How To Use infer Test Cluster

General Information

Members of VT who have received accounts on infer have access to the infer login nodes and the infer cluster job queues. Typical workflow involves logging into a login node for accessing files and applications; creating code and compiling programs; submitting jobs to a job queue to be run on compute nodes; and, displaying information about jobs and controlling submitted jobs.

Logging In

The infer cluster has one login node. The login node is a 32-core Skylake-SP Gold 6130 2.10GHz (3.70GHz Turbo) dual processor (16 cores per processor) node with 192 GB of memory, 480 GB internal solid state drive mirrored pair (including 350 GB in /localscratch) and Mellanox EDR InfiniBand Adapter. You can log in to the infer cluster login node via ssh to i**nfer1.arc.vt.edu** using your VT PID and password.

Off-campus users do not have direct ssh access and must use the VT VPN. See the following link for information on registering for and using the VT VPN: https://vt4help.service-now.com/sp?id=kb_article&sys_id=c0d7075cdb5df600294ffa38bf961918

The login nodes are to be used for accessing files, compiling codes, and submitting jobs to the queue and other queue interaction. The login nodes are shared resources used by multiple users; therefore, user programs should normally **not** be run on the login nodes. Post-processing tasks which may consume many cores or much memory or time should also be performed in the context of a job and not on a login node.

Accessing Files

Shared Storage

User Files

Each user has a home directory under the /home filesystem with a name that follows the convention of /home/<username>. The /home filesystem is shared among all infer login and compute nodes, so any files created under your home on the login nodes are accessible under your home to jobs on the compute nodes. This filesystem is served by a storage system which is local to the infer cluster, which is NOT available on other VT ARC HPC clusters and HPC systems.

Shared Storage Quota Limits

User Quotas

Each user has a user storage quota which limits the amount of data a user may store within their home directory and any group directories to which they have access. The default user quota for new users is 1 TB soft and 1.1 TB hard.

Local Storage

Temporary Files

Each login node and compute node has a local scratch space mounted at /localscratch for the storage of temporary files on each individual node. See the information above under Logging In for specifications of the size of /localscratch on login nodes, and see the information below under Available Job Queues for specifications of the size of /localscratch on compute nodes.

Other Shared Storage

For convenience, this test cluster mounts the Qumulo storage /home filesystem which is available on other VT ARC HPC clusters and HPC systems on the infer cluster login and compute nodes at /qumulo_home. Please note that the infer cluster only uses Enterprise Directory for LDAP group lookups and, therefore, LDAP groups managed by the ARC LDAP server which were used on Qumulo files will not resolve to a group name on infer.

Accessing Available Applications and Compilers

Many common scientific applications and compiler applications are centrally installed and maintained and are made available to all users through the Lmod modules environment. Using the Lmod modules environment to access and search for these applications is discussed below.

Modules Environment

Access to centrally installed applications is enabled via the module command interface to the Lmod modules package. The Lmod modules package provides for the dynamic modification of the user's environment for an application or set of applications. You MUST use the module command to add the module for an application to your environment in order to use that application. The module command can be used at the command line and within job scripts. (Job scripts are discussed in a later section.)

The Most Common module Subcommands

To see what modules are loaded in your environment:

```
module list
```

To see what modules are available:

```
module avail
```

To add (load) a module to your environment (module load is an equivalent subcommand):

```
module add <module name>
```

Example:

```
module add mvapich2/gcc/64
```

To remove (unload) a module from your environment (module unload is an equivalent subcommand):

```
module rm <module name>
```

Example:

```
module rm mvapich2/gcc/64
```

Other Useful module Subcommands

To unload all loaded modulefiles:

```
module purge
```

To reset all loaded modulefiles to defaults:

```
module reset
```

To list the full path of the modulefile(s), and environment changes the modulefile(s) will make:

```
module show <module name>
```

Example:

```
module show mvapich2/gcc/64
```

An online manual page is available for the module command:

```
man module
```

Searching the List of Available Application Modules

To search for an application module within the output of the module list subcommand, you can use the command:

```
module avail 2>&1 | grep -i <search string>
```

Example:

```
module avail 2>&1 | grep -i mvapich2
```

NOTE: We are using "2>&1" here to redirect standard error to standard output because the module avail command outputs to standard error but we are searching standard output with grep.

