-a` option) of any directory you won't see a `.snapshot` subdirectory, but you'll be able to do "`ls .snapshot`" nonetheless. This will show you the hourly/daily/weekly backups available. The `.snapshot` feature is available in any subdirectory of your home directory and will show the online backup of that subdirectory. In other words, you don't have to start at `/ccs/home/$USER` and navigate the full directory structure; if you're in a /ccs/home subdirectory several "levels" deep, an "`ls .snapshot`" will access the available backups of that subdirectory.

To retrieve a backup, simply copy it into your desired destination with the `cp` command.

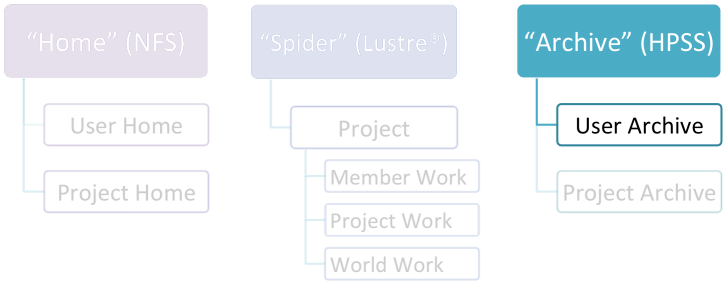
User Home Permissions

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).

User Website Directory

access to files at `http://users.nccs.gov/~user` (where `user` is your userid). If you are interested in having a user website directory created, please contact the User Assistance Center at help@olcf.ornl.gov (https://web.archive.org/web/20190219213055/mailto:help@olcf.ornl.gov).

User Archive Directories (HPSS)



The High Performance Storage System (HPSS) at the OLCF provides longer-term storage for the large amounts of data created on the OLCF compute systems. The mass storage facility consists of tape and disk storage components, servers, and the HPSS software. After data is uploaded, it persists on disk for some period of time. The length of its life on disk is determined by how full the disk caches become. When data is migrated to tape, it is done so in a first-in, first-out fashion.

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

User Archive Access

Each OLCF user receives an HPSS account automatically. Users can transfer data to HPSS from any OLCF system using the HSI or HTAR utilities. For more information on using HSI or HTAR, see the HPSS Best Practices (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-best-practices) section.

User Archive Accounting

(https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-archive-accounting) section.

For information on usage and best practices for HPSS, please see the section HPSS - High Performance Storage System (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-high-performance-storage-system) below.

Project-Centric Data Storage

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	Ret
Project Home	`/ccs/proj/[projid]`	NFS	770	50 GB	Yes	No	90
Member Work	`$MEMBERWORK/[projid]`	Lustre®	700 [1]	10 TB	**No**	**14 days**	14
Project Work	`$PROJWORK/[projid]`	Lustre®	770	100 TB	**No**	**90 days**	90

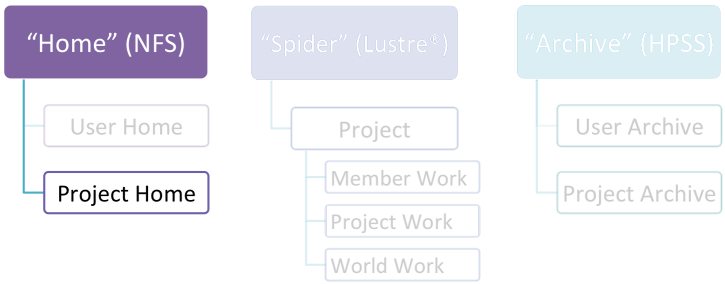
Area	Path	Type	Permissions	Quota	Backups	Purged	Ret
World Work	`$WORLDWORK/[projid]`	Lustre®	775	10 TB	**No**	**90 days**	90
Project Archive	`/proj/[projid]`	HPSS	770	100 TB [2]	**No**	No	90

> **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.

Project Home Directories (NFS)



Projects are provided with a Project Home storage area in the 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

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 in the table above (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/file-systems/#projectcentric-data-storage).

Project Home Backups

If you accidentally delete files from your project home directory, you may be able to retrieve them. Online backups are performed at regular intervals. Hourly backups for the past 24 hours, daily backups for the last 7 days, and 1 weekly backup are available. It is possible that the deleted files are available in one of those backups. The backup directories are named `hourly.*`, `daily.*` ,and `weekly.*` where `*` is the date/time stamp of the backup. For example, `hourly.2016-12-01-0905` is an hourly backup made on December 1, 2016 at 9:05 AM.

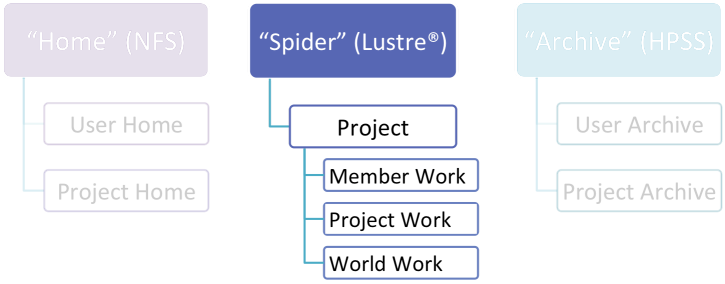
The backups are accessed via the `.snapshot` subdirectory. Note that if you do an `ls` (even with the `-a` option) of any directory you won't see a `.snapshot` subdirectory, but you'll be able to do "`ls .snapshot`" nonetheless. This will show you the hourly/daily/weekly backups available. The `.snapshot` feature is available in any subdirectory of your project home directory and will show the online backup of that subdirectory. In other words, you don't have to start at `/ccs/proj/abc123` and navigate the full directory structure; if you're in a /ccs/proj subdirectory several "levels" deep, an "`ls .snapshot`" will access the available backups of that subdirectory.

To retrieve a backup, simply copy it into your desired destination with the `cp` command.

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 includes the project's group members. All members of a project should also be members of that group-specific project. For example, all members of project

Project-Centric Work Directories



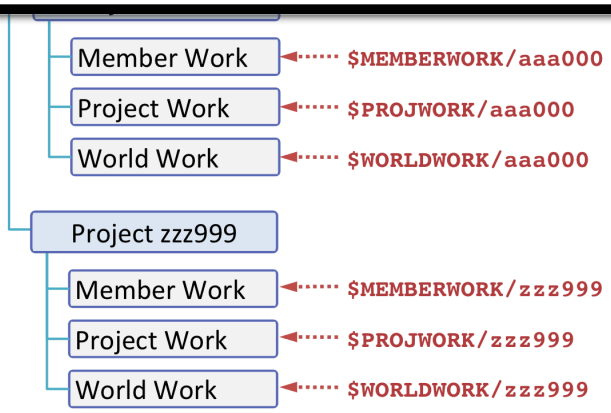
To provide projects and project members with high-performance storage areas that are accessible to batch jobs, projects are given (3) distinct project-centric work (i.e., scratch) storage areas within Spider (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#spider-the-centerwide-lustre-file-system), 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`

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.

Project-centric Work Directory Quotas

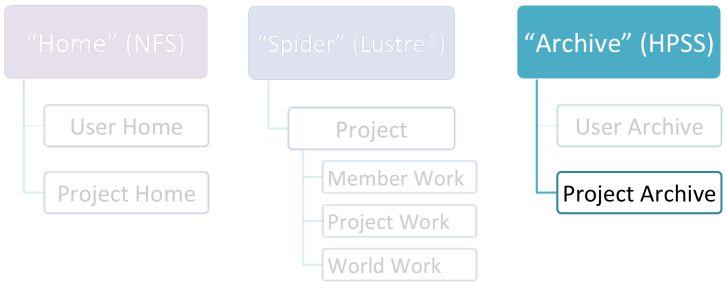
Soft quotas are enforced on project-centric work directories. The current limit is shown in the table above (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/file-systems/#projectcentric-data-storage). To request an increased 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.

Project Archive Directories

Projects are also allocated project-specific archival space on the High Performance Storage System (HPSS). The default quota is shown on the table above



(https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/file-systems/#projectcentric-data-storage). If a higher quota is needed, contact the User Assistance Center.

copies of project-related files.

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 HPSS Best Practices (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-best-practices) section.

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 HPSS Archive Accounting (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-archive-accounting) section.

For information on usage and best practices for HPSS, please see the section HPSS – High Performance Storage System (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#hpss-high-performance-storage-system) below.

Spider – the Center-Wide Lustre® File System

The OLCF's center-wide Lustre® file system, called *Spider*, is the operational work file system for most OLCF computational resources. As an extremely high-performance system, Spider has over 26,000 clients, providing 32 petabytes of disk space and can move data at more than 1 TB/s.

Spider is Center-Wide

- Titan (/web/20190219213055/https://www.olcf.ornl.gov/user-support/resource-user-guides/titan/)
- Rhea (/web/20190219213055/https://www.olcf.ornl.gov/user-support/resource-user-guides/rhea/)
- Eos (/web/20190219213055/https://www.olcf.ornl.gov/user-support/resource-user-guides/eos/)
- Data Transfer Nodes (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#employing-data-transfer-nodes)

> **Note:** Because the file system is shared by most OLCF computational resources, times of heavy load may impact file system responsiveness.

Spider is for Temporary Storage

Spider provides a location to temporarily store large amounts of data needed and produced by batch jobs. Due to the size of the file system, the area is not backed up. In most cases, a regularly running purge removes data not recently accessed to help ensure available space for all users. Needed data should be copied to more permanent locations.

> **Warning:** Spider provides temporary storage of data produced by or used by batch jobs. The space is not backed up. Users should copy needed data to more permanent locations.

Spider Comprises Multiple File Systems

Spider comprises (2) file systems:

File System	Path

atlas2	/lustre/atlas2/

Why two filesystems?

There are a few reasons why having multiple file systems within Spider is advantageous.

- **More Metadata Servers** – Currently, each Lustre filesystem can only utilize one Metadata Server (MDS) (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#basic-components-of-a-lustre-system). Interaction with the MDS is expensive; heavy MDS access will impact interactive performance. Providing (2) filesystems allows the load to be spread over (2) MDSs.
- **Higher Availability** – The existence of multiple filesystems increases our ability to keep at least one filesystem available at all times.

Available Directories on Spider

Every project is provided three distinct project-centric work storage areas: one that is only accessible by an individual project member (`$MEMBERWORK/[projid]`), one that is accessible by all project members (`$PROJWORK/[projid]`), and one that is accessible by all users of the system (`$WORLDWORK/[projid]`).

For more details, see the section on Project-Centric Work Directories (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#projectcentric-work-directories).

> **IMPORTANT!** Files in lustre directories are *not* backed up and are *purged* on a regular basis according to the timeframes listed in the Data Management Policy (/web/20190219213055/https://www.olcf.ornl.gov/olcf-policy-guide/).

Current Configuration of Spider

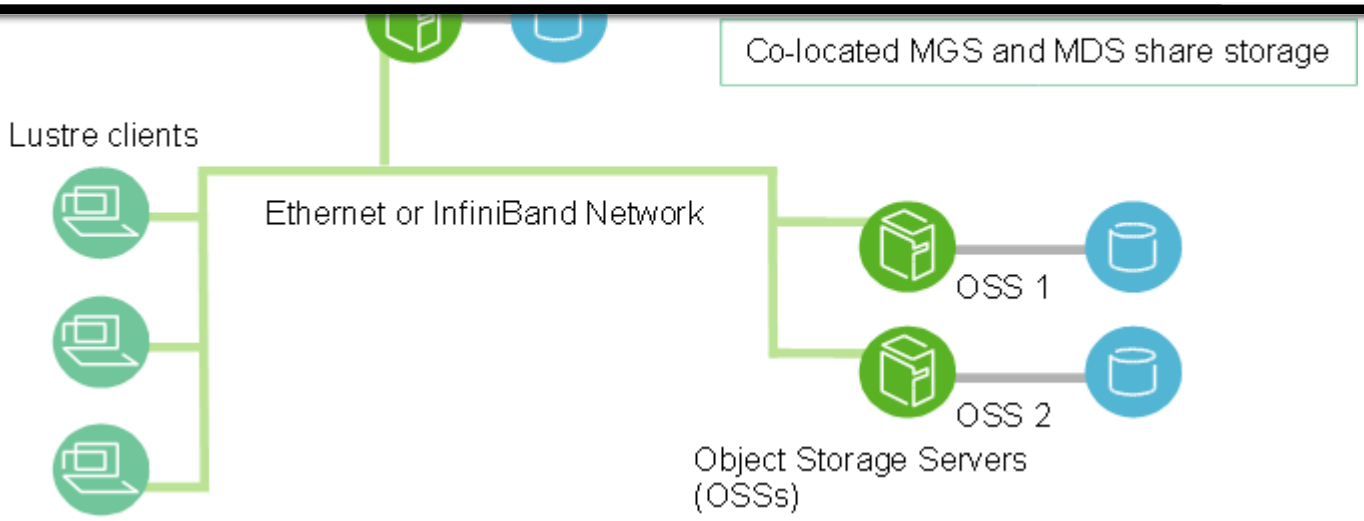
	atlas1	atlas2

Number of OSTs	1008	1008
Default stripe count	4	4
Default stripe size	1 MB	1 MB

Lustre® Basics

Basic Components of a Lustre® System

- *Metadata Server (MDS)* – The MDS makes metadata stored in the MDT available to Lustre clients. Each MDS manages the names and directories in the Lustre filesystem and provides network request handling for the MDT.
- *Metadata Target (MDT)* – The MDT stores metadata.
- *Object Storage Server (OSS)* – The OSS node provides file service and network request handling for one or more local OSTs.
- *Object Storage Target (OST)* – The OST stores file data (chunks of files) as data objects on one or more OSSs. A single file may be striped across one or more OSTs. When a file is striped across multiple OSTs, chunks of the file will exist on more than one OST.
- *Lustre Clients* – The nodes which mount the lustre file system are considered to be lustre clients. The service and computational nodes on titan are, for example, lustre clients.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2013/01/Basic_Cluster.png)

Basic Open/Write

The Metadata Server (MDS) stores information about each file including the number, layout, and location of the file's stripe(s). A file's data are stored on one or more Object Storage Targets (OSTs).

To access a file, the Lustre client must obtain a file's information from the MDS. Once this information is obtained, the client will interact directly with the OSTs on which the file is striped.

> **Warning:** Interaction with the Metadata Server (MDS) is expensive. Limiting tasks that require MDS access (e.g. directory operations, creating/opening/closing files, `stat` -ing files) will help improve file system interaction performance.

It is important to note that Spider 2, OLCF's Lustre parallel file system, is a shared resource. As such, the I/O performance observed can vary depending on the particular user jobs running on the system. Information about OLCF I/O best practices can be found here (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2013/01/olcf-user-meeting-2015-io-best-practices-sarp.pdf).

File Striping

A file may exist on one OST or multiple OSTs. If chunks of a file exist on multiple OSTs, the file is striped across the OSTs.

Advantages of Striping a File Across Multiple OSTs

- *File Size* – By placing chunks of a file on multiple OSTs the space required by the file can also be spread over the OSTs. Therefore, a file's size is not limited to the space available on a single OST.
- *Bandwidth* – By placing chunks of a file on multiple OSTs the I/O bandwidth can also be spread over the OSTs. In this manner, a file's I/O bandwidth is not limited to a single OST.

Disadvantages of Striping a File Across Multiple OSTs

- *Increased Overhead* – By placing chunks of a file across multiple OSTs, the overhead needed to manage the file separation, network connections, and multiple OSTs increases.
- *Increased Risk* – By placing chunks of a file across multiple OSTs, the odds that an event will take one of the file's OSTs down or impact data transfer increases.

File/Directory Stripe Patterns

When a file or directory is created it will inherit the parent directory's stripe settings. However, users have the ability to alter a directory's stripe pattern and set a new file's stripe pattern.

Users have the ability to alter the following stripe settings:

Setting	Default	Description
stripe count	4	Number of OSTs to stripe over
stripe size	1 MB	File chunk size in bytes

> **Warning:** Stripe counts over 512 have a negative impact on system and file performance. Please do not create files with stripe counts over 512.

File Chunk Creation

When a file's size is greater than the set stripe size, the file will be broken down into enough chunks of the specified stripe size to contain the file.

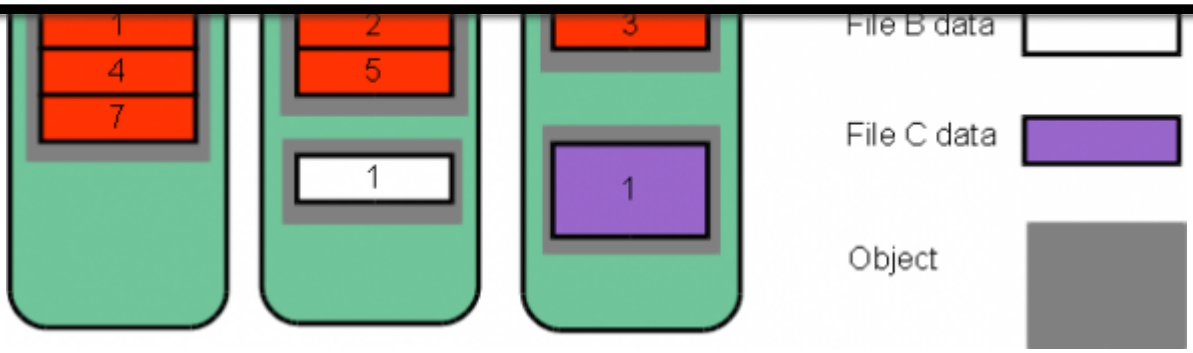
File Chunk Placement

When a file contains multiple chunks, and a stripe count greater than 1 is used, the file's chunks will be placed on OSTs in a round robin fashion.

Basic Striping Examples

The following example shows various Lustre striping patterns over 3 OSTs for 3 different file sizes. In all three cases the default stripe size of 1MB and the default stripe index (i.e. -1) are used. Note that object 6 in File A is not pictured because the corresponding data has not been written, resulting in a sparse file — Lustre does not create unnecessary objects in the underlying file system.

- File A is > 5 MB and < 7 MB, stripe count = 3
- File B is < 1 MB
- File C is > 1 MB and < 2 MB, stripe count = 1



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2013/01/File_Striping1.png)

Choosing a Stripe Pattern

A stripe pattern should be set based on a code's I/O requirements. When choosing a stripe pattern consider the following:

Stripe Count

Over-striping can negatively impact performance by causing small chunks to be written to each OST. This may under utilize the OSTs and the network. In contrast, under-striping might can place too much stress on individual OSTs, which may also cause resource and request contention. The following table provides general guidelines that can be used when choosing a stripe count for your files:

File size	Recommended Stripe Count	Notes
≤ 1 TB	4	This is the default stripe count
1 TB < 50 TB	File size / 100 GB	An 18 TB file would use 18 TB / 100 GB = 180 stripes
>50 TB	512	Stripe counts > 512 can have negative impact on performance. See warning below.

guidelines will ensure that chunks on individual OSTs have a reasonable size.

> **Warning:** Stripe counts over 512 have a negative impact on system and file performance. Please do not create files with stripe counts over 512. If you are working with files of 50 TB or more, please contact help@olcf.ornl.gov (https://web.archive.org/web/20190219213055/mailto:help@olcf.ornl.gov) for more guidelines specific to your use case.

Stripe Size

The default stripe size is 1 MB; i.e., Lustre sends data in 1 MB chunks. It is not recommended to set a stripe size less than 1 MB or greater than 4 MB.

Stripe Index

The default stripe index allows the system to select the OST on which the first data chunk will be placed. The default stripe index should always be used allowing the system to choose the OSTs. Forcing the use of specific OSTs by setting the stripe index prevents the system from managing the OST load and can unnecessarily cause high OST load. It can also cause a new file to be striped across an OST that is down or write only; this would cause a file creation to unnecessarily fail.

> **Warning:** The default stripe index, i.e. (-1), should always be used. Forcing the use of specific OSTs by setting the stripe index hinders the system's OST management.

Viewing the Striping Information

The `lfs getstripe` command can be used to view the attributes of a file or directory. The following example shows that `file1` has a stripe of (6) on OSTs 19, 59, 70, 54, 39, and 28:

```
70      28656421      0x1b54325      0
54      28652653      0x1b5346d      0
39      28850966      0x1b83b16      0
28      28854363      0x1b8485b      0
```

The following example shows that directory `dir1` has a stripe count of (6), the stripe size is set to (0) (i.e., use the default), and the stripe index/offset is set to (-1) (i.e., use the default).

```
$ lfs getstripe dir1
  ...
  dir1
  stripe_count: 6 stripe_size: 0 stripe_offset: -1
  ...
```

More details can be found in the lfs man page (`$ man lfs`).

Altering the Striping Pattern

A user can change the attributes for an *existing directory* or set the attributes when creating a *new file* in Lustre by using the `lfs setstripe` command. An existing file's stripe may not be altered.

Warning: The default stripe index, i.e, (-1), should always be used. Forcing the use of specific OSTs by setting the stripe index hinders the system's OST management.

Note: Files and directories inherit attributes from the parent directory. An existing file's stripe may *not* be altered.

Creating New Files

The following will create a zero-length file named `file1` with a stripe count of (16):

```
$ lfs setstripe -c 16 file1
```

you can create a new directory with the needed attributes and copy the existing files into the newly created directory. In this manner the files should inherit the directory's attributes.

Alter Existing Directories

The following example will change the stripe of directory `dir1` to (2).

```
$ lfs setstripe -c 2 dir1
```

More details can be found in the lfs man page (`$ man lfs`).

Viewing OST Storage

The `lfs df` command can be used to determine the amount of data stored on each Object Storage Target (OST).

The following example shows the size, used, and available space in human readable format for the /lustre/atlas2 filesystem:

```
$ lfs df -h /lustre/atlas2
   UUID                    bytes        Used     Available  Use%      Mounted on
 atlas2-OST0000_UUID       14.0T        3.0T       10.3T    22%   /lustre/atlas2[OST:
0]
 atlas2-OST0001_UUID       14.0T        3.0T       10.3T    22%   /lustre/atlas2[OST:
1]
 atlas2-OST0002_UUID       14.0T        3.2T       10.1T    24%   /lustre/atlas2[OST:
2]
 atlas2-OST0003_UUID       14.0T        3.0T       10.3T    23%   /lustre/atlas2[OST:
3]
 atlas2-OST0004_UUID       14.0T        3.2T       10.0T    24%   /lustre/atlas2[OST:
4]
 ...
 atlas2-OST03ee_UUID       14.0T        2.0T       11.2T    15%   /lustre/atlas2[OST:1
006]
 atlas2-OST03ef_UUID       14.0T        2.0T       11.2T    15%   /lustre/atlas2[OST:1
007]

 filesystem summary:       13.8P        3.0P       10.1P    23%   /lustre/atlas2
```

You can see a file or directory's associated OSTs with `lfs getstripe`. `lfs df` can then be used to see the usage on each OST.

More details can be found in the lfs man page (`man lfs`).

Associating a Batch Job with a File System

Through the PBS `gres` option, users can specify the scratch area used by their batch jobs so that the job will not start if that file system becomes degraded or unavailable.

Creating a Dependency on a Single File System

Line (5) in the following example will associate a batch job with the atlas2 file system. If atlas2 becomes unavailable prior to execution of the batch job, the job will be placed on hold until atlas2 returns to service.

```
1: #!/bin/csh
2: #PBS -A ABC123
3: #PBS -l nodes = 16000
4: #PBS -l walltime = 08:00:00
5: #PBS -l gres=atlas2
6:
7: cd $MEMBERWORK/abc123
8: aprun -n 256000 ./a.out
```

Creating a Dependency on Multiple File Systems

The following example will associate a batch job with the atlas1 and atlas2 file systems. If either atlas1 or atlas2 becomes unavailable prior to execution of the batch job, the job will be placed on hold until both atlas1 *and* atlas2 are in service.

```
-l gres=atlas1%atlas2
```

though the option `-l gres=atlas1%atlas2` had been applied to the batch submission.

> **Note:** To help prevent batch jobs from running during periods where a Spider file system is not available, batch jobs that do not explicitly specify the PBS `gres` option will be given a dependency on *all* Spider file systems.

Why Explicitly Associate a Batch Job with a File System?

- Associating a batch job with a file system will prevent the job from running if the file system becomes degraded or unavailable.
- If a batch job only uses (1) spider file systems, specifying the file systems explicitly instead of taking the default of all (2), would prevent the job from being held if a file system not used by the job becomes degraded or unavailable.

Verify/View Batch Job File System Association

The `checkjob` utility can be used to view a batch job's file system associations. For example:

```
$ qsub -lgres=atlas2 batchscript.pbs
851694.nid00004
$ checkjob 851694 | grep "Dedicated Resources Per Task:"
Dedicated Resources Per Task: atlas2: 1
```

Spider Best Practices

The following best practices can help achieve better I/O performance from your applications running on Spider.

Edit/Build Code in User Home and Project Home Areas Whenever Possible

Spider is built for large, contiguous I/O. Opening, closing, and `stat` -ing files are expensive tasks in Lustre. This, combined with periods of high load from compute jobs, can make basic editing and code builds noticeably slow. To work around this, users are encouraged to edit and build codes in their User

accessible to compute nodes which limits the possible resource load.

When using the `vi` editor, the default behavior is to create an opened file's temporary file in the current directory. If editing on a Spider file system, this can result in slowdown. You can specify that `vi` create temporary files in your User Home area by modifying your `/ccs/home/$USER/.vimrc` file:

```
$ cat ~/.vimrc
set swapsync=
set backupdir=~
set directory=~
```

Use `ls -l` Only Where Absolutely Necessary

Consider that `ls -l` must communicate with *every* OST that is assigned to any given listed file. When multiple files are listed, `ls -l` becomes a very expensive operation. It also causes excessive overhead for other users.

Open Files as Read-Only Whenever Possible

If a file to be opened is not subject to write(s), it should be opened as read-only. Furthermore, if the access time on the file does not need to be updated, the open flags should be `O_RDONLY | O_NOATIME`. If this file is opened by all files in the group, the master process (rank 0) should open it `O_RDONLY` with all of the non-master processes (rank > 0) opening it `O_RDONLY | O_NOATIME`.

Read Small, Shared Files from a Single Task

If a shared file is to be read and the data to be shared among the process group is less than approximately 100 MB, it is preferable to change the common code shown below (in C):

```
// Check file descriptor
iFD=open( PathName, O_RDONLY | O_NOATIME, 0444 );

// Check number of bytes read
iRead=read( iFD, cBuf, SIZE );
```

...to the code shown here:

```
int  iRank;
int  iRead;
char cBuf[SIZE];

MPI_Comm_rank( MPI_COMM_WORLD, iRank );
if(!iRank) {

  // Check file descriptor
  iFD=open( PathName, O_RDONLY | O_NOATIME, 0444 );

  // Check number of bytes read
  iRead=read( iFD, cBuf, SIZE );

}
MPI_Bcast( cBuf, SIZE, MPI_CHAR, 0, MPI_COMM_WORLD );
```

Similarly, in Fortran, change the code shown below:

```
INTEGER iRead
CHARACTER cBuf(SIZE)

OPEN(1, FileName)
READ(1,*) cBuf
```

...to the code shown here:

```
CHARACTER cBuf(SIZE)

CALL MPI_COMM_RANK(MPI_COMM_WORLD, iRank, ierr)
IF (iRank .eq. 0) THEN
  OPEN(UNIT=1,FILE=PathName,ACTION='READ')
  READ(1,*) cBuf
ENDIF
CALL MPI_BCAST(cBuf, SIZE, MPI_CHAR, 0, MPI_COMM_WORLD, ierr)
```

Here, we gain several advantages: Instead of making N (the number of tasks in MPI_COMM_WORLD) open, read requests, we are making only (1). Also, the broadcast will use a fan out method to reduce the network traffic by allowing the interconnect routers of intermediate nodes to process less data.

However, if the shared data size exceeds 100 MB, you should contact the OLCF User Assistance Center (/web/20190219213055/https://www.olcf.ornl.gov/for-users/user-assistance/) for further optimizations.

Limit the Number of Files in a Single Directory

For large-scale applications that are going to write large numbers of files using private data, it is best to implement a subdirectory structure to limit the number of files in a single directory. A suggested approach is a (2)-level directory structure with sqrt(N) directories each containing sqrt(N) files, where N is the number of tasks.

Place Small Files on a Single OST

If only one process will read/write the file and the amount of data in the file is small, `stat` performance will be improved by limiting the file to a single OST on creation; every stat operation must communicate with every OST which contains file data. You can stripe a file across a single OST via:

```
$ lfs setstripe PathName -s 1m -i -1 -c 1
```

Place Directories Containing Many Small Files on a Single OST

```
$ lfs setstripe DirPathName -s 1m -i -1 -c 1
```

This is especially effective when extracting source code distributions from a tarball:

```
$ lfs setstripe DirPathName -s 1m -i -1 -c 1
$ cd DirPathName
$ tar -x -f TarballPathName
```

All of the source files, header files, etc. span only (1) OST. When you build the code, all of the object files will use only (1) OST as well. The binary will span (1) OST as well, but that is not desirable, you can copy the binary with a new stripe count:

```
$ lfs setstripe NewBin -s 1m -i -1 -c 4
$ rm -f OldBinPath
$ mv NewBin OldBinPath
```

`stat` Files from a Single Task

If many processes need the information from stat on a single file, it is most efficient to have a single process perform the `stat` call, then broadcast the results. This can be achieved by modifying the following code (shown in C):

```
int iRank;
struct stat sB;

iRC=lstat( PathName, &sB );
```

To the following:

```
    iRC=lstat( PathName, &sB );
  }
  MPI_Bcast( &sB, size(struct stat), MPI_CHAR, 0, MPI_COMM_WORLD );
```

Similarly, change the following Fortran code:

```
  INTEGER*4 sB(13)

  CALL LSTAT(PathName, sB, ierr)
```

To the following:

```
  INTEGER iRank
  INTEGER*4 sB(13)
  INTEGER ierr

  CALL MPI_COMM_RANK(MPI_COMM_WORLD, iRank, ierr)
  IF (iRank .eq. 0) THEN
    CALL LSTAT(PathName, sB, ierr)
  ENDIF
  CALL MPI_BCAST(sB, 13, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
```

> **Note:** the Fortran `lstat` binding does not support files larger than 2 GB. Users must provide their own Fortran binding to the C `lstat` for files larger than 2 GB.

Consider Available I/O Middleware Libraries

For large scale applications that are going to share large amounts of data, one way to improve performance is to use a middleware libary such as ADIOS, HDF5, or MPI-IO.

Use Large and Stripe-aligned I/O Whenever Possible

I/O requests should be large, i.e., a full stripe width or greater. In addition, you will get better performance by making I/O requests *stripe aligned* whenever possible. If the amount of data generated or required from the file on a client is small, a group of processes should be selected to perform the

HPSS – High Performance Storage System

The High Performance Storage System (HPSS) at the OLCF provides longer-term storage for the large amounts of data created on the OLCF compute systems. The mass storage facility consists of tape and disk storage components, servers, and the HPSS software. After data is uploaded, it persists on disk for some period of time. The length of its life on disk is determined by how full the disk caches become. When data is migrated to tape, it is done so in a first-in, first-out fashion.

HPSS Hardware

HPSS has SL8500 tape libraries, each holding up to 10,000 cartridges. The libraries house a total of (24) T10K-A tape drives (500 GB cartridges, uncompressed) and (60) T10K-B tape drives (1 TB cartridges, uncompressed), (36) T10K-C, and (72) T10K-D. Each drive has a bandwidth of 250 MB/s.

HPSS Archive Accounting

Each file and directory archived in HPSS is associated with an *HPSS storage account*. This associated account is used to determine storage of a user and project.

Storage Account Types

There are (3) types of HPSS storage accounts:

Overhead Account	By default, files and directories created in User Archive areas on HPSS (i.e. `/home/userid`) will be associated with the user's *overhead* account. This account is named the same as your user ID.
Project Accounts	By default, files and directories created in Project Archive areas on HPSS (i.e. `/proj/projid`) will be associated with a *project* account. The project account is named the same as project's project ID.

Account associate new files to the legacy account; however, we do encourage users to associate files currently under the legacy account with the appropriate project account.

Determine Available Accounts

The `showproj` utility can be used from any NCCS system to determine your available accounts. For example:

```
$ showproj -s hpss
userid is a member of the following project(s) on hpss:
xxxyyy
userid
```

Viewing File/Directory's Associated Account

To view a file or directory's associated account, the following commands can be used:

```
$ hsi
:[/home/userid]: ls -UH
```

For example:

```
:[/home/userid]: ls -UH
  Mode         Links  Owner    Group   COS    Acct     Where   Size         DateTime
Entry
  /home/userid:
  drwxr-x---   3      userid   1099           userid                  512   Apr 01 2008
Backups
  -rw-r-----   1      userid   1099    6001   legacy   TAPE       4280320   Oct 24 2006
file.tar
  -rw-r-----   1      userid   1099    6007   xxxyyy   DISK    1956472248   Mar 20 2008
a.inp
```

In the above example:

- The file `file.tar` was created prior to March 15, 2008 and is therefore associated with the `legacy` account.
- The file `a.inp` is associated with the `xxxyyy` project.

Modifying File/Directory's Associated Account

A new file and directory will inherit the parent directory's account association. This is the easiest and recommended method for managing data stored against an account. That said, users are able to manually change a file or directory's associated account through the `hsi chacct` command.

> **Note:** When moving an existing file, it will not inherit the parent directory's account association automatically. You must change it manually.

The syntax to manually change a directory's associated account is `chacct [-R] newacct mydir`. Similarly, the syntax to manually change a file's associated account is `chacct newacct myfile.out`.

For example:

```
chacct xxxyyy a.out
```

will set `a.out` 's associated account to `xxxyyy`. Likewise,

```
chacct -R xxxyyy dir1
```

will set the associated account of `dir1` and all files and directories within `dir1` to `xxxyyy`.

We strongly encourage users to associate HPSS files with projects as opposed to their individual user accounts; this helps us understand project needs and usage.

Viewing Storage

The showusage utility may be used to view current storage associated with a user's overhead project and other allocated projects for which the user is a member. The utility should be executed from a NCCS system; it may not be executed from within an HSI interactive session. For example:

```
$ showusage -s hpss

HPSS Storage in GB:
Project                      Project Totals Storage    userid Storage
_____________________________|_____________________________|_____________
userid                       |       8.11                |       8.11
legacy                       |                           |      25.67
```

Quotas

Space on the HPSS is for files that are not immediately needed. Users must not store files unrelated to their NCCS projects on HPSS. They must also periodically review their files and remove unneeded ones. Both Overhead and Project Accounts have a storage quota. For information on quotas, please see the OLCF Storage Policy Summary (/web/20190219213055/https://www.olcf.ornl.gov/olcf-policy-guide/).

HPSS Best Practices

Currently HSI and HTAR are offered for archiving data into HPSS or retrieving data from the HPSS archive.

For optimal transfer performance we recommend sending file of 768 GB or larger to HPSS. The minimum file size that we recommend sending is 512 MB. HPSS will handle files between 0K and 512 MB, but write and read performance will be negatively affected. For files smaller than 512 MB we recommend bundling them with HTAR to achieve an archive file of at least 512 MB.

When retrieving data from a tar archive larger than 1 TB, we recommend that you pull only the files that you need rather than the full archive. Examples of this will be give in the htar section below.

(https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#choosing-a-stripe-pattern) section of the Lustre® Basics (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#lustre-basics) page to learn how and why choosing the right striping pattern is important.

We also recommend using our data transfer nodes (DTNs) for achieving the fastest possible transfer rates. This can be done by logging on to `dtn.ccs.ornl.gov` and initiating transfers interactively or by submitting a batch job from any OLCF resource to the DTNs as described in the HSI and HTAR Workflow section.

Using HSI

Issuing the command `hsi` will start HSI in interactive mode. Alternatively, you can use:

```
hsi [options] command(s)
```

...to execute a set of HSI commands and then return.

To list you files on the HPSS, you might use:

```
hsi ls
```

`hsi` commands are similar to `ftp` commands. For example, `hsi get` and `hsi put` are used to retrieve and store individual files, and `hsi mget` and `hsi mput` can be used to retrieve multiple files.

To send a file to HPSS, you might use:

```
hsi put a.out
```

To put a file in a pre-existing directory on hpss:

To retrieve one, you might use:

```
hsi get /proj/projectid/a.out
```

Warning: If you are using HSI to retrieve an single file larger than 1 TB please make sure that the stripe pattern that you choose is approprate for this file's size. See the "Choosing a Stripe Pattern" section of "Choosing a Stripe Pattern" (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#choosing-a-stripe-pattern) to learn how and why.

Here is a list of commonly used hsi commands.

Command	Function
cd	Change current directory
get, mget	Copy one or more HPSS-resident files to local files
cget	Conditional get – get the file only if it doesn't already exist
cp	Copy a file within HPSS
rm mdelete	Remove one or more files from HPSS
ls	List a directory
put, mput	Copy one or more local files to HPSS
cput	Conditional put – copy the file into HPSS unless it is already there
pwd	Print current directory

mkdir	Create an HPSS directory
rmdir	Delete an HPSS directory

Additional HSI Documentation

There is interactive documentation on the `hsi` command available by running:

```
hsi help
```

Additionally, documentation can be found at the Gleicher Enterprises website, including an HSI Reference Manual (https://web.archive.org/web/20190219213055/http://mgleicher.us/index.html/hsi/hsi_reference_manual_2 and the HSI man page (https://web.archive.org/web/20190219213055/http://mgleicher.us/index.html/hsi/hsi_man_page.html).

Using HTAR

The `htar` command provides an interface very similar to the traditional `tar` command found on UNIX systems. It is used as a command-line interface. The basic syntax of `htar` is:

htar -{c|K|t|x|X} -f tarfile [directories] [files]
As with the standard Unix `tar` utility the `-c`, `-x`, and `-t` options, respectively, function to create, extract, and list tar archive files. The `-K` option verifies an existing tarfile in HPSS and the `-X` option can be used to re-create the index file for an existing archive.

For example, to store all files in the directory `dir1` to a file named `allfiles.tar` on HPSS, use the command:

```
htar -cvf allfiles.tar dir1/*
```

```
htar -xvf allfiles.tar
```

`htar` will overwrite files of the same name in the target directory.

When possible, extract only the files you need from large archives.

To display the names of the files in the `project1.tar` archive file within the HPSS home directory:

```
htar -vtf project1.tar
```

To extract only one file, `executable.out`, from the `project1` directory in the Archive file called `project1.tar`:

```
htar -xm -f project1.tar project1/ executable.out
```

To extract all files from the `project1/src` directory in the archive file called `project1.tar`, and use the time of extraction as the modification time, use the following command:

```
htar -xm -f project1.tar project1/src
```

HTAR Limitations

The `htar` utility has several limitations.

Apending data

You cannot add or append files to an existing archive.

File Path Length

File path names within an `htar` archive of the form prefix/name are limited to 154 characters for the prefix and 99 characters for the file name. Link names cannot exceed 99 characters.

Size

Individual File Size Maximum	68GB, due to POSIX limit
Maximum Number of Files per Archive	1 million

For example, when attempting to HTAR a directory with one member file larger that 64GB, the following error message will appear:

```
[titan-ext1]$htar -cvf hpss_test.tar hpss_test/

INFO: File too large for htar to handle: hpss_test/75GB.dat (75161927680 bytes)
ERROR: 1 oversize member files found - please correct and retry
ERROR: [FATAL] error(s) generating filename list
HTAR: HTAR FAILED
```

Additional HTAR Documentation

Additional documentation can be found on the Gleicher Enterprises website, including the HTAR user's guide (https://web.archive.org/web/20190219213055/http://mgleicher.us/index.html/htar/htar_user_guide.html) and the HTAR man page (https://web.archive.org/web/20190219213055/http://mgleicher.us/index.html/htar/htar_man_page.html).

HSI and HTAR Workflow

Transfers with the HPSS should be launched from the external Titan login nodes, the interactive data transfer nodes (dtns), or the batch-accessible dtns.

If the file size is above 512 MB and HSI is initiated from titan-ext, or titan-batch nodes the HSI-DTN will transfer files in a further optimized and striped method.

To submit a data archival job from any OLCF resource use the -q dtn option with qsub.

```
qsub -q dtn Batch-script.pbs
```

Your allocation will not be charged time for this job.

Below is an example batch script using HTAR.