Modules Loaded by Default

Modules Loaded by Default on Login Nodes

The following modules are loaded by default for all users on the login nodes:

shared

apps

gcc

slurm

site/infer/easybuild/setup

The slurm module is necessary for queue submission and should not be manually removed.

The site/infer/easybuild/setup module adds access to applications that are installed and maintained with the EasyBuild software build and installation framework.

NOTE: To reload the set of default user modules, removing all non-default modules, use the command:

```
module reset
```

Modules Loaded by Default on Compute Nodes

The following modules are loaded by default for all users on the compute nodes:

shared

apps

gcc

slurm

site/infer/easybuild/setup

gcc Compiler Suite

The gcc compiler collection (gcc, gfortran, g++, etc.) included with the operating system on the login and compute nodes is older than the various gcc compiler collections available via loadable modules, and much older than the gcc compiler collection that is loaded by default for all users. Manually removing the gcc module form your environment will switch you back to the older gcc compiler collection included with the operating system. Manually loading a different gcc compiler collection available via loadable modules will take precedence over the gcc compiler collection included with the operating system and any previously loaded gcc compiler collections available via loadable modules.

Applications Installed and Maintained with EasyBuild

VT ARC is transitioning to the use of EasyBuild for installing and maintaining most centrally installed applications. EasyBuild is an open source software build and installation framework that allows efficient management of scientific software on High Performance Computing (HPC) systems.

IMPORTANT NOTE:

Loading an EasyBuild module loads several applications that form a toolchain used to build the requested application. As such, care must be taken to ensure that some of the applications being loaded as dependencies by the requested EasyBuild application module do not conflict with any non-EasyBuild applications (such as those loadable from under /cm/shared/modulefiles and /apps/modulefiles) which may have been previously loaded into your environment.

An EasyBuild application of note which is loaded by other EasyBuild applications is the EasyBuild GCC module, which **may create a conflict** as it tends to be an older gcc compiler collection version than the one being loaded by default for all users. **If you rely on the gcc compiler collection version being loaded by default for all users, then you may need to unload and reload the default gcc compiler collection after loading an EasyBuild module.** F or example:

infer1:~> which gcc

/cm/local/apps/gcc/9.2.0/bin/gcc

infer1:~> module load Python

infer1:~> which gcc

/apps/easybuild/software/infer-skylake/GCCcore/8.3.0/bin/gcc

infer1:~> module unload gcc

infer1:~> module load gcc

infer1:~> which gcc

/cm/local/apps/gcc/8.2.0/bin/gcc

One EasyBuild application of particular concern, however, is OpenMPI; it will conflict with any previously loaded MPI implementation such as the mvapich2 and openmpi modules loadable from under /cm/shared/modulefiles, which are built to work with our Slurm and InfiniBand implementations. You will therefore need to unload and reload any required mvapich2 or openmpi modules previously loaded from under /cm/shared /modulefiles after loading an EasyBuild application. For example:

infer1:~> module load mvapich2/gcc/64

infer1:~> which mpicc

/cm/shared/apps/mvapich2/gcc/64/2.3.2/bin/mpicc

infer1:~> which mpiexec

/cm/shared/apps/mvapich2/gcc/64/2.3.2/bin/mpiexec

infer1:~> module load networkx

infer1:~> which mpicc

/apps/easybuild/software/infer-skylake/OpenMPI/3.1.4-GCC-8.3.0/bin/mpicc <= This is the wrong one!

infer1:~> which mpiexec

/apps/easybuild/software/infer-skylake/OpenMPI/3.1.4-GCC-8.3.0/bin/mpiexe <= This is the wrong one!

infer1:~> module unload mvapich2/gcc

infer1:~> module load mvapich2/gcc/64

infer1:~> which mpicc

/cm/shared/apps/mvapich2/gcc/64/2.3.2/bin/mpicc

infer1:~> which mpiexec

/cm/shared/apps/mvapich2/gcc/64/2.3.2/bin/mpiexec

Building and Installing Your Own Applications with EasyBuild

Besides having access to software maintained by VT ARC under `/apps/easybuild`, users can install additional software via EasyBuild under their home directory in the `~/easybuild/` directory. An additional environment module sets up all required paths.

To get started, load the EasyBuild module:

```
module load EasyBuild
```

Now you can run the `eb` command to search and install new software.

Example:

- search for NetworkX: `eb -S networkx`
- install a specific NetworkX version: `eb networkx-2.4-foss-2019b-Python-3.7.4.eb --robot`

More information on the VT ARC implementation of EasyBuild can be found at: https://code.vt.edu/arc-se/easybuild ***** INTERNAL NOTE: THIS REPO STILL HAS REFERENCES TO BI, NEEDS TO BE UPDATE FOR ARC *****

Submitting Jobs

Access to cluster compute nodes is enabled via jobs submitted to the Slurm workload manager, which includes both a resource manager and a job scheduler. Slurm manages: (1) the resources of compute **nodes**; (2) **partitions**, which group compute nodes into logical sets; (3) **jobs**, which are allocations of resources assigned to a user for a specified amount of time; and, (4) **job steps**, which are sets of (possibly parallel) tasks within a job. The partitions can be considered job queues, each of which may have an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. [Source: http://slurm.schedmd.com/quickstart.html] For an overview of Slurm usage and commands, see: Slurm Quick Reference. ***** INTERNAL NOTE: THIS LINK STILL POINTS TO A BI WIKI PAGE, NEED TO MAKE A NEW TARGET *****

Available Job Queues (a.k.a., Partitions)

Queue / Partition	Allowed Account Group(s)	Quality of Service	Node Allocation Policy*	Number of Compute Nodes	Compute Node Specifications
infer_q	arc.infer	Standard	ExclusiveUser=YES	18	**32-core** Skylake-SP Gold 6130 2.10GHz (3.70GHz Turbo) dual processor (16 cores per processor) node with **192 GB** of memory, 240 GB internal M.2 solid state drive, 480 GB internal solid state drive striped pair (providing 900 GB in /localscratch), Mellanox EDR InfiniBand Adapter, and **NVIDIA Turing TU104** [Tesla T4] GPU Accelerator with compute capability 7.5 => Hostnames: inf001 - inf018

gpu_q	arc.infer	Standard	ExclusiveUser=YES	18	**32-core** Skylake-SP Gold 6130 2.10GHz (3.70GHz Turbo) dual processor (16 cores per processor) node with **192 GB** of memory, 240 GB internal M.2 solid state drive, 480 GB internal solid state drive striped pair (providing 900 GB in /localscratch), Mellanox EDR InfiniBand Adapter, and **NVIDIA Turing TU104** [Tesla T4] GPU Accelerator with compute capability 7.5 => Hostnames: inf001 - inf018

*Node Allocation Policy: If ExclusiveUser is set to YES (the Slurm equivalent of a Moab scheduler SINGLEUSER node access policy), then nodes will be exclusively allocated to users. Multiple jobs may be run for the same user on a node, i.e, a user may pack jobs on nodes, but only one user can be active at a time on a node. If ExclusiveUser is set to NO (the Slurm equivalent of a Moab scheduler SHARED node access policy), then multiple users can be active at a time on a node. NOTE: A user may request that dedicated nodes be allocated to a single job (equivalent to a Moab scheduler SINGLEJOB node access policy) by specifying the --exclusive option in their job submission; this option is exemplified below.

Selecting a Queue

The **infer_q** is the general production queue where most jobs should be submitted. The **gpu_q** is available for users who need access to nodes with GPUs.

Displaying Information About Queues (Partitions)

The **sinfo** command reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options. Useful examples are shown below.

The following example shows the default sinfo output, which includes node states and therefore lists down or drained (i.e., offline) nodes on separate lines.

```
$ sinfo

PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
infer_q*     up   infinite     18   idle inf[001-018]
gpu_q        up   infinite     18   idle inf[001-018]
```

The following example shows custom output formatting to display the number of CPUs by state in the format "allocated/idle/other/total".

```
$ sinfo -o "%12P %.5a %.6D %C"

PARTITION    AVAIL  NODES CPUS(A/I/O/T)
infer_q*        up     18 0/576/0/576
gpu_q           up     18 0/576/0/576
```

Types of Jobs

Normally one submits **batch jobs** using the sbatch command to submit job scripts. However, you may occasionally have a need for an interactive shell on a compute node, such as for application testing; if so, you may request **interactive shell jobs** on compute nodes using interactive srun commands. Both methods are discussed below.