Batch-script.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/
```

See the workflow documentation for more workflow examples. (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#enabling-workflows-through-crosssystem-batch-submission)

Storage Locations

Users are provided with a User Archive directory on HPSS that is located at `/home/userid` (where `userid` is your User ID). Additionally, each project is given a Project Archive directory located at `/proj/projectid` (where `projectid` is the six-character project ID).

A Note on Bundling Data

allow HPSS to store the data more efficiently.

The HTAR command is very useful for bundling smaller files, and is often faster than using the conventional `tar` command and then transferring via HSI. HTAR has an individual file size limit of 64GB, due to the POSIX tar specification. However, HTAR can be used to store and retrieve directories that are in total large than 64GB, provided that they do not contain any individual files large than 64GB.

When retrieving a large number of files, if HSI knows there are many files needed, it can bundle retrieves. This method allows HPSS to gather needed files on a single tape and perform fewer mount/seeks/rewind/unmounts. For example:

The following will create a list of files and pass the list to HPSS to retrieve. Note that this method does not preserve directory structure and is better used when directory structure is not needed:

```
echo "get << EOFMARKER" > dir0.lst
hsi -q find testdir -type f >>& dir0.lst
echo "EOFMARKER" >> dir0.lst
hsi "out dir0.out ; in dir0.lst"
```

Classes of Service and Data Redundancy

The HPSS has several Classes of Service (COS) to ensure that files are efficiently stored based on their size. The COS is set automatically based on the size of the file that is being stored.

COS ID	Name based on filesize	# Tapes
11	NCCS 0MB<16MB	3 copies
12	NCCS 16MB<8GB	RAIT 2+1
13	NCCS 8GB<1TB	RAIT 4+1
14	NCCS >1TB	RAIT 4+1

For files less than 16 MB in size, three copies are written to tape. For files 16MB or greater in size, HPSS supports a Redundant Array of Independent Tapes (RAIT) so there is no need to use multiple copies to ensure file safety in the event of tape failure.

Neither multiple-copies nor RAIT will protect your data if you accidentally delete it. Therefore avoid `hsi rm */*`.

Using File Globbing Wildcard Characters with HSI

HSI has the option to turn file globbing on and off.
If you get this message:

```
O:[/home/user]: ls -la file*
*** hpss_Lstat: No such file or directory [-2: HPSS_ENOENT]
  /home/user/file*
```

...then you'll need to turn on HSI file globbing with the `glob` command:

```
O:[/home/user]: glob
filename globbing turned on

O:[/home/user]: ls -la file*
-rw-r--r--   1 user  users     6164480 Jan 14 10:36 file.tar
-rw-r--r--   1 user  users     6164480 Jan  6 11:08 file1.tar.gz
-rw-r--r--   1 user  users     6164480 Jan  6 11:08 file2.tar
-rw-r--r--   1 user  users     6164480 Jan  6 11:09 file3.tar
-rw-r--r--   1 user  users     6164480 Jan  6 11:09 file4.tar
-rw-r--r--   1 user  users     6164480 Jan  6 11:09 file5.tar
```

Employing Data Transfer Nodes

The OLCF provides nodes dedicated to data transfer that are available via dtn.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

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 Policy Summary section (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#data-management-policy-summary).

Batch DTN Access

Batch data transfer nodes can be accessed through the Torque/MOAB queuing system on the dtn.ccs.ornl.gov interactive node. The DTN batch nodes are also accessible from the Titan, Eos, and Rhea batch systems through 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.

```
$ 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=atlas1 # or atlas2

  # 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

$
```

```
$ 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:

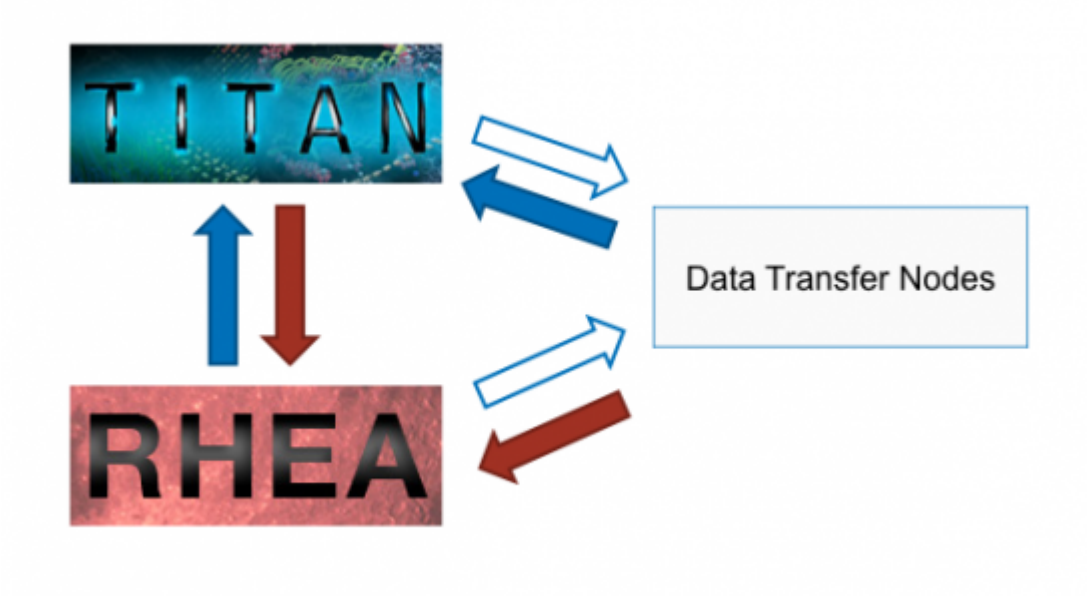
- Batch jobs submitted to the `dtn` partition will be executed on a DTN that is accessible exclusively via batch submissions. These batch-accessible DTNs have identical configurations to those DTNs that are accessible interactively; the only difference between the two is accessibility.
- The DTNs are 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.

Scheduled DTN Queue

- The walltime limit for jobs submitted to the `dtn` partition is 24 hours.
- Users may request a maximum of 4 nodes per batch job.
- There is a 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 when appropriate.
- The queue allows each user a maximum of 6 running jobs.

Submission

The OLCF 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.



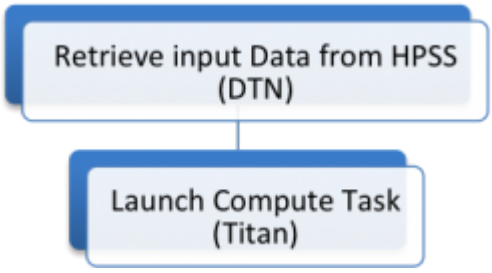
Cross-Submission allows jobs on one OLCF resource to submit new jobs to other OLCF resources.

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:

Eos	`qsub -q eos visualization.pbs`
Titan	`qsub -q titan compute.pbs`
Data Transfer Nodes (DTNs)	`qsub -q dtn retrieve_data.pbs`

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
```

```
#PBS -l walltime=2:00:00
#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 dtn 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/
```

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.

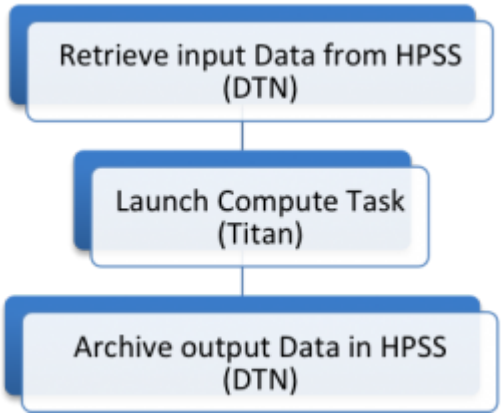
Initializing the Workflow

We can initialize this workflow in one of two ways:

- Log into `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 2: Data Staging, Compute, and Archival

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 dtn 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/
```

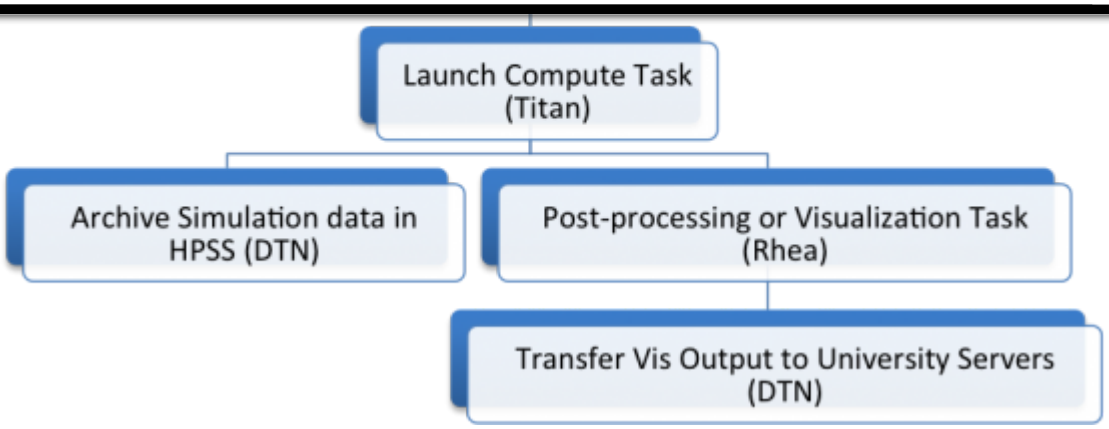
Initializing the Workflow

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 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 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	`qstat -a @rhea-batch`	`showq --host=rhea-batch`
Eos	`qstat -a @eos-batch`	`showq --host=eos-batch`
Titan	`qstat -a @titan-batch`	`showq --host=titan-batch`
Data Transfer Nodes (DTNs)	`qstat -a @dtn-batch`	`showq --host=dtn-batch`

Deleting Remote Jobs

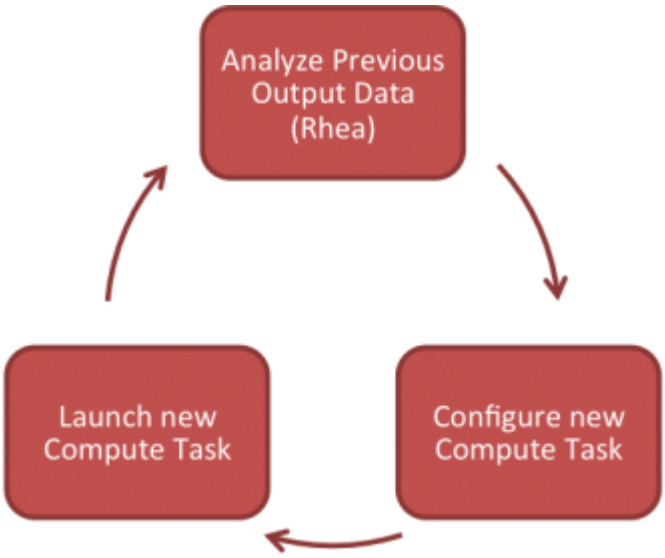
In order to delete a job (say, job number 18688) from a remote queue, you can do the following

Host	Remote qdel
Rhea	`qdel 18688@rhea-batch`
Eos	`qdel 18688@eos-batch`
Titan	`qdel 18688@titan-batch`

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.

Local Transfers

archival storage and from one scratch area to another.

The following utilities can be used to transfer data between partitions of the filesystem.

Utility	Amount of Data to Transfer	Where to run utility
cp	< 500 GB	command line / batch script

	to Transfer	run utility
bbcp (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#bbcp)	< 500 GB	DTN command line or batch script

	to Transfer	run utility
fcp (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#fcp)	> 500 GB	batch script

For large transfers, transfers to the High Performance Storage System, and when running fcp or bbcp, the DTNs should be used to prevent overloading the compute system's login nodes.

> To help reduce load on the compute systems' login resources, please use the DTNs when using fcp or bbcp, or transferring more than ~500GB.

Since all areas of the shared luster filesystem are periodically swept, moving data from lustre to archival storage on High performance storage system is a necessary part of most users' work flow. The commands `hsi` and `htar` provide users with easy-to-use interfaces to their User Archive and Project Archive spaces on the OLCF's HPSS-based archival storage system.

guides/titan/titan-user-guide/#hpss-best-practices) section contains examples using HSI and HTAR, as well as sample workflows.

fcp

fcp is a program designed to do large-scale parallel data transfer from a source directory to a destination directory across locally mounted file systems. It is not for wide area data transfers such as ftp, bbcp, or globus-ftp. In that sense, it is closer to cp. One crucial difference from regular cp, is that fcp requires the source and destination to be directories. fcp will fail if these conditions are not met. In the most general case, fcp works in two stages: first it analyzes the workload by walking the tree in parallel; and then it parallelizes the data copy operation.

At the OLCF, fcp is provided as a modulefile and it is available on the data transfer nodes (DTNs) and the analysis cluster (Rhea). To use fcp:

1. Load the pcircle modulefile:

```
$ module load pcircle
```

2. Use mpirun to copy a directory:

```
$ mpirun -np 8 fcp src_dir dest_dir
```

fcp options

In addition, fcp supports the following options:

```
-f, --force: Overwrite the destination directory. The default is off.

--verify: Perform checksum-based verification after the copy.

-s, --signature: Generate a single sha1 signature for the entire dataset.
This option also implies --verify for post-copy verification.

--chunksize sz: fcp will break up large files into pieces to increase
parallelism. By default, fcp adaptively sets the chunk size based on the
overall size of the workload. Use this option to specify a particular
chunk size in KB, MB. For example, --chunksize 128MB.

--reduce-interval: Controls progress report frequency. The default is 10
seconds.
```

Using fcp inside a batch job

The data transfer nodes (DTNs) can be used to submit an fcp job to the batch queue:

1. Connect to the DTNs:

```
ssh <username>@dtn.ccs.ornl.gov
```

2. Prepare a PBS submission script:

```
#!/bin/bash -l
#PBS -l nodes=2
#PBS -A <projectID>
#PBS -l walltime=00:30:00
#PBS -N fcp_job
#PBS -j oe

cd $PBS_O_WORKDIR
module load pcircle
module list

mpirun -n 16 --npernode 8 fcp ./src_dir ./dest_dir
```

The `--pernode` is needed to distribute the MPI processes across physical nodes. Ommitting this option would place 16 MPI processes on the same node, which in this example is not the desired behavior, as it would reduce the amount of memory available to each process.

network, the destination file system, and the number of processes and nodes involved in the transfer. Using more processes per node does not necessarily result in better performance due to an increase in the number of metadata operations as well as additional contention generated from a larger number of processes. A rule of thumb is to match or halve the number of physical cores per transfer node.

Both post copy verification (–verify) and dataset signature (–signature) options have performance implications. When turned on, fcp calculates the checksums of each chunk/file for both source and destination, in addition to reading back from destination. This increases both the amount of bookkeeping and memory usage. Therefore, for large scale data transfers, a large memory node is recommended.

Author and Code

Feiyi Wang (fwang2@ornl.gov), ORNL Technology Integration group.
Source code for PCircle can be found on the OLCF GitHub page (https://web.archive.org/web/20190219213055/https://github.com/olcf/pcircle).

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 by default, secure with configuration	Insecure by default, secure with configuration	Secure	Insecure & unsuited for sensitive projects
Authentication	Passcode	Passcode	Passcode	Passcode
Transfer speed	Fast	Fast	Slow	Fast
Required Infrastructure	GridFTP server at remote site	GridFTP server at remote site	Comes standard with SSH install	BBCP installed at remote site

Globus 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.
Globus is a kind of GridFTP that provides a web user-interface for initiating, managing, and monitoring GridFTP transfers between endpoints. An endpoint is the logical address of a resource or filesystem attached to a Globus Connect GridFTP server. Many institutions host their own shared Globus Connect Servers and endpoints. However, it is possible to turn any non-commercial private resource into an endpoint using the Globus Connect Personal client. Globus can also be scripted.

The Globus GridFTP suite provides the following interfaces for managing data transfer:

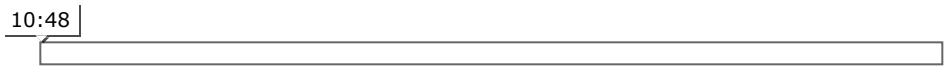
- Globus Online Web-UI
- Globus Online command line interface (https://web.archive.org/web/20190219213055/https://docs.globus.org/cli/examples/)
- `globus_url_copy` shell command

of the authentication method chosen. GridFTP is currently available on each OLCF system and can be added to your environment using the globus module:

```
$ module load globus
```

If your site does not already have GridFTP available, it can be downloaded from Globus. Download and installation information can be found on the Globus Toolkit Documentation (https://web.archive.org/web/20190219213055/http://globus.org/toolkit/docs/) site.

Transferring Data with Globus Online

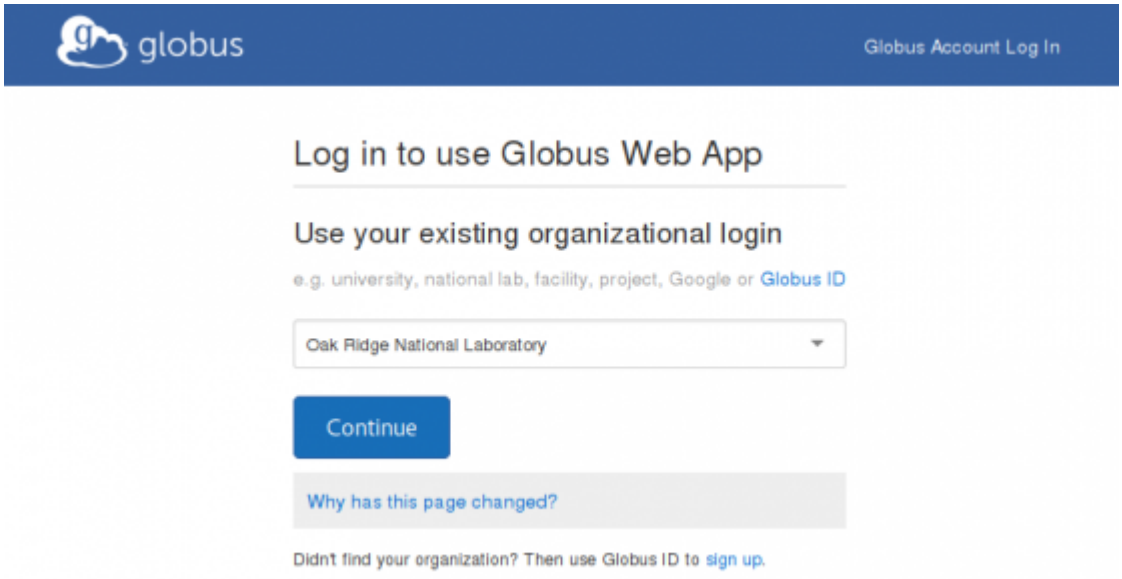


In the browser of your choice, visit https://www.globus.org (https://web.archive.org/web/20190219213055/https://www.globus.org. /).

follow the “Globus ID” link to login.

> Only users who have an ORNL UCAMS institutional ID may choose “Oak Ridge National Lab” from the dropdown menu.

See the Globus accounts FAQ (https://web.archive.org/web/20190219213055/https://docs.globus.org/faq/security/#how_to_use_a_ for more information.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2015/10/new_login.png)

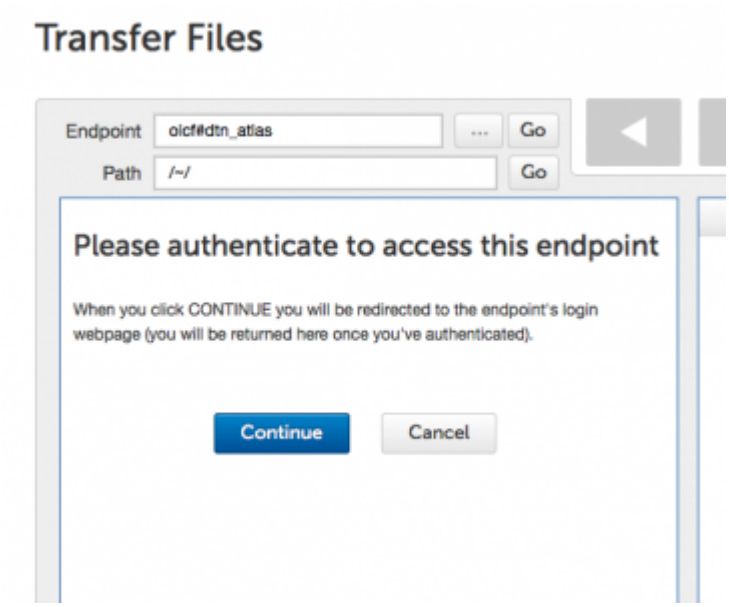
Login to the Globus webapp using an existing Globus ID, linked institutional ID, or new Globus account.

2 Once logged in to the Globus webapp, select “Manage Data” and then “Transfer Files” from blue menu bar.

(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2015/10/manage_data.png)

Choose Transfer Tiles from the navigation menu.

3 Enter an OLCF endpoint into one of the two Endpoint fields. Using the endpoint selection dialog that appears, enter `OLCF DTN` to choose the OLCF as an endpoint.

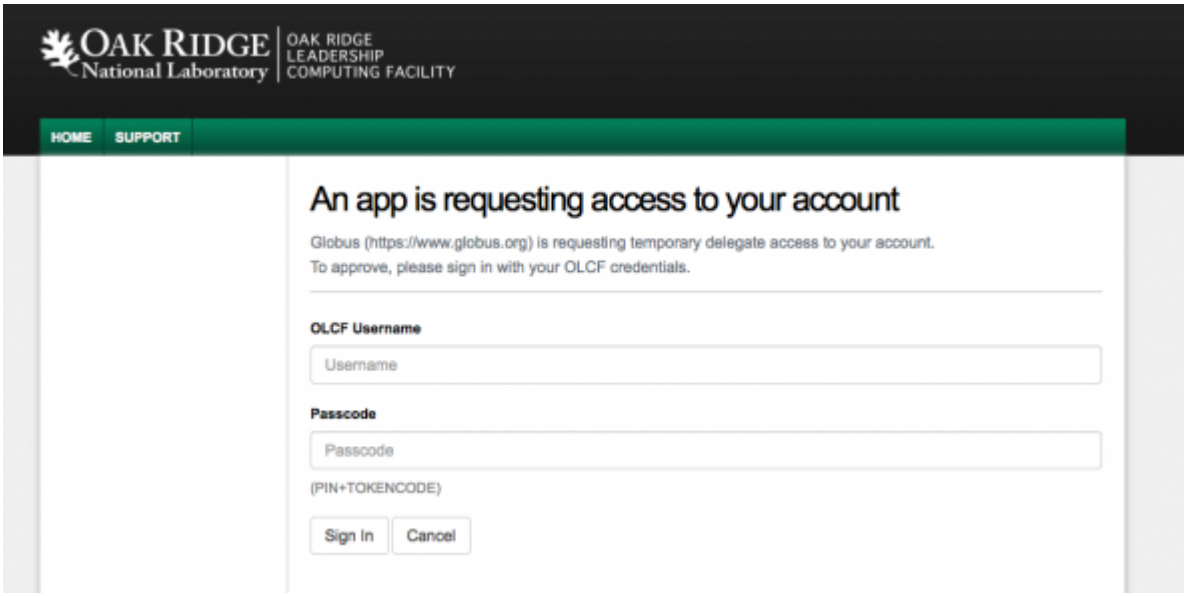


(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2015/10/Screen-Shot-2015-10-30-at-10.51.13.png)

Select an OLCF endpoint in one of the two available fields.

> Workflows established prior to February 2016 may have used the now-discontinued `olcf#dtn` endpoint. This endpoint should no longer be used. Questions about migrating legacy workflows can be directed to help@olcf.ornl.gov

4 Globus must request permission from you via the OLCF to access your files. Press the Continue button. You will be redirected to our authentication page at "myproxy*.ccs.ornl.gov". Enter your OLCF username in the "username" box and your OLCF SecurID passcode in the "Passcode" box. Upon success, you will be returned to the Globus web interface.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2015/10/Screen-Shot-2015-10-30-at-11.14.42.png)

Activating an endpoint using only your OLCF credentials requires a browser to authenticate the OAuth request from Globus.

> The endpoint lifetime is 72 hours. If the endpoint authentication expires before the transfer is complete, the transfer will automatically resume the next time you reactivate the endpoint.

5 Enter the path to your data (for example /tmp/work/username) in the "path: window. Soon you should see a list of the files in that directory appear in the window below.

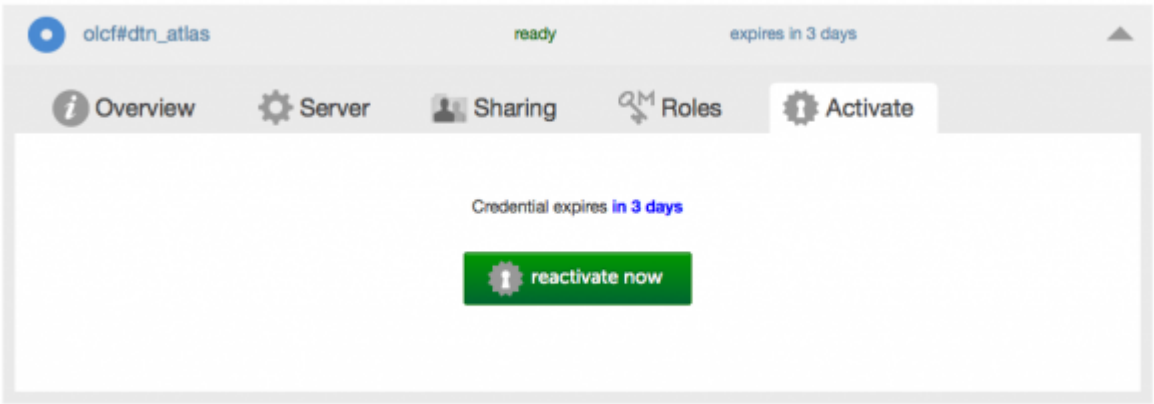
7 Select the files you want to transfer by clicking on them. Use the arrows in the middle of the page to do the transfer.

8 Globus will give you a message at the top of the page about the status of your transfer and send you an email when your transfer is complete.

Reactivating an Expired Endpoint

If the endpoint or proxy expires before the transfer is complete, the transfer will automatically resume the next time you activate the endpoint.

To reactivate an expired endpoint, choose “Manage Endpoints” from the Globus web interface. Select the OLCF endpoint you wish to reactivate and choose the “Activate” tab. Press the “Reactivate Now” button and enter your OLCF credentials to approve the request by Globus to access your account.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2015/10/Screen-Shot-2015-10-30-at-11.17.17.png)

Reactivate an endpoint under the Manage Endpoints section.

Globus Online Command Line Interface

Globus Online also provides a scriptable command-line interface available via SSH at `cli.globusonline.org` using your Globus account credentials. Complete information about `cli.globusonline.org` can be found in the official Globus documentation (https://web.archive.org/web/20190219213055/http://dev.globus.org/cli/using-the-cli/).

To use the CLI interface, you must first generate an SSH public-private key pair on the host from which you will use the interface. From a terminal, call

```
$ ssh-keygen -t rsa -b 4096 -C "Globus key on $HOST" -f $HOME/.ssh/id_rsa.globus
```

> It is highly recommended that all your SSH keys are protected by passphrases. Passphrase-protected keys can be used in conjunction with an SSH-agent for convenience.

Compromised passphrase-less keys linked to Globus will allow read-write access to all of your activated endpoints.
Add the public key to your Globus ID's list of authorized keys. From the web-UI, click on the Account menu link, then choose "manage SSH and X.509 keys", then "Add a New Key". Give the key any alias, choose the SSH Public Key Type, paste the full contents of `$HOME/.ssh/id_rsa.globus.pub` into the body field and click "Add Key".

To use the interface, start an SSH session as your globus ID username with

```
$ ssh -i $HOME/.ssh/id_rsa.globus ${GLOBUS_UNAME}@cli.globusonline.org
```

This command will place you into an interactive console from which globus transfer management commands can be issued. Calling `help` will list all of the available commands. Full documentation for each command is available through `man $COMMAND`.

uses the Globus Online Tutorial Endpoints *go#ep1* and *go#ep2* , which are available to all Globus users for practice, to demonstrate basic operations.

```
# using the ssh://cli.globusonline.org interface.
#
#================================================================================

# Edit these as needed for individual use
PUBKEY="$HOME/.ssh/id_rsa.globus"
GLOBUS_UNAME="FIXME"

# Function to simplify remote Globus command invocations.
gocli() {
  ssh -i ${PUBKEY} ${GLOBUS_UNAME}@cli.globusonline.org "$@"
}

# Print available commands. Man pages can be read by starting an interactive
# session over ssh using `ssh ${GLOBUS_UNAME}@cli.globusonline.org`
gocli help

# Activate the endpoints.
# endpoint-activate returns 0 if active or successfully activated.
# Some endpoints may be interactively activated, but not the OLCF's.
# It is a good practice to exit the script on activation problems when the
# script is run non-interactively.
# TODO - Add a trap or timeout if this script is run non-interactively
#        against endpoints that can be interactively activated.
gocli endpoint-activate "go#ep1"
[ "$?" = 1 ] && exit 1

gocli endpoint-activate "go#ep2"
[ "$?" = 1 ] && exit 1

# Make destination dirs - this is not strictly necessary, just showing off
# `mkdir`.
if ! $(gocli ls "go#ep2/~/simple_demo" > /dev/null 2>&1); then
  gocli mkdir "go#ep2/~/simple_demo"
fi

# List the SRC and DST folder contents.
gocli ls -l "go#ep1/share/godata"
gocli ls -l "go#ep2/~/simple_demo"

# Bulk file transfer:
# Constuct array of INPUTLINE(s) from ls output on src dir:
DATA_FILES=( $(gocli ls "go#ep1/share/godata/*.txt") )
{
for i in "${!DATA_FILES[@]}"; do
  f="${DATA_FILES[$i]}"
```

```
# Recursive transfer:
gocli transfer -s 3 --label="scripted_recursive_xfer_demo" -- \
  "go#ep1/share/godata/" \
  "go#ep2/~/simple_demo/recursive/" -r

# Print the status of the last two transfers:
# See `gocli man status` for format options to make parsing easier.
gocli status -a -l 2
```

Globus GridFTP with SSH Authentication

No setup is required if you will be using SSH for authentication. However, to use this for remote transfers, the remote facility must also accept SSH for authentication.

Transferring Data

Files are transferred using the `globus-url-copy` command. The arguments to that command will differ based on your authentication method.

To use `globus-url-copy` with SSH authentication load the globus module:

```
$ module load globus
```

Then run the `globus-url-copy` command:

For example, while on an OLCF resource, you can transfer `file1` in your OLCF User Work area to `file2` on a remote system:

```
$ globus-url-copy -tcp-bs 12M -bs 12M -p 4 -v -vb file:/lustre/atlas/scratch/$USER/file1
sshftp://user@remote.system/remote/dir/file2
```

From the OLCF, transfer `file1` on a remote system to `file2` in your User Work area:

From remote system, transfer `file1` on a remote system to `file2` in your User Work area:

```
 $ globus-url-copy -tcp-bs 12M -bs 12M -p 4 -v -vb file:/remote/dir/file1 sshftp://userid
@dtn.ccs.ornl.gov/lustre/atlas/scratch/$USER/file2
```

SSH Keys

SSH keys can not be used to grant passwordless access to OLCF resources. However, SSH keys can be created for OLCF systems to use for access to remote resources that support ssh keys.

To create an SSH key pair for dtn.ccs.ornl.gov:

Log in to dtn.ccs.ornl.gov and go to your .ssh directory or create a .ssh directory if you do not have one.

```
$ssh-keygen -t dsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /ccs/home/$USER/.ssh/id_dsa.
Your public key has been saved in /ccs/home/$USER/.ssh/id_dsa.pub.
```

This will generate a ssh key pair consisting of id_dsa.pub and id_dsa. If you choose not to enter a passphrase, anyone who gains access to your private key file will then be able to assume your identity.

To cache the private key and passphrase so that you do not need to enter it for every ssh operation in your session:

```
$ ssh-agent
$ ssh-add
Identity added: /ccs/home/$USER/.ssh/id_rsa (/ccs/home/$USER/.ssh/id_rsa)
Enter passphrase for /ccs/home/$USER/.ssh/id_dsa:
Identity added: /ccs/home/$USER/.ssh/id_dsa (/ccs/home/$USER/.ssh/id_dsa)
```

Copy your id_dsa.pub to the remote resource's .ssh directory and copy the contents into a file called "authorized_keys".

SFTP & 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.

Both SCP and SFTP are available on all NCCS systems and should be a part of each user's environment. For example, on a UNIX-based system, to transfer the file `oldfile` from your local system to your `$HOME` directory on OLCF systems as `newfile`, you would use one of the following commands:

SFTP

```
sftp userid@dtn.ccs.ornl.gov
sftp> put oldfile newfile
sftp> bye
```

SCP

```
scp ./oldfile userid@dtn.ccs.ornl.gov:~/newfile
```

...where `userid` is your given NCCS username.

Standard file transfer protocol (FTP) and remote copy (RCP) should not be used to transfer files to the NCCS high-performance computing (HPC) systems due to security concerns.

SCP works with NCCS systems only if your per-process initialization files produce no output. The means that files such as `.cshrc`, `.kshrc`, `.profile`, etc. must not issue any commands that write to standard output. If you would like for this file to write to standard output for interactive sessions, you must edit the file so that it does so only for interactive sessions.

For *sh-type* shells such as `bash` and `ksh` use the following template:

```
    fi
```

For *c-shell-type* shells such as `csh` and `tcsh` use:

```
( /usr/bin/tty ) > /dev/null
  if ( $status == 0 ) then
    /usr/bin/echo "interactive stuff goes here"
  endif
```

BBCP

Before you can use the BBCP utility, it must be installed on both the local and remote systems. It is currently available on each OLCF system and should be a part of each user's default environment. Several pre-compiled binaries (https://web.archive.org/web/20190219213055/http://www.slac.stanford.edu/~abh/bbcp/bin/) as well as the source can be downloaded from the Stanford Linear Accelerator Center (SLAC) BBCP (https://web.archive.org/web/20190219213055/http://www.slac.stanford.edu/~abh/bbcp/) page.

Installation from Source Tips

- Refer to your operating system's documentation to satisfy missing dependencies or problems in your build environment.
- Clone the BBCP source code git repository from the Stanford Linear Accelerator Center (SLAC) BBCP (https://web.archive.org/web/20190219213055/http://www.slac.stanford.edu/~abh/bbcp/) page.

```
$ git clone http://www.slac.stanford.edu/~abh/bbcp/bbcp.git
```

- From within the decompressed BBCP directory, run `make`. This should build the BBCP executable into the created bin directory. The build has been tested on Linux-based systems and should build with few or no modifications. If you system's `uname` command does not return Linux or Darwin, you may need to modify the `Makefile`.

```
$ make
```

Common variable modifications

- In `MakeSname`, the `test` command is hard coded to `/usr/bin/test`. If this is not the location of `test` on your system, you can change the following line to the correct path (`which test` should return the path to test on your system):

```
if /usr/bin/test -${1} $2; then
```

- If the `uname` command is not in `/bin` on your system, change the `uname` variable in the `MakeSname` file. You will also need to change the following line in the file `Makefile`:

```
@cd src;$(MAKE) make`/bin/uname` OSVER=`../MakeSname`
```

- If the `libz.a` library is not located at `/usr/local/lib/libz.a` on your system, change the `libzMakefile` file.
- The file `Makefile` contains compiler and compiler flag options for the BBCP build. You can change the compilers and flags by modifying variables in this file. For example, to change the compilers used on a Linux system, modify the variables `LNXCC` and `LNXcc`.

Usage

To transfer the local file `/local/path/largefile.tar` to the remote system `remotesystem` as `/remote/path/largefile.tar`, use the following:

```
bbcp -P 2 -V -w 8m -s 16 /local/path/largefile.tar remotesystem:/remote/path/largefile.ta
r
```

where

`-w 8m` sets the size of the disk input/output (I/O) buffers.

`-s 16` sets the number of parallel network streams to 16.

BBCP assumes the remote system's non-interactive environment contains the path to the BBCP utility. This can be determined with the following command:

```
ssh remotesystem which bbcp
```

If this is not the case, the `-T` BBCP option can be used to specify how to start BBCP on the remote system. For example, you could use the following:

```
bbcp -P 2 -V -w 8m -s 16 -T 'ssh -x -a -oFallBackToRsh=no %I -l %U %H /remote/path/to/bbcp' /local/path/largefile.tar remotesystem:/remote/path/largefile.tar
```

Often, during large transfers the connection between the transferring systems is lost. The `-a` option gives BBCP the ability to pick up where it left off. For example, you could use the following:

```
bbcp -k -a /remotesystem/homedir/.bbcp/ -P 2 -V -w 8m -s 16 /local/path/largefile.tar remotesystem:/remote/path/largefile.tar
```

To transfer an entire directory tree, use the following:

```
bbcp -r -P 2 -V -w 8m -s 16 /local/path/* remotesystem:/remote/path
```

We strongly recommend that you use the Data Transfer Nodes (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#employing-data-transfer-nodes) when transferring files to and from the OLCF. If you are, however, connecting directly to systems such as the Cray XK, it is necessary to specify

example, you could use the following:

```
bbcp -r -P 2 -V -w 8m -s 16 /local/path/* titan-login3.ccs.ornl.gov:/remote/path
```

You may encounter an error similar to the following:

```
bbcp: Accept timed out on port 5031
bbcp: Unable to allocate more than 0 of 8 data streams.
Killed by signal 15.
```

If this happens, add the `-z` option to your bbcp command. This tells bbcp to use the "reverse connection protocol" and can be helpful when a transfer is being blocked by a firewall.

Further Reading

More information on BBCP can be found by typing "bbcp -h" on OLCF systems as well as on the Stanford Linear Accelerator Center (SLAC) BBCP (https://web.archive.org/web/20190219213055/http://www.slac.stanford.edu/~abh/bbcp/) page.

Data Management Policy Summary

Users must agree to the full Data Management Policy (/web/20190219213055/https://www.olcf.ornl.gov/olcf-policy-guide) 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

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	Ret
Project Home	/ccs/proj/[projid]	NFS	770	50 GB	Yes	No	90
Member Work	$MEMBERWORK/[projid]	Lustre®	700 [2]	10 TB	**No**	**14 days**	14
Project Work	$PROJWORK/[projid]	Lustre®	770	100 TB	**No**	**90 days**	90
World Work	$WORLDWORK/[projid]	Lustre®	775	10 TB	**No**	**90 days**	90
Project Archive	/proj/[projid]	HPSS	770	100 TB [3]	**No**	No	90

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.

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.

Software

> **Note:** HPC container runtimes are unsupported on Titan.

For a full list of software available at the OLCF, please see the Software (/web/20190219213055/https://www.olcf.ornl.gov/for-users/software/) section.

OLCF systems provide hundreds of software packages and scientific libraries pre-installed at the system-level for users to take advantage of. To facilitate this, environment management tools are employed to handle necessary changes to the shell dynamically. The sections below provide information about using the management tools at the OLCF.

Default Shell

A user's default shell is selected when completing the User Account Request form. The chosen shell is set across all OLCF resources. Currently, supported shells include:

- bash
- tsch
- csh
- ksh

If you would like to have your default shell changed, please contact the OLCF User Assistance Center (/web/20190219213055/https://www.olcf.ornl.gov/for-users/user-assistance/) at help@olcf.ornl.gov (https://web.archive.org/web/20190219213055/mailto:help@olcf.ornl.gov).

Using Modules

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

Modules Overview

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
`module list`	Lists modules currently loaded in a user's environment
`module avail`	Lists all available modules on a system in condensed format
`module avail -l`	Lists all available modules on a system in long format
`module display`	Shows environment changes that will be made by loading a given module
`module load`	Loads a module
`module unload`	Unloads a module
`module help`	Shows help for a module
`module swap`	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(defaul
t)
atp/1.4.1                         netcdf-hdf5parallel/4.1.3          trilinos/10.6.4.0(d
efault)
...
```

To search for availability of a module by name:

```
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
```

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

Installed Software

The OLCF provides hundreds of pre-installed software packages and scientific libraries for your use, in addition to taking software installation requests (/web/20190219213055/https://www.olcf.ornl.gov/support/software/software-request/). See the Software (/web/20190219213055/https://www.olcf.ornl.gov/for-users/software/) page for complete details on existing installs.

Compiling

> **Note:** For GPU specific compilation details please see the Accelerated Computing Guide (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/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 encourged.
- 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:

- GCC (https://web.archive.org/web/20190219213055/http://gcc.gnu.org/), the GNU Compiler Collection
- CCE, the Cray Compiling Environment
- Intel (https://web.archive.org/web/20190219213055/http://software.intel.com/en-us/intel-composer-xe), Intel Composer XE

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.

Cray Compiler Wrappers

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:

Wrapper	Purpose
cc	To use the C compiler
CC	To use the C++ compiler
ftn	To use the FORTRAN compiler

The `-craype-verbose` option can be used to view the compile line built by the compiler wrapper:

Compiling and Node Types

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 (Cross Compilation)

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:

```
titan-ext$ cc code.c
titan-ext$ CC code.cc
titan-ext$ ftn code.f90
```

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.

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 a Lustre scratch area (`$MEMBERWORK` or `$PROJWORK`) or the NFS `/ccs/proj` area and then update your `LD_LIBRARY_PATH` environment variable to include this directory. Due to the meta-data overhead, /ccs/proj is the suggested area for shared objects and python modules. For example, the following command will append your project's NFS home area to the `LD_LIBRARY_PATH` in bash:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/ccs/proj/[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 or /ccs/proj 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.

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. Add the `-target-cpu=mc8` flag to your `cc`, `CC`, or `ftn` command
2. Use the `craype-mc8` module: `module swap craype-interlagos craype-mc8`
3. Call the underlying compilers directly (e.g. `pgf90`, `ifort`, `gcc`)
4. Use the `craype-network-none` module to remove the network and MPI libraries:
   `module swap craype-network-gemini craype-network-none`

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.

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.

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 (default)
- PrgEnv-gnu

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
```

Note: 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.

Compiling Threaded Codes

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, -qopenmp

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

Running with OpenMP and PrgEnv-intel

An extra thread created by the Intel OpenMP runtime interacts with the CLE thread binding mechanism and causes poor performance. To work around this issue, CPU-binding should be turned off. This is only an issue for OpenMP with the Intel programming environment.

We refer to the number of MPI tasks or processing elements per socket as *npes*; this is controlled by the `-n` option to `aprun`. In the following examples, replace `depth` with the number of threads per MPI task, and `npes` with the number of MPI tasks (processing elements) per socket that you plan to use.