Submitting Batch Jobs

Batch job submission is done by submitting a job script with the **sbatch** command. Job scripts specify run options and run applications, which may be serial applications (a single process running at any one time) or parallel applications (multiple coordinated processes running in parallel, typically MPI applications). Examples of both types of applications are shown below.

Submitting a Serial Application

Example Serial Job Script

```
#!/bin/bash

# Example Slurm sbatch script for simple serial jobs in the infer cluster
# infer_q partition.

#
# SLURM JOB SCRIPT OPTIONS:
#
# The #SBATCH lines set various SLURM directive parameters for the job.
#
# The "#" sign in front of "SLURM" at the beginning of a sbatch script line is
# not a comment; Slurm interprets lines beginning with "#SBATCH" as denoting a
# job option directive.  All other instances of lines beginning with a "#" sign
# (including, e.g., "##SBATCH" and "# SBATCH") are comments as normal.
#
# It is important to keep all #SBATCH lines together near the top of the
# sbatch script.  No commands or variable settings should occur until
# after the #SBATCH lines.
```

```
# SPECIFYING THE ACCOUNT
#SBATCH -A myAccount

#
# REQUESTING RESOURCES:
#
# You will need to edit the --time resource limit line
# and the --nodes and --ntasks-per-node resource limits line
# to suit the requirements of your job.
#
# NOTE: --ntasks-per-node is equivalent to number of cores per node (and
#       therefore comparable to ppn in Torque), unless also specifying
#       a --cpus-per-task setting greater than 1.
#
# You may optionally request whole nodes by uncommenting the --exclusive line.
#
# You may optionally request memory required per node or memory required per
# allocated CPU by uncommenting and editing the --mem or --mem-per-cpu line.
#
# NOTE: The --mem parameter would generally be used if whole nodes are
#       allocated to jobs, while the --mem-per-cpu parameter would generally
#       be used if individual processors are allocated to jobs.


# Set the time, which is the maximum time your job can run in HH:MM:SS

#SBATCH --time=00:05:00


# Set the number of nodes, and the number of tasks per node (up to 32 per node)
#
# NOTE: The maximum number of tasks per node that can be requested will be
#       affected by the cpus per task setting if this setting is specified
#       with --cpus-per-task greater than 1.

#SBATCH --nodes=1 --ntasks-per-node=1


# If needed, request whole nodes by uncommenting the "#SBATCH --exclusive" line
# below

##SBATCH --exclusive


# If needed, request memory required per node or memory required per allocated
# CPU, in MegaBytes
#
# NOTE: Memory required per node would generally be used if whole nodes are
#       allocated to jobs, while memory required per allocated CPU would
#       generally be used if individual processors are allocated to jobs.
#
# NOTE: The commented out examples below show the currently configured default
#       values.

##SBATCH --mem=182272
##SBATCH --mem-per-cpu=5696


# Set the partition to submit to (a partition is equivalent to a queue)

#SBATCH -p infer_q


# Optional: If modules are needed, source modules environment
#
# NOTE: This step is normally only needed for users whose default shell is
#       csh or tcsh.
#
# (Uncomment the following line if you wish to use it.)
# . /etc/profile.d/modules.sh

# Optional: Reset all loaded modulefiles to the normal defaults
# (Uncomment the following line if you wish to use it.)
# module reset

# Load any modules you require:
#
# NOTE: The following example module "null" does nothing; replace the module
#       name with one you wish to load.
#
# (Uncomment the following line if you wish to use it.)
# module load null
```

```
# Below here enter the commands to start your job

hostname
```

Submitting the Example Serial Job Script

The following example uses the name "hostname.sbsatch" as the file name of our example serial job script shown above.

```
$ sbatch hostname.sbatch

Submitted batch job 43

$ cat slurm-43.out

inf001
```

Compiling and Submitting a Parallel MPI Application

Example MPI Application

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char ** argv) {
  int size,rank,namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Get_processor_name(processor_name, &namelen);
  printf("Hello MPI! Process %d of %d on %s\n", rank, size, processor_name);
  MPI_Finalize();
}
```

Compiling the Example MPI Application

The following example uses the name "hello_mpi.c" as the file name of our example MPI application shown above and creates a compiled program named "hello_mvapich2_gcc."