For the case of running when `depth` divides evenly into the number of processing elements on a socket (npes),

```
$ export OMP_NUM_THREADS="<=depth"
$ aprun -n npes -d "depth" -cc numa_node a.out
```

For the case of running when `depth` does not divide evenly into the number of processing elements on a socket (npes),

```
$ export OMP_NUM_THREADS="<=depth"
$ aprun -n npes -d “depth” -cc none a.out
```

SHMEM

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

```
$ module load cray-shmem
```

Accelerator Compiler Directives

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

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 OpenACC with 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 OpenACC with Fortran

PGI Compiler

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

Cray Compiler

CapsMC Compiler

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

OpenACC Tutorials and Resources

The OpenACC specification provides the basis for all OpenACC implementations and is available OpenACC specification (https://web.archive.org/web/20190219213055/http://www.openacc.org/sites/default/files/OpenACC.1.0_
In addition the implementation specific documentation may be of use. PGI has a site dedicated to collecting OpenACC resources (https://web.archive.org/web/20190219213055/http://www.pgroup.com/resources/accel.htm).
Chapter 5 of the Cray C and C++ Reference Manual (https://web.archive.org/web/20190219213055/http://docs.cray.com/books/S-2179-81/S-2179-81.pdf) provides details on Crays implementation. CapsMC has provided an OpenACC Reference Manual (https://web.archive.org/web/20190219213055/http://www.olcf.ornl.gov/wp-content/uploads/2012/10/HMPPOpenACC-3.2_ReferenceManual.pdf).

The OLCF provides Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/openacc-vector-addition/) and Game of Life (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/tutorials/openacc-game-of-life/) example codes demonstrating the OpenACC accelerator directives.

PGI Accelerator

> **Note:** This section covers environments and compiler flags, not PGI's OpenACC implementation. For implementation details, see PGI Accelerator Tutorials and Resources. (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-

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.

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 **T**arget **A**ccelerator flag is used:

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

PGI Accelerator Tutorials and Resources

PGI provides a useful web portal for Accelerator resources (https://web.archive.org/web/20190219213055/http://www.pgroup.com/resources/accel.htm). 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 (https://web.archive.org/web/20190219213055/http://www.pgroup.com/userforum//viewforum.php?f=12) where PGI staff regularly answer questions.

The OLCF provides Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/pgi-accelerator-vector-addition/) and Game of Life (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/pgi-accelerator-game-of-life/) example

CapsMC

> **Note:** This section covers environments and compiler flags, not Caps OpenACC implementation. For implementation details, see CapsMC Tutorials and Resources (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#capsmc-tutorials-and-resources).

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.

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
```

CapsMC Tutorials and Resources

CAPS provides several documents and code snippets to get you started with HMPP Workbench. It is recomended to start off with the HMPP directives reference manual (https://web.archive.org/web/20190219213055/http://www.olcf.ornl.gov/wp-content/uploads/2012/02/HMPPWorkbench-3.0_HMPP_Directives_ReferenceManual.pdf) and the

content/uploads/2012/02/HMPPWorkbench-3.0_HMPPCG_Directives_ReferenceManual.pdf).

The OLCF provides Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/hmpp-vector-addition/) and Game of Life (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/hmpp-game-of-life/) example codes demonstrating the HMPP accelerator directives.

GPU Languages/Frameworks

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.

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**.

```
$ nvcc source.cu
```

For a full usage walkthrough please see the supplied tutorials.

NVIDIA CUDA C Tutorials and Resources

NVIDIA provides a comprehensive web portal for CUDA developer resources here (https://web.archive.org/web/20190219213055/http://developer.nvidia.com/object/gpucomputing.html). The developer documentation center (https://web.archive.org/web/20190219213055/http://developer.nvidia.com/nvidia-gpu-computing-documentation) 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 (https://web.archive.org/web/20190219213055/https://devtalk.nvidia.com/).

The OLCF provides both a Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/cuda-vector-addition/) and Game of Life (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/cuda-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.

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

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.

PGI CUDA Fortran Tutorials and Resources

PGI provides a comprehensive web portal for CUDA Fortran resources here (https://web.archive.org/web/20190219213055/http://www.pgroup.com/resources/cudafortran.htm). The portal links to the PGI Fortran & C Accelerator Programming Model (https://web.archive.org/web/20190219213055/http://www.pgroup.com/doc/pgicudaforug.pdf) 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 (https://web.archive.org/web/20190219213055/http://www.pgroup.com/userforum//viewforum.php?f=12) where PGI staff regularly answer questions.

The OLCF provides both a Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/cuda-fortran-vector-addition/) and Game of Life (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/cuda-fortran-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.

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

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

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

OpenCL Tutorials and Resources

Khronos provides a web portal for OpenCL (https://web.archive.org/web/20190219213055/http://www.khronos.org/opencl/). From here you can view the specification, browse the reference pages (https://web.archive.org/web/20190219213055/http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xht and get individual level help from the OpenCL forums (https://web.archive.org/web/20190219213055/https://forums.khronos.org/forumdisplay.php/87-OpenCL). A developers page (https://web.archive.org/web/20190219213055/http://www.khronos.org/developers/resources/opencl/#ti 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 (https://web.archive.org/web/20190219213055/http://developer.amd.com/zones/openclzone/pages/defaul and NVIDIA (https://web.archive.org/web/20190219213055/http://developer.nvidia.com/opencl) OpenCL developer zones.

The OLCF provides both a Vector Addition (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/opencl-vector-addition/) and Game of Life (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/opencl-game-of-life/) example code

Running Jobs on Titan

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, timely and 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.

Login vs. Service vs. Compute 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.

Filesystems Available to Compute Nodes

The compute nodes do not mount all filesystems available from the login and service nodes. The Lustre® areas (`$MEMBERWORK` and `$PROJWORK`) as well as the /ccs/proj areas are available to compute nodes on OLCF Cray systems. User home directories are not mounted on compute nodes.

> **Warning:** Home directories, `/ccs/home/$USER` are not available from the compute nodes.
>
> As a result, job executable binaries and job input files must reside within a Lustre or /ccs/proj work space., e.g.
>
> `$MEMBERWORK/[projid]` .

Overview of filesystems available to compute nodes

Type	Access	Mount	Suggested Use
Lustre	$MEMBERWORK, $PROJWORK, $WORLDWORK	Read/Write	Batch job Input and Output
NFS	/ccs/proj	Read-only	Binaries, Shared Object Libraries, Python Scripts

> **Notice:** Due to the Meta-data overhead, the NFS areas are preferred storage locations for shared object libraries and python scripts.

Shared Object and Python Scripts

Because the /ccs/proj areas are backed-up daily and are accessible by all members of a project, the areas are very useful for sharing code with other project members. Due to the Meta-data overhead, the NFS areas are preferred storage locations for shared object libraries and python modules.

/ccs/proj Update Delay and Read-Only Access

Access to /ccs/proj area from compute nodes is read-only. The /ccs/proj areas are not directly mounted on the compute nodes, but rather rely on a periodic `rsync` to provide access to shared project-centric data. The /ccs/proj areas are mounted for read/write on the login and service nodes, but it may take up to 30 minutes for changes to propagate to the compute nodes.

> **Note:** Due to /ccs/proj areas being read-only on compute nodes, job output must be sent to a Lustre work space.

Home Directory Access Error

Batch jobs can be submitted from the User Home space, but additional steps are required to ensure the job runs successfully. Batch jobs submitted from Home areas should `cd` into a Lustre work directory prior to invoking `aprun` in the batch script. 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/use
rid
No such file or directory
```

Writing Batch Scripts

Batch scripts, or job submission scripts, are the mechanism by which a user configures and submits a job for execution. A batch script is simply a shell script that also includes commands to be interpreted by the batch scheduling software (e.g. PBS).

designated 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**

   The batch scheduler configuration 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**

   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 executable commands 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

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=atlas1%atlas2
 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 require both the atlas1 and atlas2 Lustre® filesystems to be online.
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 `date` command.
11	–	This invocation will run 24,000 MPI instances of the executable `a.out` on the compute nodes allocated by the batch system.

Additional Example Batch Scripts

exist in these Lustre-backed areas. More information can be found in the Filesystems Available to Compute Nodes (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#filesystems-available-to-compute-nodes) section.

Launching an MPI-only job

Suppose we want to launch a job on 300 Titan nodes, each using all 16 available CPU cores. The following example will request (300) nodes for 1 hour and 30 minutes. It will then launch 4,800 (300 x 16) MPI ranks on the allocated cores (one task per core).

```
#!/bin/bash
# File name: my-job-name.pbs
#PBS -A PRJ123
#PBS -l walltime=1:30:00
#PBS -l nodes=300
#PBS -l gres=atlas1%atlas2

cd $MEMBERWORK/prj123

aprun -n 4800 my-simulation.exe
```

The first line (`#PBS -A PRJ123`) PBS script will tell the system scheduler that you'd like to launch this job against the `PRJ123` allocation. For example, if you are a member of project `SCI404` , your PBS scripts would need to say `#PBS -A SCI404` instead. If you are a member of multiple projects, be careful to double check that your jobs launch with the intended allocation.

To invoke the above script from the command line, simply:

```
$ qsub my-job-name.pbs
  123456.nid00004
```

You can check the status of job number `123456` by running:

Naming Jobs

Users who submit many jobs to the queue at once may want to consider naming their jobs in order to keep track of which ones are running and which are still being held in the queue. This can be done with the `#PBS -N my-job-name` option. For example, to name your job `P3HT-PCBM-simulation-1`:

```
#!/bin/bash
# File name: simulation1.pbs
#PBS -A PRJ123
#PBS -N P3HT-PCBM-simulation-1
#PBS -l walltime=1:30:00
#PBS -l nodes=300
#PBS -l gres=atlas1%atlas2

cd $MEMBERWORK/prj123

aprun -n 4800 my-simulation.exe
```

Controlling Output

By default, when your jobs print data to `STDOUT` or `STDERR`, it gets aggregated into two files: `job-name.o123456` and `job-name.e123456` (where `123456` is your job id). These files are written into the directory from which you submitted your job with the `qsub` command.

If you wish to aggregate this output into a single file (which may help you understand where errors occur), you can join these two output streams by using the `#PBS -j oe` option. For example,

```
#PBS -N P3HT-PCBM-simulation-1
#PBS -j oe
#PBS -l walltime=1:30:00
#PBS -l nodes=300
#PBS -l gres=atlas1%atlas2

cd $MEMBERWORK/prj123

aprun -n 4800 my-simulation.exe
```

Using Environment Modules

By default, the `module` environment tool is not available in batch scripts. If you need to load modules before launching your jobs (to adjust your `$PATH` or to make shared libraries available), you will first need to import the `module` utility into your batch script with the following command:

```
source $MODULESHOME/init/<myshell>
```

where `<myshell>` is the name of your default shell.

As an example, let's load the ddt module before launching the following simulation (assuming we are using the `bash` shell):

```
#!/bin/bash
# File name: simulation.pbs
#PBS -A PRJ123
#PBS -N P3HT-PCBM-simulation
#PBS -j oe
#PBS -l walltime=1:30:00
#PBS -l nodes=300
#PBS -l gres=atlas1%atlas2

source $MODULESHOME/init/bash
module load ddt
cd $MEMBERWORK/prj123

aprun -n 4800 my-simulation.exe
```

programming environment, and may not load correctly if the programming environment is not specified first.

Basic MPI on Partial Nodes

A node's cores cannot be shared by multiple batch jobs or *aprun* jobs; therefore, a job must be allocated all cores on a node. However, users do not have to utilize all of the cores allocated to their batch job. Through *aprun* options, users have the ability to run on all or only some of a node's cores and they have some control over which cores are being used.

Reasons for utilizing only a portion of a node's cores can be: increasing memory available to each task, utilizing one floating point unit per compute unit, and increasing memory bandwidth available to each task.

Each node contains (2) NUMA nodes. Users can control CPU 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 node's task load over the (2) NUMA nodes using *aprun* `-S`. The `-j` can also be used to utilize only the integer core on each compute unit.

The following example will request 4,000 nodes for (8) hours. It will then run a 24,000 task MPI job using (6) of each allocated node's (16) cores.

```
#!/bin/bash
#PBS -A PRJ123
#PBS -N mpi-partial-node
#PBS -j oe
#PBS -l walltime=8:00:00,nodes=4000
#PBS -l gres=atlas1%atlas2

cd $MEMBERWORK/prj123

aprun -n 24000 -S 3 -j1 a.out
```

```
    234567   userid   Running   64000   00:00:44   Mon Oct 09 03:11:23
```

Please note that per Titan's scheduling policy (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan/running-jobs/#titan-scheduling-policy), the job will be charged for all 4,000 nodes.

Hybrid OpenMP/MPI

The following example batch script will request (3) nodes for (1) hour. It will then run a hybrid OpenMP/MPI job using (3) MPI tasks each running (16) threads.

```
#!/bin/bash
#PBS -A PRJ123
#PBS -N hybrid-test
#PBS -j oe
#PBS -l walltime=1:00:00,nodes=3
#PBS -l gres=atlas1%atlas2

cd $PROJWORK/prj123

export OMP_NUM_THREADS=16

aprun -n 3 -N 1 -d 16 mpi-openmp-ex.x
```

```
$ cc -mp mpi-openmp-ex.c -o mpi-openmp-ex.x
$ qsub mpi-openmp-ex.pbs
  345678.nid00004
$  showq | grep 345678
  345678   userid   Running   48   00:00:44   Mon Aug 19 21:49:18
```

Thread Performance Considerations

On Titan, each pair of CPU cores shares a single Floating Point Unit (FPU). This means that arithmetic-laden threads on neighboring CPU cores may contend for the shared FPU, which could lead to performance degradation.

```
#!/bin/bash
#PBS -A PRJ123
#PBS -N hybrid-test
#PBS -j oe
#PBS -l walltime=1:00:00,nodes=3
#PBS -l gres=atlas1%atlas2

cd $PROJWORK/prj123

export OMP_NUM_THREADS=8

aprun -n 3 -N 1 -d 8 -j 1 mpi-openmp-ex.x
```

Launching Several Executables at Once

> **Warning:** 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 50 aprun processes per batch job.

The following example will request 6,000 nodes for (12) hours. It will then run (4) MPI jobs each simultaneously running on 24,000 cores. The OS will spread each *aprun* job out such that the jobs do not share nodes.

```
#!/bin/bash
#PBS -A PROJ123
#PBS -N multi-job
#PBS -j oe
#PBS -l walltime=12:00:00,nodes=6000
#PBS -l gres=atlas1%atlas2

cd $MEMBERWORK/prj123

aprun -n 24000 a.out1 &
aprun -n 24000 a.out2 &
aprun -n 24000 a.out3 &
aprun -n 24000 a.out4 &

wait
```

```
456789   userid    Running   96000   00:00:44   Thu Oct 07 11:32:52
```

Important Considerations for Simultaneous Jobs in a Single Script

- **The `aprun` instances must be backgrounded**

  The `&` symbols in the exmaple above will place each `aprun` in the background allowing the OS to place and run each simultaneously. Without placing the `aprun` s in the background, the OS will run them serially waiting until one completes before starting the next.

- **The batch script must wait for backgrounded processes**

  The `wait` command will prevent the batch script from returning until each backgrounded `aprun` completes. Without the `wait` the script will return once each `aprun` has been started, causing the batch job to end, which kills each of the backgrounded `aprun` processes.

- **The `aprun` instances cannot share nodes**

  The system will only run one `aprun` per node; the system will *not* run multiple `aprun` instances on the same node at the same time. For exmaple, users cannot run (2) 8-core `aprun` jobs on the same node. In order to run (2) 8-core `aprun` instances at the same time, (2) nodes must be allocated.

> **Note:** `aprun` disallows multiple instances on a single node. See the wraprun section (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#using-the-aprun-command) for details regarding sharing a node's cores among multiple `aprun` instances.

Chaining Batch Jobs

The following example will

1 Submit 1.pbs which will be immediately eligible for execution

```
$ qsub 1.pbs
123451
$ qsub -W depend=afterok:123451 2.pbs
123452
```

You can then use the `showq` and `checkjob` utilities to view job states:

```
$ showq -u userid
...
123451              userid    Running    16
...
123452              userid       Hold    16
...
$ checkjob 123452
...
NOTE:  job cannot run  (dependency 123451 jobsuccessfulcomplete not met)
```

Batch Scheduler node Requests

A node's cores cannot be allocated to multiple jobs. The OLCF charges allocations based upon the computational resources a job makes *unavailable* to others. Thus, 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.

Submitting Batch Scripts

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

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` .

Interactive Batch Jobs

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 pjt000 -q debug -X -l nodes=3,walltime=30:00
```

This request will:

`-A`	Charge to the "pjt000" project
`-X`	Enables X11 forwarding. The DISPLAY environment variable must be set.
`-q debug`	Run in the debug queue
`-l nodes=3,walltime=30:00`	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.

```
ALL          4744     802      INFINITY       00:00:00       HH:MM:SS_MM/DD
```

See `showbf –help` for additional options.

Common Batch Options to PBS

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.
	`#PBS -l partition=<partition_name>`	Allocates resources on specified partition.
`-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		
-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. Since 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.

Batch Environment Variables

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

Variable	Description
$PBS_O_WORKDIR	The directory from which the batch job was *submitted*. By default, a new job starts in your home directory. You can get back to the directory of job submission with cd $PBS_O_WORKDIR . Note that this is not necessarily the same directory in which the batch script resides.
$PBS_JOBID	The job's full identifier. A common use for PBS_JOBID is to append the job's ID to the standard output and error files.

$PBS_NUM_NODES	
$PBS_JOBNAME	The job name supplied by the user.
$PBS_NODEFILE	The name of the file containing the list of nodes assigned to the job. Used sometimes on non-Cray clusters.

Modifying Batch Jobs

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.

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.

Monitoring Batch Jobs

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
```

To see all jobs submitted to partitionA:

```
$ showq -p partitionA
```

To see all completed jobs:

```
$ showq -c
```

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.
----------------------------------------------------------------------
```

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
```

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
`batch`	No explicit request required	Default; most production work runs in this queue.	See the Titan Scheduling Policy (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan/running-jobs/#titan-scheduling-policy) for details.
`killable`	`#PBS -q killable`	Opportunistic; jobs start even if they will not complete before the onset of a scheduled outage.	
`debug`	`#PBS -q debug`	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 availalable prior to a scheduled outage.

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan/running-jobs/#titan-scheduling-policy) page.

Job Execution on Titan

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.

> **Warning:** On Titan, each batch job is limited to 200 simultaneous processes. Attempting to open more simultaneous processes than the limit will result in `No space left on device` errors.

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.

Using the aprun command

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description) 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

`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 50 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 50)
```

Warning: Users are limited to 50 aprun processes per batch job.

Single-*aprun* Process Ensembles with *wraprun*

Wraprun is a utility that enables independent execution of multiple MPI applications under a single aprun call. It borrows from aprun MPMD syntax and also contains some wraprun specific syntax. The latest documentation can be found on the wraprun development README (https://web.archive.org/web/20190219213055/https://github.com/olcf/wraprun).

In some situations, the simultaneous *aprun* limit can be overcome by using the utility *wraprun*. Wraprun has the capacity to run an arbitrary number and combination of qualified MPI or serial applications under a single *aprun* call.

Note: MPI executables launched under *wraprun* must dynamically linked. Non-MPI applications must be launched using a serial wrapper included with *wraprun*.

Using Wraprun

By default wraprun applications must be dynamically linked. Wraprun itself depends on Python and is compatible with python/2.7.X and python/3.X but requires the user to load their preferred python environment module before loading wraprun. Below is an example of basic wraprun use for the applications foo.out and bar.out.

```
$ module load dynamic-link
$ cc foo.c -o foo.out
$ cc bar.c -o bar.out
$ module load python wraprun
$ wraprun -n 80 ./foo.out : -n 160 ./bar.out
```

foo.out and bar.out will run independently under a single aprun call.

In addition to the standard process placement flags available to aprun, the `--w-cd` flag can be set to change the current working directory for each executable:

```
$ wraprun -n 80 --w-cd /foo/dir ./foo.out : -n 160 --w-cd /bar/dir ./bar.out
```

This is particularly useful for legacy FORTRAN applications that use hard coded input and output file names.

Multiple instances of an application can be placed on a node using comma-separated PES syntax PES1,PES2,...,PESN syntax, for instance:

```
$ wraprun -n 2,2,2 ./foo.out [ : other tasks...]
```

would launch 3 two-process instances of foo.out on a single node.

In this case, the number of allocated nodes must be at least equal to the sum of processes in the comma-separated list of processing elements, divided by the maximum number of processes per node.

```
$ wraprun -n 2,2,2 --w-cd /foo/dir1,/foo/dir2,/foo/dir3 ./foo.out [ : other tasks...]
```

For nonMPI executables a wrapper application, serial, is provided. This wrapper ensures that all executables will run to completion before aprun exits. To use, place serial in front of your application and arguments:

```
$ wraprun -n 1 serial ./foo.out -foo_args [ : other tasks...]
```

The stdout/err of each task run under wraprun will be directed to to it's own unique file in the current working directory with names in the form:

```
${PBS_JOBID}_w_${TASK_ID}.out
${PBS_JOBID}_w_${TASK_ID}.err
```

Recommendations and Limitations

It is recommended that applications be dynamically linked. On Titan this can be accomplished by loading the dynamic-link module before invoking the Cray compile wrappers CC,cc, ftn.

The library may be statically linked although this is not fully supported.

All executables must reside in a compute node visible filesystem, e.g. Lustre. The underlying `aprun` call made by `wraprun` enforces the `aprun` 'no-copy' ('-b') flag.

`wraprun` works by intercepting all MPI function calls that contain an MPI_Comm argument. If an application calls an MPI function, containing an MPI_Comm argument, not included in src/split.c, the results are undefined.

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.

-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.

> **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.

-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.

> **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.

-j

> **For Titan:** Number of CPUs to use per *paired-core compute unit* (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description). The -j parameter specifies the number of CPUs to be allocated per *paired-core compute unit* (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description). 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).

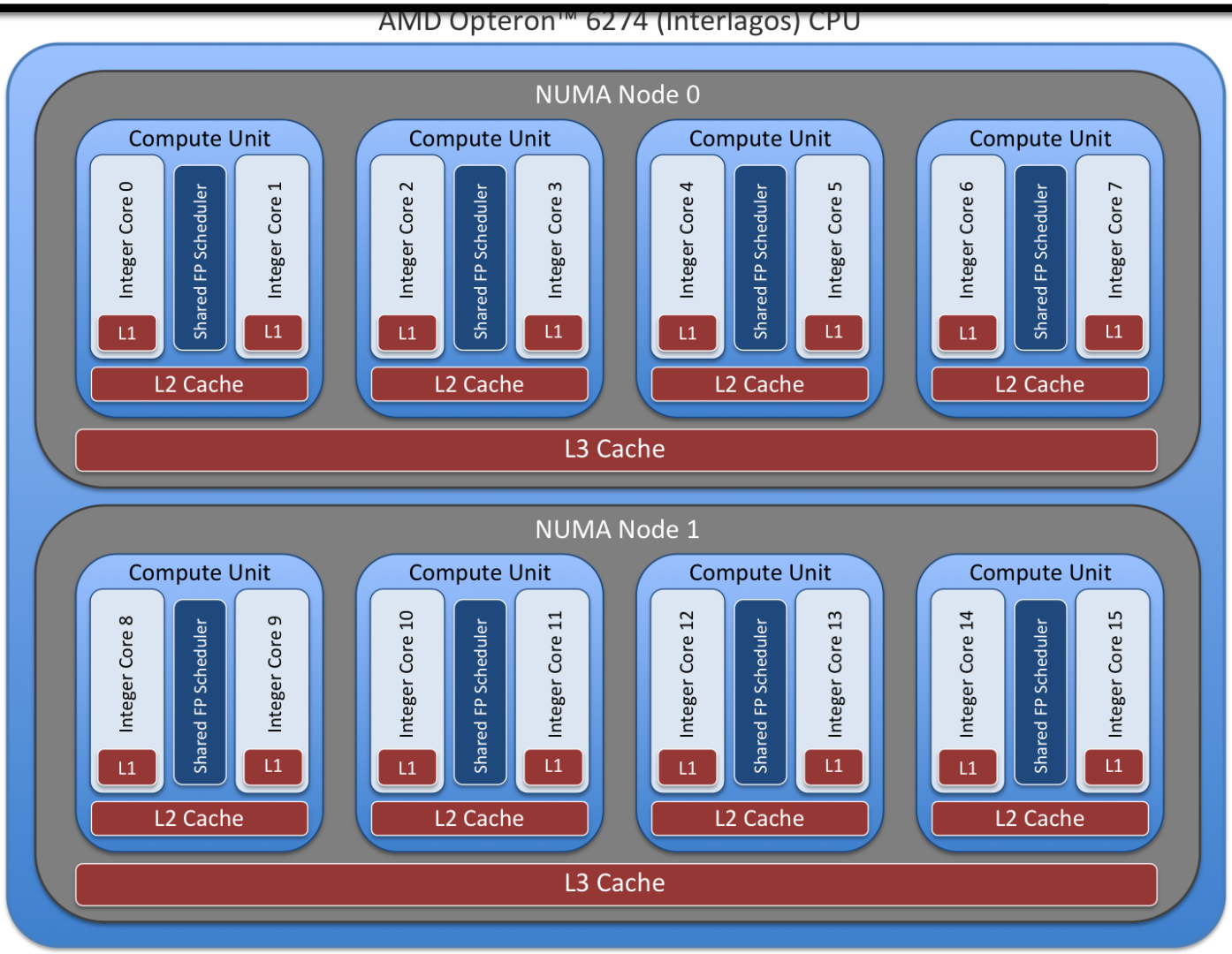
> **For Eos:** The -j parameter controls Hyper Threading. The valid values for -j are 0 (use the system default), 1 (turn Hyper Threading off), and 2 (turn Hyper Threading on; this is the system default).

-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.

XK7 CPU Description

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.



Controlling MPI Task Layout Within a Physical Node

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

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.

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							

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* (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#xk7-cpu-description). 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

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*

Controlling MPI Task Layout Across Many Physical Nodes

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. The default MPI task layout is SMP-style. This means MPI will sequentially allocate all virtual cores on one physical node before allocating tasks to another physical node.

Viewing 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.

```
# On Titan
$ aprun -n 32 ./a.out

# On Eos, Hyper-threading must be disabled:
$ aprun -n 32 -j1 ./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

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

# On Eos, Hyper-threading must be disabled:
$ aprun -n 32 -j1 ./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

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					

Controlling Thread Layout Within a Physical Node

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`.

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):

```
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
Rank 0, Thread 4, Node 0, NUMA 0, Core 4 <-- slave
Rank 0, Thread 5, Node 0, NUMA 0, Core 5 <-- slave
Rank 0, Thread 6, Node 0, NUMA 0, Core 6 <-- slave
Rank 0, Thread 7, Node 0, NUMA 0, Core 7 <-- slave
Rank 0, Thread 8, Node 0, NUMA 1, Core 0 <-- slave
Rank 0, Thread 9, Node 0, NUMA 1, Core 1 <-- slave
Rank 0, Thread 10,Node 0, NUMA 1, Core 2 <-- slave
Rank 0, Thread 11,Node 0, NUMA 1, Core 3 <-- slave
Rank 0, Thread 12,Node 0, NUMA 1, Core 4 <-- slave
Rank 0, Thread 13,Node 0, NUMA 1, Core 5 <-- slave
Rank 0, Thread 14,Node 0, NUMA 1, Core 6 <-- slave
Rank 0, Thread 15,Node 0, NUMA 1, Core 7 <-- slave
Rank 1, Thread 0, Node 1, NUMA 0, Core 0 <-- MASTER
Rank 1, Thread 1, Node 1, NUMA 0, Core 1 <-- slave
Rank 1, Thread 2, Node 1, NUMA 0, Core 2 <-- slave
Rank 1, Thread 3, Node 1, NUMA 0, Core 3 <-- slave
Rank 1, Thread 4, Node 1, NUMA 0, Core 4 <-- slave
Rank 1, Thread 5, Node 1, NUMA 0, Core 5 <-- slave
Rank 1, Thread 6, Node 1, NUMA 0, Core 6 <-- slave
Rank 1, Thread 7, Node 1, NUMA 0, Core 7 <-- slave
Rank 1, Thread 8, Node 1, NUMA 1, Core 0 <-- slave
Rank 1, Thread 9, Node 1, NUMA 1, Core 1 <-- slave
Rank 1, Thread 10,Node 1, NUMA 1, Core 2 <-- slave
Rank 1, Thread 11,Node 1, NUMA 1, Core 3 <-- slave
Rank 1, Thread 12,Node 1, NUMA 1, Core 4 <-- slave
Rank 1, Thread 13,Node 1, NUMA 1, Core 5 <-- slave
Rank 1, Thread 14,Node 1, NUMA 1, Core 6 <-- slave
Rank 1, Thread 15,Node 1, NUMA 1, Core 7 <-- 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 (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#xk7-cpu-description). This requests (1) physical compute nodes and requires a nodes request of (1):

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

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			Rank1 Thread0	Rank1 Thread1	Rank1 Thread2	Rank1 Thread3	Rank1 Thread4	Rank1 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 (https://web.archive.org/web/20190219213055mp_/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/titan-user-guide/#xk7-cpu-description). 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
Rank 2, Thread 0, Node 1, NUMA 0, Core 0 <-- MASTER
Rank 2, Thread 1, Node 1, NUMA 0, Core 1 <-- slave
Rank 3, Thread 0, Node 1, NUMA 1, Core 0 <-- MASTER
Rank 3, Thread 1, Node 1, 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* (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description). This example will launch (2) MPI tasks, each with (4) threads. Place only (1) MPI task [and its (4) threads] on each NUMA node (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description). One core per *paired-core compute unit*

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description)*. This example will launch (1) MPI tasks, each with (8) threads. One core per *paired-core compute unit* (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description) 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

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
```

Running Accelerated Applications on Titan

Each of Titan's 18,688 compute nodes contains an NVIDIA K20X accelerator, the login and service nodes do not. As such the only way to reach a node containing an accelerator is through aprun (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/kb_articles/using-the-aprun-command/). For more details on the types of nodes that constitute Titan please see login vs service vs compute nodes (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/kb_articles/login-vs-service-vs-compute-nodes/). No additional steps are required to access the GPU beyond what is required by the acceleration technique used. Titan does possess a few unique accelerator characteristics that are discussed below.

Accelerator Modules

cudatoolkit/). This module contains access to NVIDIA provided tools such as nvcc as well as libraries such as the CUDA run time. When the **cudatoolkit** module is loaded shared linking will be enabled in the Cray compiler wrappers CC,cc,ftn. The **craype-accel-nvidia35** module will load the **cudatoolkit** as well as set several accelerator options used by the Cray toolchain.

Compiling and Linking CUDA

When compiling on Cray machines, the Cray compiler wrappers (e.g. `cc`, `CC`, and `ftn`) work in conjunction with the modules system to link in needed libraries such as MPI; it is therefore recommended that the Cray compiler wrappers be used to compile CPU portions of your code.

To generate compiler-portable code it is necessary to compile CUDA C and CUDA runtime containing code with NVCC. The resulting NVCC compiled object code must then be linked in with object code compiled with the Cray wrappers. NVCC performs GNU style C++ name mangling on compiled functions so care must be taken in compiling and linking codes.

The section below briefly covers this technique. For complete examples, please see our tutorial on Compiling Mixed GPU and CPU Code (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/tutorials/compiling-mixed-gpu-and-cpu-code/).

C++ Host Code

When linking C++ host code with NVCC compiled code, the C++ code must use GNU-compatible name mangling. This is controlled through compiler specific wrapper flags.

PGI Compiler

```
$ nvcc -c GPU.cu
$ CC --gnu CPU.cxx GPU.o
```

```
$ nvcc -c GPU.cu
$ CC CPU.cxx GPU.o
```

C Host Code

NVCC name mangling must be disabled if it is to be linked with C code. This requires the `extern "C"` function qualifier be used on functions compiled with NVCC but called from `cc` compiled code.

```
extern "C" void GPU_function()
{
...
}
```

Fortran: Simple

To allow C code to be called from Fortran, one method requires that the C function name be modified. NVCC name mangling must be disabled if it is to be linked with Fortran code. This requires the `extern "C"` function qualifier. Additionally, function names must be lowercase and end in an underscore character (i.e., _).

NVCC Compiled

```
extern "C" void gpu_function_()
{
...
}
```

`ftn` Compiled

```
call gpu_function()
```

Fortran: `ISO_C_BINDING`

`ISO_C_BINDING` provides Fortran a greater interoperability with C and removes the need to modify the C function name. Additionally the `ISO_C_BINDING` guarantees data type compatibility.

```
extern "C" void gpu_function()
{
...
}
```

`ftn` Compiled

```
module gpu
  INTERFACE
    subroutine gpu_function() BIND (C, NAME='gpu_function')
      USE ISO_C_BINDING
      implicit none
    end subroutine gpu_function
  END INTERFACE
end module gpu

...

call gpu_function()
```

CUDA Proxy

The default GPU compute mode for Titan is *exclusive process*. In this mode, many threads within a process may access the GPU context. To allow multiple processes access to the GPU context, such as multiple MPI tasks on a single node accessing the GPU, the *CUDA proxy server* was developed. Once enabled, the CUDA proxy server transparently manages work issued to the GPU context from multiple processes.

> **Warning:** Currently, GPU memory between processes accessing the proxy is not guarded, meaning process *i* can access memory allocated by process *j*. This **SHOULD NOT** be used to share memory between processes and care should be taken to ensure process only access GPU memory they have allocated themselves.

How to Enable

To enable the proxy server the following steps must be taken before invoking aprun:

Issues

Currently GPU debugging and profiling are not supported when the proxy is enabled. On Titan, specifying the qsub flag `-l feature=gpudefault` will switch the compute mode from exclusive process to the CUDA default mode. In the default mode, debugging and profiling are available, and multiple MPI ranks will be able to access the GPU. The default compute mode is **not** recommended on Titan. In the default compute mode approximately 120 MB of device memory is used per processes accessing the GPU. Additionally, inconsistent behavior may be encountered under certain conditions.

GPUDirect: CUDA-enabled MPICH

Cray's implementation of MPICH2 allows GPU memory buffers to be passed directly to MPI function calls, eliminating the need to manually copy GPU data to the host before passing data through MPI. Several examples of using this feature are given below.

How to Enable

To enable GPUDirect the following steps must be taken before invoking aprun:

```
$ export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:$LD_LIBRARY_PATH
$ export MPICH_RDMA_ENABLED_CUDA=1
```

Optimizations

Several optimizations for improving performance are given below. These optimizations are highly application dependent and may require some trial-and-error tuning to achieve best results.

Pipelining

Pipelining allows for overlapping of GPU to GPU MPI messages and may improve message passing performance for large bandwidth-bound messages. Setting the environment variable `MPICH_G2G_PIPELINE=N` allows a maximum of *N* GPU to GPU messages to be in flight at any given

Nemesis

Applications using asynchronous MPI calls may benefit from enabling the MPICH asynchronous progress feature. Setting the `MPICH_NEMESIS_ASYNC_PROGRESS=1` environment variable enables additional threads to be spawned to progress the MPI state.

This feature requires that the thread level be set to multiple: `MPICH_MAX_THREAD_SAFETY=multiple`.

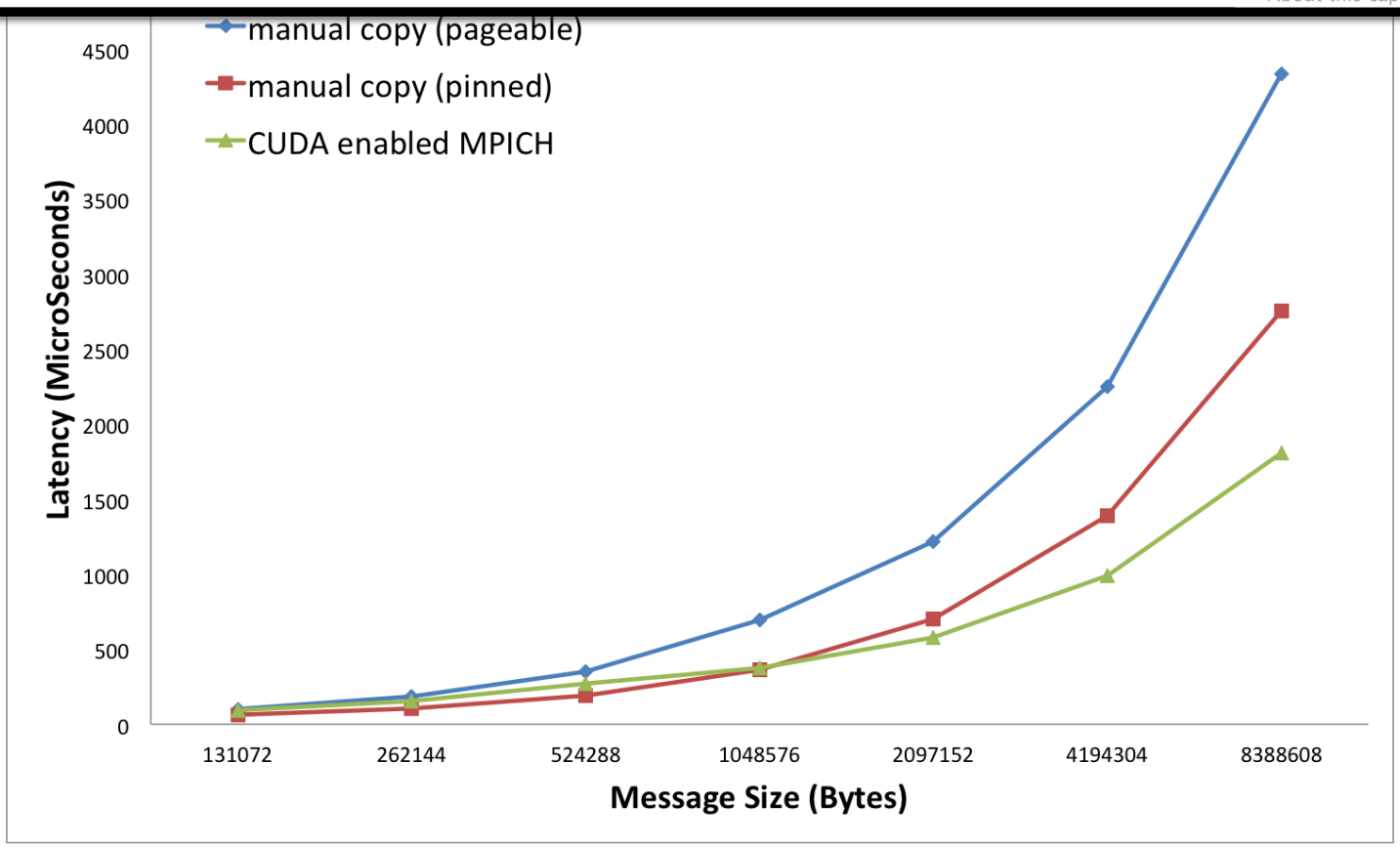
This feature works best when used in conjunction with core specialization: `aprun -r N`, which allows for *N* CPU cores to be reserved for system services.

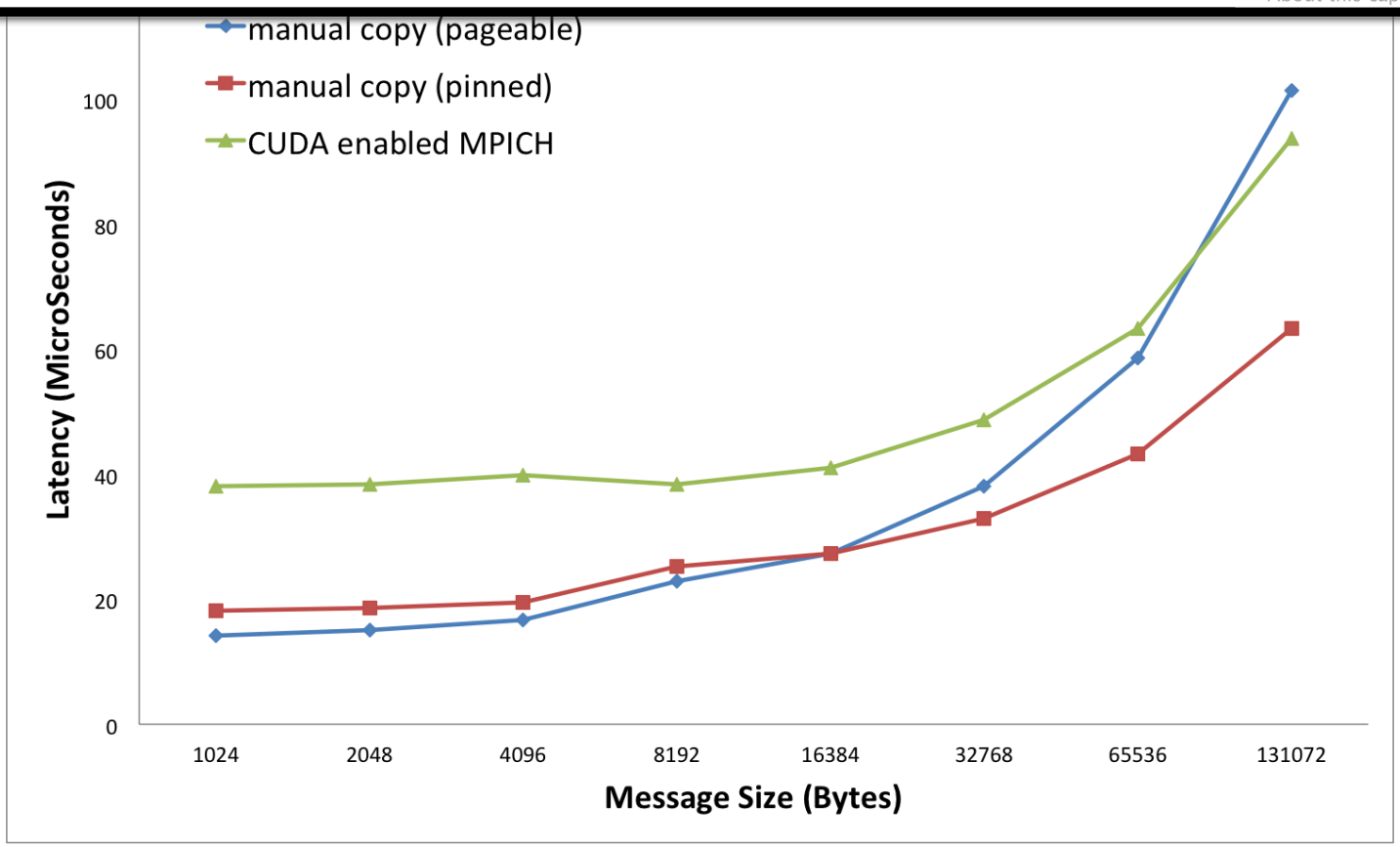
Example

Several examples are provided in our GPU Direct Tutorial (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/tutorials/gpudirect-mpich-enabled-cuda/).