```
$ module load mvapich2/gcc/64

$ mpicc -o hello_mvapich2_gcc hello_mpi.c
```

Example MPI Job Script

```
#!/bin/bash

# Example Slurm sbatch script for parallel MPI jobs in the infer cluster
# infer_q partition.

#
# SLURM JOB SCRIPT OPTIONS:
#
# The #SBATCH lines set various SLURM directive parameters for the job.
#
# The "#" sign in front of "SLURM" at the beginning of a sbatch script line is
# not a comment; Slurm interprets lines beginning with "#SBATCH" as denoting a
# job option directive.  All other instances of lines beginning with a "#" sign
# (including, e.g., "##SBATCH" and "# SBATCH") are comments as normal.
#
# It is important to keep all #SBATCH lines together near the top of the
# sbatch script.  No commands or variable settings should occur until
# after the #SBATCH lines.


# SPECIFYING THE ACCOUNT
#SBATCH -A myAccount

#
# REQUESTING RESOURCES:
#
# You will need to edit the --time resource limit line
# and the --nodes and --ntasks-per-node resource limits line
# to suit the requirements of your job.
#
# NOTE: --ntasks-per-node is equivalent to number of cores per node (and
#       therefore comparable to ppn in Torque), unless also specifying
#       a --cpus-per-task setting greater than 1.
#
```

```
# You may optionally request whole nodes by uncommenting the --exclusive line.
#
# You may optionally request memory required per node or memory required per
# allocated CPU by uncommenting and editing the --mem or --mem-per-cpu line.
#
# NOTE: The --mem parameter would generally be used if whole nodes are
#       allocated to jobs, while the --mem-per-cpu parameter would generally
#       be used if individual processors are allocated to jobs.


# Set the time, which is the maximum time your job can run in HH:MM:SS

#SBATCH --time=00:05:00


# Set the number of nodes, and the number of tasks per node (up to 32 per node)
#
# NOTE: The maximum number of tasks per node that can be requested will be
#       affected by the cpus per task setting if this setting is specified
#       with --cpus-per-task greater than 1.

#SBATCH --nodes=2 --ntasks-per-node=2


# If needed, request whole nodes by uncommenting the "#SBATCH --exclusive" line
# below

##SBATCH --exclusive


# If needed, request memory required per node or memory required per allocated
# CPU, in MegaBytes
#
# NOTE: Memory required per node would generally be used if whole nodes are
#       allocated to jobs, while memory required per allocated CPU would
#       generally be used if individual processors are allocated to jobs.
#
# NOTE: The commented out examples below show the currently configured default
#       values.

##SBATCH --mem=182272
##SBATCH --mem-per-cpu=5696

# Set the partition to submit to (a partition is equivalent to a queue)

#SBATCH -p infer_q


# Optional: If modules are needed, source modules environment
#
# NOTE: This step is normally only needed for users whose default shell is
#       csh or tcsh.
#
# (Uncomment the following line if you wish to use it.)
# . /etc/profile.d/modules.sh

# Optional: Reset all loaded modulefiles to the normal defaults
# (Uncomment the following line if you wish to use it.)
# module reset

# Load any modules you require:
#
# NOTE: The following example module "null" does nothing; replace the module
#       name with one you wish to load.
#
# (Uncomment the following line if you wish to use it.)
# module load null
module load mvapich2/gcc/64

# Below here enter the commands to start your job

srun -n $SLURM_NTASKS --mpi=pmi2 ./hello_mvapich2_gcc
```

Submitting the Example MPI Job Script

The following example uses the name "hello_mvapich2_gcc.sbatch" as the file name of our example MPI job script shown above.

```
$ sbatch hello_mvapich2_gcc.sbatch

Submitted batch job 44

$ cat slurm-44.out

Hello MPI! Process 0 of 4 on inf001
Hello MPI! Process 1 of 4 on inf001
Hello MPI! Process 2 of 4 on inf002
Hello MPI! Process 3 of 4 on inf002
```

Compiling and Submitting an MPI Application Using Open MPI

Example MPI Application

This is the same example MPI application as appears above.

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char ** argv) {
  int size,rank,namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Get_processor_name(processor_name, &namelen);
  printf("Hello MPI! Process %d of %d on %s\n", rank, size, processor_name);
  MPI_Finalize();
}
```