Microbenchmark

The following benchmarks were performed with cray-mpich2/6.1.1.

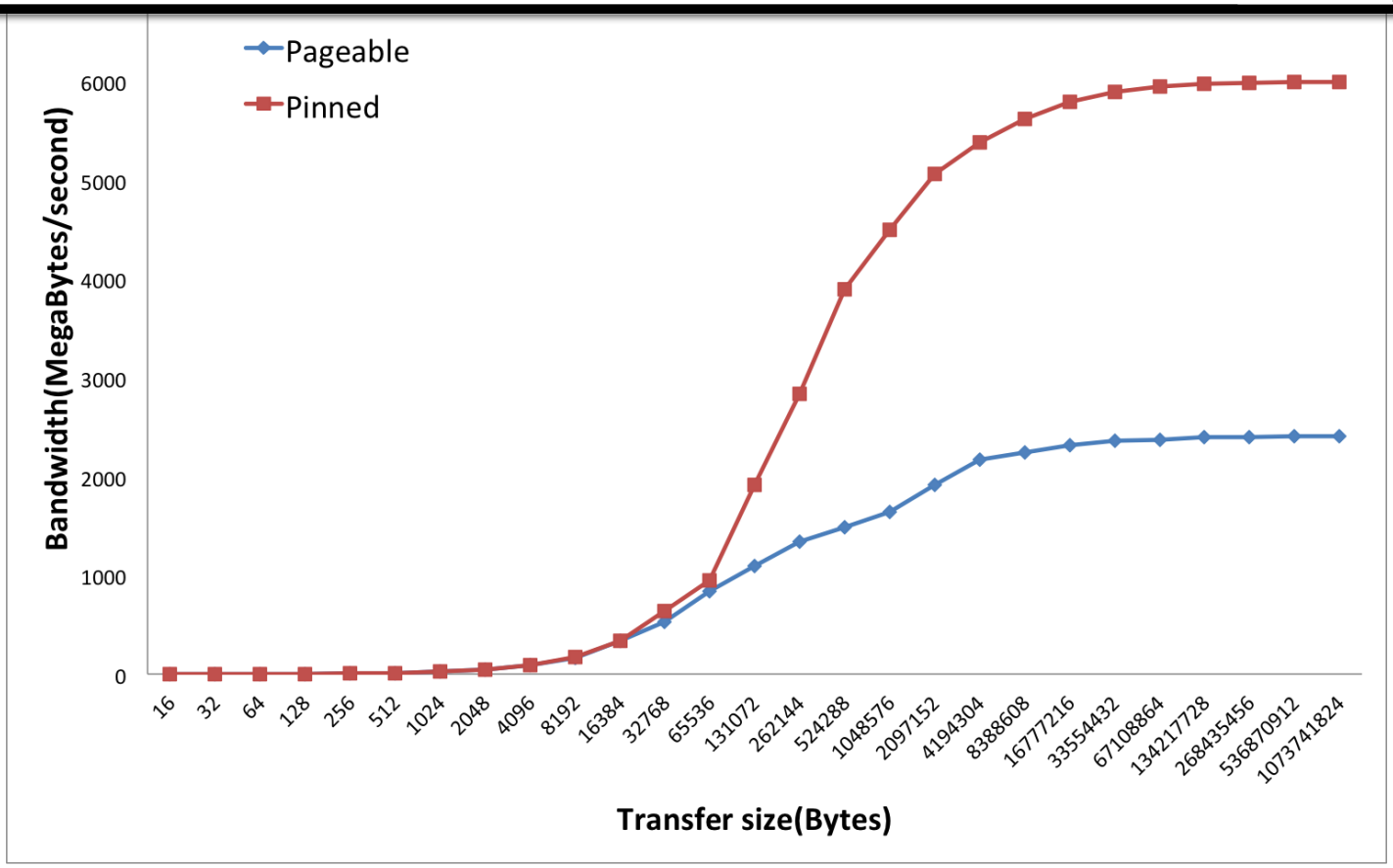




Pinned Memory

Memory bandwidth between the CPU (host) and GPU (device) can be increased through the use of pinned, or page-locked, host memory. Additionally, pinned memory allows for asynchronous memory copies.

To transfer memory between the host and device, the device driver must know the host memory is pinned. If it does not, the memory will be first copied into a pinned buffer and then transferred, effectively lowering copy bandwidth. For this reason, pinned memory usage is recommended on Titan.



Job Resource Accounting

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.

Titan Core-Hour Calculation

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

```
Titan core-hours = nodes requested * 30 * ( batch job endtime - 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*.

Note: Projects are allocated time on Titan in units of "Titan core-hours". Other OLCF systems are allocated in units of "core-hours".

Viewing Allocation Utilization

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 (https://web.archive.org/web/20190219213055/https://users.nccs.gov/dashboard?do=login).

On the Command Line via `showusage`

```
$ showusage
Usage on titan:
                                  Project Totals          <userid>
 Project      Allocation        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 (https://web.archive.org/web/20190219213055/https://users.nccs.gov/dashboard?do=login). 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.

Titan Scheduling Policy

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

> **Note:** The command line utility `$ mdiag -p` can be used to see the individual factors contributing to a job's priority.

If your jobs require resources outside these queue policies, please complete the relevant request form on the Special Requests (/web/20190219213055/https://www.olcf.ornl.gov/for-users/getting-started/special-request-form/) 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	126	312	6.0	0
5	1	125	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)

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 (4) *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. Any additional jobs will be blocked until one of the running jobs completes.

> **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.

- 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

the percentage that the project is over its allocation, as shown in the table below:

% Of Allocation Used	Priority Reduction	number eligible-to-run	number running
< 100%	0 days	4 jobs	unlimited jobs
100% to 125%	30 days	4 jobs	unlimited jobs
> 125%	365 days	4 jobs	1 job

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 (https://web.archive.org/web/20190219213055/http://www.olcf.ornl.gov/support/getting-started/special-request-form/) 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).`

Aprun Tips

- 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 floa
ting-point unit contention
```

Each Titan compute node (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#xk7-cpu-description) 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. 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`

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:

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:

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#job-execution-on-titan) 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.

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

while (not successful)

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

sleep 120

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.

Working Around Hanging Apruns

This simple script demonstrates how to kill a job that is not making forward progress and start again without going back to the scheduling queue. A job may not be making forward progress for a variety of reasons including hardware and software failures. The script does not help a user in identifying the root cause of why a job is hung. The goal of this script is to help a user do the following things:

1. Detect that a job is hung (i.e., the job is not making forward progress).
2. Kill the hung job.
3. Restart the application without having to go back to the scheduling queue.

period, then the script tags the jobs as hung and takes further action. There are two key items that a user needs to do for this step to work correctly. The `OUTFILE` and `WINDOW` variables have to be set appropriately. The `OUTFILE` variable corresponds to the output file which this script watches periodically. The `WINDOW` variable is the longest time interval (in minutes) after which the script tags the job as hung if there is no write to `OUFILE` . Currently, the `WINDOW` variable is set to 120 mins, but it can be changed as needed.

If a job is detected to be hung, then the script automatically kills the job by obtaining its APID without user intervention.

Finally, the script automatically attempts to restart the job by relaunching the aprun with the same parameters. For this to work correctly, the user is advised to allocate a couple of more nodes than what is used in the aprun command. This is illustrated in the script. The user can change the number of such restart trial by changing the loop iteration counter as desired (“for i in `seq 1 4`;”). If the user does not allocate a few spare nodes, the application will not restart correctly if there any hardware problem with one of the allocated node.

```
#PBS -m e
#PBS -A ABC123

WINDOW=120 # 2 hour window of no activity/progress, while loop checks the file every minu
te
USER=`whoami`
BINDIR="/lustre/atlas1/abc123/scratch/${USER}/apkill"
cd $BINDIR

for i in `seq 1 4`;
do
    aprun -n 1 ./a.out $i &  # the "&" at the end is essential so that the code below exe
cuted, the code below monitors the temporary output file
    #echo "loop = $i"

##############################################################################################
############################
    # Snippet to be moved to application PBS/qsub script:
    # Make sure to set the variable USER and WINDOW same as above or appropriately
    # Flow: store the status (number of lines) of temporary output file and keep checking
every minute for updates,
    # if it is being updated, keep within while loop, if not updating for a long duration
(2 hours), do apkill.
##############################################################################################
############################
    OUTFILE="$PBS_JOBID.OU"
    OUTLEN=`wc -l $OUTFILE | awk '{print $1}'`
    #echo "outlen = $OUTLEN"
    TIME=0;
    while true
    do
        sleep 60; # sleep in number of seconds
        OUTLEN_NEW=`wc -l $OUTFILE | awk '{print $1}'`
        #echo "len = $OUTLEN and $OUTLEN_NEW"
        if [ $OUTLEN -eq $OUTLEN_NEW ]; then
            TIME=`expr $TIME + 1`
        else
            TIME=0;
            OUTLEN=$OUTLEN_NEW
        fi

        #echo "len after = $OUTLEN and $OUTLEN_NEW"

        APID=`apstat | grep $USER | tail -n 1 | awk '{print $1}'`
        #echo "apid = $APID"
        if [ -n "$APID" ]; then
```

```
        else
            break # break the while loop if there is no APID found
        fi
    done
###############################################################################################
############################
    #end of snippet to be moved to application pbs script
###############################################################################################
############################

done
wait
```

Troubleshooting and Common Errors

Below are some error messages that you may encounter. Check back often, as this section will be updated.

Error Message: claim exceeds reservation's nodes

This may occur if the batch job did not request enough nodes. In this case, request more nodes in the #PBS -lnodes line of your batch script.

In some cases, the error may occur if a troubled node is allocated to the job.

In this case, it may be useful to:

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.

Here is an example pseudo code.

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

sleep 30

end
```

The purpose of the loop is to attempt to run the aprun process on the extra nodes requested in the batch script if 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.

Error Message: MPICH2 ERROR

```
Error Message:  MPICH2 ERROR [Rank 65411] [job id 2526230] [Thu May 16 04:17:23 2013] [c1
8-3c1s6n0] [nid07084] - MPID_nem_gni_check_localCQ(): GNI_CQ_EVENT_TYPE_POST had error (S
OURCE_SSID_DREQ:MDD_INV)
```

Recommendation: Resubmit job. Possible causes of this error could be system issues. If the error reoccurs you may have an error in your code.

Error Message: Received node event ec_node_failed

```
Error Message:  [NID 04228] 2013-05-10 08:08:28 Apid 2509826 killed. Received node event
ec_node_failed for nid 4313
```

Explanation: Sometimes a node will be in an unstable state but the system will still consider it to be up. When a job runs on it and fails, the system software sees that failure and then marks the node down.

Error Message: CUDA_ERROR_LAUNCH_FAILED

```
Error Message:  ACC: craylibs/libcrayacc/acc_hw_nvidia.c:548 CRAY_ACC_ERROR -  cuStreamSy
nchronize returned CUDA_ERROR_LAUNCH_FAILED[NID 11408] 2013-05-10 01:57:46 Apid 2508898:
initiated application termination
```

Recommendation: Try to resubmit the job. Possible causes of this error could be system issues. If the error reoccurs you may have race conditions or other errors in your code. Contact user support if you have questions.

Error Message: CUDA driver error 700

Recommendation: Try to resubmit the job. Possible causes of this error could be system issues. If the error reoccurs you may have race conditions or other errors in your code. Contact user support if you have questions.

Cray HSN detected criticalerror

```
[NID ####] 2013-01-01 23:55:00 Apid #######: Cray HSN detected criticalerror 0x40c[ptag 2
49]. Please contact admin for details. Killing pid ###(@@##)
```

Recommendation: This could be a problem with the user code, MPI library, or compiler.

0x40c decodes to (GHAL_ERROR_MASK_FMA:HT_BAD_NP_REQUEST)

This can happen when a code has corrupted memory in some way that leads to loads targeting the fma window. It is possible for this error to show up whenever the application has somehow generated an address to load/store to that is not valid. Typically this would in fact cause a segmentation fault since the process would take a page fault and the kernel would see that the vaddr involved in the load/store was invalid, thus raising a sigsegv. However, on Gemini, there are 3 FMA windows, each 1 GB in size mapped

even a store into the window will trigger an error since the window has to be prepared for accepting stores out on to the network. The routines to do this preparation aren't available to end user apps.If the number of nodes used in on the order of 10, we recommend that the user enable core dumps and try a debug run with:

```
export MPICH_NEMESIS_NETMOD=tcp
```

This will tell MPI to use TCP instead of the low-level Gemini interface. Performance will be worse, but the code should segfault and dump core instead of be killed by the Gemini driver.

To enable core dumps, place one of the following commands in your batch script before the aprun call:

```
ulimit -c unlimited (if you're using sh/ksh/bash)
limit coredumpsize unlimited (if you're using csh/tcsh)
```

You may want to first recompile your code and add the " `-g` " option to compile commands. This will enable debugging information and will make it easier to pinpoint the source of the problem.

The module list command will show you the current versions of MPI and and compilers that you have loaded.

CRAY_CUDA_MPS=1 Segmentation Faults

Issue: When using OpenCL, if I set `CRAY_CUDA_MPS=1` (turn on Proxy), I get seg. faults no matter what I do until I release the node.

Recommendation: It is a known issue that `CRAY_CUDA_MPS=1` is incompatible with OpenCL. Do not use `CRAY_CUDA_MPS=1` with Open CL.

Possible DNS Spoofing Detected on DTNs

Error: You try to login to dtn.ccs.ornl.gov and you get this message:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for dtn has changed,
and the key for the according IP address 160.91.202.138
is unchanged. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
Offending key for IP in /ccs/home/suzanne/.ssh/known_hosts:97
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ 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.
The fingerprint for the RSA key sent by the remote host is
b3:31:ac:44:83:2b:ce:37:cc:23:f4:be:7a:40:83:85.
Please contact your system administrator.
Add correct host key in /ccs/home/user/.ssh/known_hosts to get rid of this message.
Offending key in /ccs/home/user /.ssh/known_hosts:106
RSA host key for dtn has changed and you have requested strict checking.
Host key verification failed.
```

Reason: We have just changed dtn.ccs.ornl.gov to point to dtn03 and dtn04 rather than dtn01 and dtn02. The ssh client will notice that the key signatures for dtn.ccs.ornl.gov no longer match those that were stored in your `/ccs/home/user/.ssh/known_hosts` file.

Resolution: You must remove the dtn key signatures from `/ccs/home/user/.ssh/known_hosts`. From OLCF resources like home.ccs.ornl.gov, you may do this by issuing

```
ssh-keygen -R dtn.
```

You may also need to do this on your desktop machine if you log directly into the dtns. If your desktop does not have ssh-keygen, you can manually remove all the dtn signatures from you desktop's `/.ssh/know_hosts` with vi or any text editor.

Debugging

debugging.

Arm DDT

Arm DDT is an advanced debugging tool used for scalar, multi-threaded, and large-scale parallel applications.
In addition to traditional debugging features (setting breakpoints, stepping through code, examining variables), DDT also supports attaching to already-running processes and memory debugging. In-depth debugging information is beyond the scope of this guide, and is best answered by the Arm Forge User Guide (https://web.archive.org/web/20190219213055/https://developer.arm.com/docs/101136/latest).

Additional DDT Articles

In addition to the information below, the following articles can help you perform specific tasks with DDT:

- Using the Forge Remote Client (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/)
- Fixing Memory Leaks with DDT Leak Reports (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/fixing-memory-leaks-with-ddt-leak-reports/)

Launching DDT

guides/titan/connecting/#x11-forwarding), or by running a remote client on your local machine, and connecting it to the remote machine.

The remote client provides a native GUI (for Linux / OS X / Windows) that should be more far more responsive than X11, but requires a little extra setup. It is also useful if you don't have a preconfigured X11 server.

To get started using the remote client, follow the Forge Remote Client setup guide (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/).

To use X11 forwarding, in a terminal do the following:

```
$ ssh -X user@<host>.ccs.ornl.gov
$ module load forge
$ ddt &
```

Running your job

Once you have launched a DDT GUI, we can initiate a debugging session from a batch job script using DDT's "Reverse Connect" functionality. This will connect the debug session launched from the batch script to an already running GUI.

This is the most widely applicable method of launching, and allows re-use of any setup logic contained in existing batch scripts.

(This method can also be easily modified to launch DDT from an interactive batch session.)

- Copy or modify an existing job script. (If you don't have an existing job script, you may wish to read the section on letting DDT submit your job to the queue).
- Include the following near the top of your jobs script:

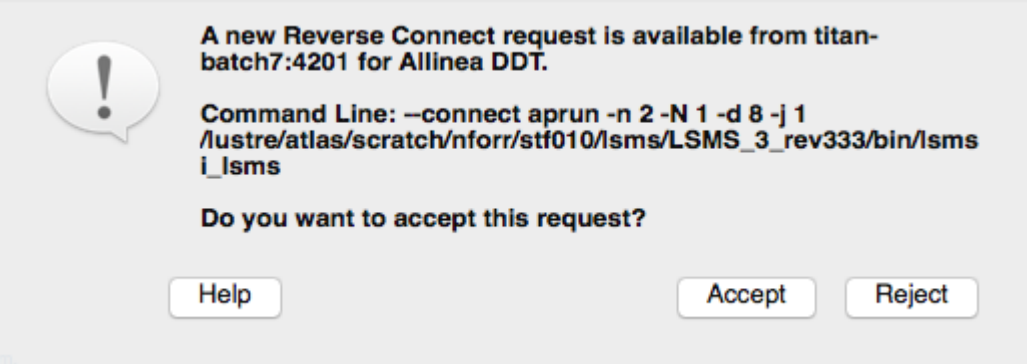
- Finally, prefix the `aprun` / `mpirun` with `ddt --connect`, e.g.:

```
$ aprun -n 1024 -N 8 ./myprogram
```

becomes:

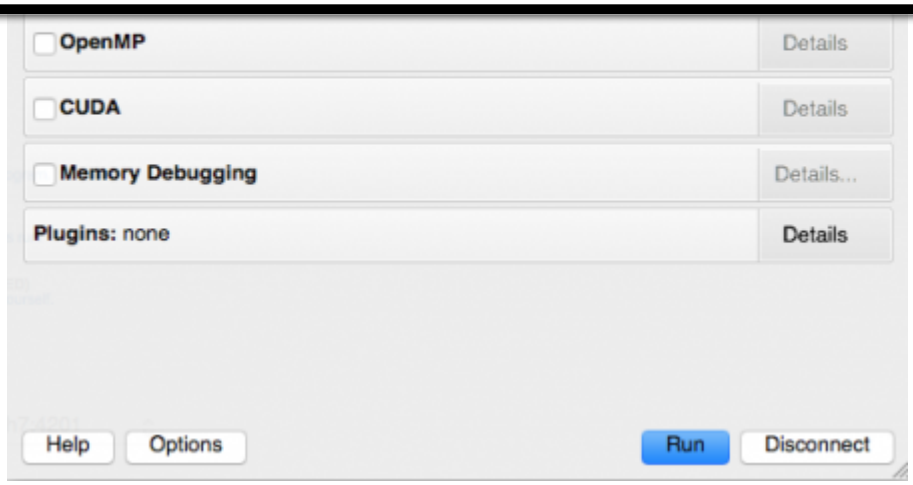
```
$ ddt --connect aprun -n 1024 -N 8 ./myprogram
```

After submitting this script to the batch system (and waiting for it to be scheduled), a prompt will appear in the DDT GUI asking if you would like to debug this job.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2013/09/ddt-reverse-connect-prompt.png)

Once accepted, you can configure some final options before launching your program.



(https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/wp-content/uploads/2013/09/ddt-reverse-connect-run-dialog.png)

Offline Debugging

In addition to debugging interactively, DDT also supports "Offline Debugging". This can be particularly useful if your job takes a long time to schedule (and you're not sure if you'll be available when it runs).

DDT will execute you program under the debugger, and write a plain text or HTML report for you to inspect at your convenience.

To run your program with DDT's Offline Debugging, modify your existing job script and modify your aprun command such that:

```
$ aprun -n 1024 -N 8 ./myprogram
```

Would become:

```
$ ddt --offline=output.html aprun -n 1024 -N 8 ./myprogram
```

Replacing printf / debug statements with tracepoints

Rather than adding logging statements, you can add tracepoints inside DDT. Tracepoints have the following advantages over debug statements:

- No source code modification – this means there's no need to recompile, and no need to track down and remove logging statements after debugging.
- Scalability – variables can be collected and summarized over thousands of processes without worrying about where/how to store the output, or how to sift through the data afterwards.
- Variables are automatically compared across processes. Variables with differing values across processes are highlighted and sparklines are included to give a quick graphical representation of the distribution on values.

For more information on tracepoints (including how to use them with interactive debugging), please see the Forge user guide (https://web.archive.org/web/20190219213055/https://developer.arm.com/docs/101136/latest). (Section 6.14 Tracepoints refers to tracepoints in general, while syntax can be found in section 15 DDT: Offline Debugging).

Attaching to a running job

You can also use DDT to connect to an already-running job. To do so, you must be connected to the system on which the job is running. You do not need to be logged into the job's head node (the node from which `aprun` / `mpirun` was launched), but DDT needs to know the head node. The process is fairly simple:

1. Find your job's head node:
   - On Titan and Eos, run `qstat -f <jobid> | grep login_node_id` . The node listed is the head node.

2 Start DDT by running `module load forge` and then `ddt`.

3 When DDT starts, select the option to "**Attach to an already running program**".

4 In that dialog box, make sure the appropriate MPI implementation is selected. If not, click the "Change MPI" button and select the proper one.

5 If the job's head node is not listed after the word "Hosts", click on "Choose Hosts".

- Click "Add".
- Type the host name in the resulting dialog box and click "OK".
- To make things faster, uncheck any other hosts listed in the dialog box.
- Click "OK" to return.

6 Once DDT has finished scanning, your job should appear in the "Automatically-detected jobs" tab, select it and click the "Attach" button.

Letting DDT submit your job to the queue

This method can be useful when using the Forge Remote Client (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/) or when your program doesn't have a complex existing launch script.

1 Run `module load forge`.

2 Run `ddt`.

3 When the GUI starts, click the "**Run and debug a program**" button.

4 In the DDT "Run" dialog, ensure the "**Submit to Queue**" box is checked.

5 Optionally select a queue template file (by clicking "**Configure**" by the "**Submit to Queue**" box). If your typical job scripts are basically only an `aprun` command, the default is fine.

Guide (https://web.archive.org/web/20190219213055/https://developer.arm.com/docs/101136/latest).

6 Click the "**Parameters**" button by the "**Submit to Queue**" box.

7 In the resulting dialog box, select an appropriate walltime limit, account, and queue. Then click "**OK**".

8 Enter your executable in the "**Application**" box, enter any command line options your executable takes on the "**Arguments**" line, and select an appropriate number of processes and threads.

9 Click "**Submit**". Your job will be submitted to the queue and your debug session will start once the job begins to run. While it's waiting to start, DDT will display a dialog box displaying `showq` output.

Starting DDT from an interactive-batch job

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

Starting DDT from within an interactive job gives you the advantage of running repeated debug sessions with different configurations while only waiting in the queue once.

1 Start your interactive-batch job with `qsub -I -X ...` (`-X` enables X11 forwarding).

2 Run `module load forge`.

3 Start DDT with the command `ddt`.

4 When the GUI starts, click the "**Run and debug a program**" button.

5 In the DDT "Run" dialog, ensure the "**Submit to Queue**" box is **not** checked.

Memory Debugging (on Cray systems)

In order to use the memory debugging functionality of DDT on Titan, you need to link against the DDT memory debugging library. (On non-Cray systems DDT can preload the shared library automatically
if your program uses dynamic linking).

In order to link the memory debugging library:

1. `module load ddt` (This determines the location of the library to link).
2. `module load ddt-memdebug` (This tells the `ftn` / `cc` / `CC` compiler wrappers to link the library).
3. Re-link your program (e.g. by deleting your binary and running make).

Once re-linked, run your program with DDT, ensuring you enable the "*Memory Debugging*" option in the run dialog.

Memory Debugging Caveats

- The behavior of `ddt-memdebug` depends on the current programming environment. For this reason, you may encounter issues if you switch programming environments after `ddt-memdebug` has been loaded. To avoid this, please ensure that you unload `ddt-memdebug` before switching programming environments (you can then load it again).
- The Fortran `ALLOCATE` function cannot currently be wrapped when using the PGI compiler, so allocations will not be tracked, or protected.
- When using the Cray compiler, some libraries are compiled in such a way that DDT can not collect a backtrace when allocating memory. In this case, DDT can only show the location (rather than the

For additional information on memory debugging, see the Forge User Guide (https://web.archive.org/web/20190219213055/https://developer.arm.com/docs/101136/latest/arm-forge/introduction) and/or or how to fix memory leaks with DDT Leak Reports (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/tutorials/fixing-memory-leaks-with-ddt-leak-reports/).

Debugging scalar/non-MPI programs (Cray systems)

Launching a debug session on the Cray systems requires the program be linked with the Cray PMI library. (This happens automatically when linking with MPI.) In addition, DDT must be told not to run your program to the `MPI_Init` function (as it won't be called).

If you are using the Cray compiler wrappers, you can load the `ddt-non-mpi` module (before linking your program) to include the PMI library.

The same module should also be loaded prior to running `ddt` (to tell DDT not to attempt to run to `MPI_Init` during initialization).

Finally, enable the "MPI" option in the DDT run dialog. This will ensure DDT launches your program with `aprun`.

Using the ddt-non-mpi module with the DDT Remote Client

When using the Forge Remote Client (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/), we can't load the `ddt-non-mpi module` in to the client itself. Instead we have three options:

1. If using "Reverse Connect", load the module before launching `ddt --connect ...`

3 Load the module using the "remote script" mechanism while configuring your remote host in the DDT Remote Client. For convenience, you may specify the following file as the remote script: `/sw/sources/ddt/ddt-non-mpi.sh`.

Optimization & Profiling

Optimization Guide for AMD64 Processors

AMD offers guidelines specifically for serial code optimization on the AMD Opteron processors. Please see AMD's Developer Documentation (https://web.archive.org/web/20190219213055/http://developer.amd.com/resources/developer-guides-manuals/) site for whitepages and information on the latest generation of AMD processors.

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/file-systems/#spider-the-centerwide-lustre-file-system) page.

CrayPat

performance analysis tool for evaluating program execution on Cray systems. CrayPat consists of three main components:

- `pat_build` – used to instrument the program for analysis
- `pat_report` – a text report generator that can be used to explore data gathered by instrumented program execution
- `Apprentice2` – a graphical analysis tool that can be used in addition to pat_report to explore and visualize the data gathered by instrumented program execution

> **Note:** Details of these components can be found in the `pat_build`, `pat_report`, and `app2` man pages made available by loading the perftools-base module.

The standard workflow for program profiling with CrayPat is as follows:

1. Load the perftools-base and perftools modules
2. Build your application as normal
3. Instrument the application with pat_build
4. Execute the instrumented application
5. View the performance data generated in Step 4 with pat_report and Apprentice2

The following steps will guide you through performing a basic analysis with CrayPat and Apprentice2.

Begin with a fully debugged and executable program. Since CrayPat is a performance analysis tool, not a debugger, the targeted program must be capable of running to completion, or intentional termination.

Step 1: Load the appropriate modules

```
module load perftools-base
module load perftools
```

The perftools-base module must be loaded before the perftools module. Attempting to load perftools first will result in the following message:

```
------------------------------------------------------------------------------
Error: The Perftools module is available only after the perftools-base module
is loaded.

The Perftools-base module:
    - Provides access to Perftools man pages, Reveal and Cray Apprentice2
    - Does not alter compiling or program behavior
    - Makes the following instrumentation modules available:

perftools               - full support, including pat_build and pat_report
perftools-lite          - default CrayPat-lite profile
perftools-lite-events   - CrayPat-lite event profile
perftools-lite-gpu      - CrayPat-lite gpu kernel and data movement
perftools-lite-loops    - CrayPat-lite loop estimates (for Reveal)
perftools-lite-hbm      - CrayPat-lite memory bandwidth estimates (for Reveal)
------------------------------------------------------------------------------
```

Step 2: Build your application

Now that CrayPat is loaded, compile the program using the recommended Cray compiler wrappers (cc, CC, ftn) and appropriate flags to preserve all .o (and .a, if any) files created at compile time. CrayPat requires access to these object files (and archive files, if any exist).

For example, if working with a Fortran program, use commands similar to

```
ftn -c my_program.f
ftn -o my_program my_program.o
```

in order to retain object files.

Similarly, if you are using a makefile, perform a

> **Note:** By default, a copy of the build's .o files will be placed in /ccs/home/$USER/.craypat. This may increase your home directory usage to the quota limit. To change the location of the .craypat directory, set the `PAT_LD_OBJECT_TMPDIR` environment variable. For example, `export PAT_LD_OBJECT_TMPDIR=/tmp/work/$USER`

Step 3: Instrument the application with pat_build

If desired, use

```
pat_build -O apa my_program
```

where `-O apa` is a special argument for Automatic Program Analysis. `apa` instrumented programs will produce a .apa file at execution, which includes recommended parameters for improving the analysis.

This will produce an instrumented executable `my_program+pat` .

Step 4: Execute the instrumented program

Executing the instrumented program will generate and collect performance analysis data, written to one or more data files. On a Cray XT/XK Series CNL system, programs are executed using the aprun command (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/running-jobs/#using-the-aprun-command).

```
aprun -n <numproc> my_program+pat
```

This will produce on completion (or termination) the data file `my_program+pat+PID-nodesdt.xf` , which contains basic asynchronously derived program profiling data.

Step 5: View the performance data with pat_report or Apprentice2

```
pat_report -T -o report1.txt my_program+pat+PID-nodesdt.xf
```

will produce:

- a sampling-based text report, `report1.txt`
- a .ap2 file, `my_program+pat+PID-nodesdt.ap2` , which contains both the report data and the associated mapping from addresses to functions and source code line numbers. This file can be opened for viewing with Apprentice2.
- a .apa file, `my_program+pat+PID-nodesdt.apa` , which contains the pat_build arguments for further analysis.

If using the Automatic Program Analysis parameters is desired, open the .apa file in a text editor and make changes as appropriate. Any pat_build options may be added to this file (Most commonly used: `-g mpi` , `-g blacs` , `-g blas` , `-g io` , `-g lapack` , `-g lustre` , `-g math` , `-g scalapack` , `-g stdio` , `-g sysio` , `-g system`). Re-instrument the program with pat_build, run the new executable, and generate another set of data files.

> **Note:** When re-building the program for Automatic Program Analysis, it is not necessary to specify the program name.
>
> ```
> pat_build -O my_program+pat+PID-nodesdt.apa
> ```
>
> is sufficient, since the .apa file contains the program name.

To view results graphically, first ensure X-forwarding is enabled (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/connecting/#x11-forwarding), then use

A GUI should appear to interact with the collected performance data.

Simple GPU Profiling Example

A simple GPU profiling example could be preformed as follows:

With PrgEnv-Cray:

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

With PrgEnv other than Cray:

```
$ module load cudatoolkit
$ module load perftools-base 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
```

CrayPAT Temporary Files

When building code with CrayPat in your `$MEMBERWORK/[projid]` or `$PROJWORK/[projid]` area, a copy of the build's `.o` files will, by default, be placed in `/ccs/home/$USER/.craypat`.

This may increase your User Home directory usage above quota. The `PAT_LD_OBJECT_TMPDIR` environment variable can be used to control the location of the `.craypat` directory. For example:

```
  export PAT_LD_OBJECT_TMPDIR $MEMBERWORK/[projid]
```

Compiling Mixed GPU and CPU Code (/web/20190219213055/https://www.olcf.ornl.gov/tutorials/compiling-mixed-gpu-and-cpu-code/).

NVPROF

NVIDIA's command-line profiler, `nvprof`, provides profiling for CUDA codes. No extra compiling steps are required to use `nvprof`. 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, graphical analysis.

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 (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/connecting/#x11-forwarding).

> The `aprun -b` flag is currently required to use NVPROF, this requires that your executable reside on a compute node visible filesystem.

```
$ 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 **h**ost and **p**rocess.

```
$ aprun -b -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:

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

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

For information on how to view the output in the NVIDIA visual profiler, please see the NVIDIA documentation (https://web.archive.org/web/20190219213055/http://docs.nvidia.com/cuda/profiler-users-guide/index.html#import-session).

Additional NVPROF Resources

The **nvprof user guide** is available on the NVIDIA Developer Documentation Site (https://web.archive.org/web/20190219213055/http://docs.nvidia.com/cuda/profiler-users-guide/#nvprof-overview) and provides comprehensive coverage of the profiler's usage and features.

Score-P

The Score-P (https://web.archive.org/web/20190219213055/http://score-p.org/) measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications. Score-P supports analyzing C, C++ and Fortran applications that make use of multi-processing (MPI, SHMEM), thread parallelism (OpenMP, PThreads) and accelerators (CUDA, OpenCL, OpenACC) and combinations.

For detailed information about using Score-P on Titan and the builds available, please see the Score-P Software Page (/web/20190219213055/https://www.olcf.ornl.gov/software_package/score-p/).

Vampir

run including the use of programming paradigms like MPI, OpenMP, PThreads, CUDA, OpenCL and OpenACC. It also incorporates file I/O, hardware performance counters and other performance data sources. Various interactive displays offer detailed insight into the performance behavior of the analyzed application. Vampir's scalable analysis server and visualization engine enable interactive navigation of large amounts of performance data. Score-P (https://web.archive.org/web/20190219213055/https://olcf.ornl.gov/software_package/score-p) and TAU (https://web.archive.org/web/20190219213055/https://www.olcf.ornl.gov/software_package/tau) generate OTF2 trace files for Vampir to visualize.

For detailed information about using Vampir on Titan and the builds available, please see the Vampir Software Page (/web/20190219213055/https://www.olcf.ornl.gov/software_package/vampir/).

TAU

TAU Performance System is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java, and Python. Generated traces can be viewed in the included Paraprof GUI or displayed in Vampir.

Simple GPU Profiling

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
```

```
$ paraprof
```

Other TAU uses

```
module load tau
```

This command sets the TAUROOT environment variable on OLCF platforms and puts the TAU compiler wrappers in your PATH.

Automatic instrumentation when compiling with the C TAU wrapper:

```
>  export TAU_MAKEFILE=${TAU_LIB}/Makefile.tau-papi-mpi-pdt-openmp-opari-pgi
>  export TAU_MAKEFILE=${TAU_LIB}/Makefile.tau-papi-mpi-pthread-pdt-pgi
>  tau_f90.sh test.f
```

Debug: Parsing with PDT Parser

```
> /sw/xt/tau/2.17/cnl2.0+pgi7.0.7/pdtoolkit-3.12//craycnl/bin/f95parse mpi_example8.f
-I/sw/xt/tau/2.17/cnl2.0+pgi7.0.7/tau-2.17/include -I/opt/xt-mpt/default/mpich2-64/P/incl
ude
```

Debug: Instrumenting with TAU

```
> /sw/xt/tau/2.17/cnl2.0+pgi7.0.7/tau-2.17/craycnl/bin/tau_instrumentor mpi_example8.pdb
mpi_example8.f -o
mpi_example8.inst.f
```

Debug: Compiling (Individually) with Instrumented Code

```
> ftn -I. -c mpi_example8.inst.f -I/sw/xt/tau/2.17/cnl2.0+pgi7.0.7/tau-2.17/include
-I/opt/xt-mpt/default/mpich2-64/P/include -o mpi_example8.o
/opt/xt-pe/2.0.33/bin/snos64/ftn: INFO: linux target is being used
```

```
> ftn mpi_example8.o -L/opt/xt-mpt/default/mpich2-64/P/lib -L/sw/xt/tau/2.17/cnl2.0+pgi7.
0.7/tau-2.17/craycnl/lib
-lTauMpi-mpi-pdt -lrt -lmpichcxx -lmpich -lrt -L/sw/xt/tau/2.17/cnl2.0+pgi7.0.7/tau-2.17/
craycnl/lib -ltau-mpi-pdt
-L/opt/pgi/7.0.7/linux86-64/7.0/bin/../lib -lstd -lC -lpgc -o a.out
/opt/xt-pe/2.0.33/bin/snos64/ftn: INFO: linux target is being used
```

Debug: cleaning inst file

```
> /bin/rm -f mpi_example8.inst.f
```

Debug: cleaning PDB file

```
> /bin/rm -f mpi_example8.pdb
> aprun -n 4 ./a.out
> ls prof*
profile.0.0.0  profile.1.0.0  profile.2.0.0  profile.3.0.0
```

To visualize the profile with the Paraprof tool:

```
> module load java-jre
> module load tau   #if not loaded
> paraprof
```

Additional Tau Resources

The TAU documentation website (https://web.archive.org/web/20190219213055/http://www.cs.uoregon.edu/Research/tau/docs.php) contains a complete User Guide, Reference Guide, and even video tutorials.

Arm MAP

of the Arm Forge suite, with DDT) is a profiler for parallel, multithreaded or single threaded C, C++, Fortran and F90 codes. It provides in depth analysis and bottleneck pinpointing to the source line. Unlike most profilers, it's designed to be able to profile pthreads, OpenMP or MPI for parallel and threaded code. MAP aims to be simple to use – there's no need to instrument each source file, or configure.

Linking your program with the MAP Sampler (for Cray systems)

In order to collect information about your program, you must link your program with the MAP sampling libraries. When using shared libraries on, MAP can do this automatically at runtime.

On Cray systems, the `map-static-link` and `map-dynamic-link` modules can help with this.

1. `module load forge`
2. `module load map-link-static # or map-link-dynamic`
3. Re-compile or re-link your program.

Do I need to recompile?

There's no need to instrument your code with Arm MAP, so there's no strict requirement to recompile. However, if your binary wasn't compiled with the `-g` compiler flag, MAP won't be able to show you source-line information, so recompiling would be beneficial.

> Note: If using the Cray compiler, you may wish to use `-G2` instead of `-g`. This will prevent the compiler from disabling most optimizations, which could affect runtime performance.

Generating a MAP output file

moving from an existing launch configuration.

MAP profiles are small in size, and there's generally no configuration required other than your existing aprun command line.

To generate a profile using MAP, take an existing queue submission script and modify to include the following:

```
source $MODULESHOME/init/bash # May already be included if using modules
module load forge
```

And then add a prefix your aprun command so that:

```
aprun -n 128 -N 8 ./myapp a b c
```

would become:

```
map --profile aprun -n 128 -N 8 ./myapp a b c
```

Once your job has completed running, the program's working directory should contain a timestamped `.map` file such as `myapp_1p_1t_2016-01-01_12-00.map`.

Profiling a subset of your application

To profile only a subset of your application, you can either use the `--start-after=TIME` and its command line options (see `map --help` for more information), or use the API to have your code tell MAP when to start and stop sampling, as detailed here (https://web.archive.org/web/20190219213055/https://www.allinea.com/blog/201407/profiling-subset-your-code-allinea-map).

Viewing a MAP profile

```
map ./myapp_1p_1t_2016-01-01_12-00.map
```

(The above will require a SSH connection with X11 forwarding (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/connecting/#x11-forwarding), or other remote graphics setup.)

An alternative that provides a local, native GUI (for Linux, OS X, or Windows) is to install the Arm Forge Remote Client (https://web.archive.org/web/20190219213055/https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge) on your local machine. This client is able to load and view profiles locally (useful when working offline), or remotely (which avoids the need to copy the profile data and corresponding source code to your local machine).

The remote client can be used for both Arm DDT and Arm MAP. For more information on how to install and configure the remote client, see the remote client setup page (/web/20190219213055/https://www.olcf.ornl.gov/for-users/system-user-guides/titan/debugging/#ddt-remote-client-setup-and-usage).

For more information see the Arm Forge user guide (https://web.archive.org/web/20190219213055/https://developer.arm.com/docs/101136/latest) (also available via the “Help” menu in the MAP GUI).

Additional Arm MAP resources

- Arm MAP Website (https://web.archive.org/web/20190219213055/https://www.arm.com/products/development-tools/hpc-tools/cross-platform/forge/map)

forge/introduction)

If you have questions regarding the documentation above, please contact OLCF Support at help@olcf.ornl.gov (https://web.archive.org/web/20190219213055/mailto:help@olcf.ornl.gov).

Last updated January 8, 2019