Compiling the Example MPI Application Using Open MPI

The following example uses the name "hello_mpi.c" as the file name of our example MPI application shown above and creates a compiled program named "hello_openmpi_gcc."

```
$ module load openmpi/gcc/64

$ mpicc -o hello_openmpi_gcc hello_mpi.c
```

Example MPI Job Script Using Open MPI

```
#!/bin/bash

# Example Slurm sbatch script for parallel Open MPI jobs in the infer cluster
# infer_q partition.

#
# SLURM JOB SCRIPT OPTIONS:
#
# The #SBATCH lines set various SLURM directive parameters for the job.
#
# The "#" sign in front of "SLURM" at the beginning of a sbatch script line is
# not a comment; Slurm interprets lines beginning with "#SBATCH" as denoting a
# job option directive.  All other instances of lines beginning with a "#" sign
# (including, e.g., "##SBATCH" and "# SBATCH") are comments as normal.
#
# It is important to keep all #SBATCH lines together near the top of the
# sbatch script.  No commands or variable settings should occur until
# after the #SBATCH lines.


# SPECIFYING THE ACCOUNT
#SBATCH -A myAccount

#
# REQUESTING RESOURCES:
#
# You will need to edit the --time resource limit line
# and the --nodes and --ntasks-per-node resource limits line
# to suit the requirements of your job.
#
# NOTE: --ntasks-per-node is equivalent to number of cores per node (and
#       therefore comparable to ppn in Torque), unless also specifying
#       a --cpus-per-task setting greater than 1.
#
# You may optionally request whole nodes by uncommenting the --exclusive line.
#
# You may optionally request memory required per node or memory required per
# allocated CPU by uncommenting and editing the --mem or --mem-per-cpu line.
#
```

```
# NOTE: The --mem parameter would generally be used if whole nodes are
#       allocated to jobs, while the --mem-per-cpu parameter would generally
#       be used if individual processors are allocated to jobs.


# Set the time, which is the maximum time your job can run in HH:MM:SS

#SBATCH --time=00:05:00


# Set the number of nodes, and the number of tasks per node (up to 32 per node)
#
# NOTE: The maximum number of tasks per node that can be requested will be
#       affected by the cpus per task setting if this setting is specified
#       with --cpus-per-task greater than 1.

#SBATCH --nodes=2 --ntasks-per-node=2


# If needed, request whole nodes by uncommenting the "#SBATCH --exclusive" line
# below

##SBATCH --exclusive


# If needed, request memory required per node or memory required per allocated
# CPU, in MegaBytes
#
# NOTE: Memory required per node would generally be used if whole nodes are
#       allocated to jobs, while memory required per allocated CPU would
#       generally be used if individual processors are allocated to jobs.
#
# NOTE: The commented out examples below show the currently configured default
#       values.

##SBATCH --mem=182272
##SBATCH --mem-per-cpu=5696

# Set the partition to submit to (a partition is equivalent to a queue)

#SBATCH -p infer_q


# Optional: If modules are needed, source modules environment
#
# NOTE: This step is normally only needed for users whose default shell is
#       csh or tcsh.
#
# (Uncomment the following line if you wish to use it.)
# . /etc/profile.d/modules.sh

# Optional: Reset all loaded modulefiles to the normal defaults
# (Uncomment the following line if you wish to use it.)
# module reset

# Load any modules you require:
#
# NOTE: The following example module "null" does nothing; replace the module
#       name with one you wish to load.
#
# (Uncomment the following line if you wish to use it.)
# module load null
module load openmpi/gcc/64


# Below here enter the commands to start your job

mpiexec -n $SLURM_NTASKS ./hello_openmpi_gcc
```

Submitting the Example MPI Job Script

The following example uses the name "hello_openpmi_gcc.sbatch" as the file name of our example MPI job script shown above.

```
$ sbatch hello_openmpi_gcc.sbatch

Submitted batch job 45

$ cat slurm-45.out

Hello MPI! Process 1 of 4 on inf001
Hello MPI! Process 0 of 4 on inf001
Hello MPI! Process 2 of 4 on inf002
Hello MPI! Process 3 of 4 on inf002
```

Running Muiltiple Processes in Parallel from a Batch Job

Example Job Script for Multiple Serial Processes Run in Parallel

The following example job script shows how more than one serial process can be run in parallel as parallel job steps. Each job step will be executed as processors become available for their dedicated use. In this example, each srun runs a different program, and both sruns will run at the same time since they each request ntasks=1 and the batch script requests ntasks=2.

```
#!/bin/bash

#SBATCH -A myAccount


#SBATCH --time=00:05:00

#SBATCH --ntasks=2

#SBATCH -p infer_q


# Below here enter the commands to start your job

srun --ntasks=1 --exclusive my_serial_program_1 &

srun --ntasks=1 --exclusive my_serial_program_2 &

wait
```

NOTE: Running multiple processes with identical parameters in parallel from a batch job may be accomplished by submitting a job array, which is discussed further below.

Example Job Script for Multiple MPI Processes Run in Parallel

The following example job script shows how more than one MPI process can be run in parallel as parallel job steps. Each job step will be executed as processors become available for their dedicated use. In this example, both sruns will run at the same time and will each spread their MPI tasks evenly across two nodes since they each request ntasks=4 and the batch script requests --nodes=2 --ntasks-per-node=4 (for a total of eight tasks across two nodes).

```
#!/bin/bash

#SBATCH -A myAccount


#SBATCH --time=00:05:00

#SBATCH --nodes=2 --ntasks-per-node=4

#SBATCH -p infer_q


module load mvapich2/gcc/64


# Below here enter the commands to start your job

srun --ntasks=4 --exclusive --mpi=pmi2 ./hello_mvapich2_gcc &

srun  --ntasks=4 --exclusive --mpi=pmi2 ./hello_mvapich2_gcc &

wait
```

Useful Queue Directive Options

Naming the Job

You may specify a name for the job allocation. The specified name will appear along with the job id number when querying running jobs on the system. The default is the name of the batch script, or just "sbatch" if the script is read on sbatch's standard input.

Directive Description	Specified As
Name the job <jobname>	#SBATCH -J <jobname>

Example:

```
#SBATCH -J hello_mvapich2_gcc
```

Naming Standard Output and Standard Error Files

By default both standard output and standard error are directed to a file of the name "slurm-%j.out", where the "%j" is replaced with the job allocation number. For example, for a job with job number 9626, this file will be named: slurm-9626.out

The following directives can be used to set the name of the output file, and to name and direct standard error to a separate error file:

Directive Description	Specified As
Set output log name to <jobname.log>*	#SBATCH -o <jobname.log>
Set error log name to <jobname.err>*	#SBATCH -e <jobname.err>

Examples:

```
#SBATCH -o hello_mvapich2_gcc-%j.out
```

```
#SBATCH -e hello_mvapich2_gcc-%j.err
```

Receiving E-Mail Notifications of Job Events

You may use the --mail-user=<user> directive together with the --mail-type=<type> directive to receive e-mail notifications when certain job event types occur.

Directive Description	Specified As
Mail <user@address>	#SBATCH --mail-user <user@address>
Mail on events of a specified <type>	#SBATCH --mail-type=<type>

Valid <type> values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit), TIME_LIMIT_50 (reached 50 percent of time limit) and ARRAY_TASKS (send emails for each array task). Multiple type values may be specified in a comma separated list. Unless the ARRAY_TASKS option is specified, mail notifications on job BEGIN, END and FAIL apply to a job array as a whole rather than generating individual email messages for each task in the job array.

Example of requesting notification of job END:

```
#SBATCH --mail-user joe@vt.edu
```

```
#SBATCH --mail-type=END
```

Example of requesting notification of job BEGIN, END, FAIL, REQUEUE, and STAGE_OUT:

```
#SBATCH --mail-user joe@vt.edu
```

```
#SBATCH --mail-type=ALL
```

Submitting a Job Array

The --array=<indexes> directive is used to submit a job array, multiple jobs to be executed with identical parameters. The indexes specification identifies what array index values should be used. Multiple values may be specified using a comma separated list and/or a range of values with a "-" separator. For example, "--array=0-15" or "--array=0,6,16-32". A step function can also be specified with a suffix containing a colon and number. For example, "--array=0-15:4" is equivalent to "--array=0,4,8,12". A maximum number of simultaneously running tasks from the job array may be specified using a "%" separator. For example "--array=0-15%4" will limit the number of simultaneously running tasks from this job array to 4. The minimum index value is 0. The maximum value is one less than the configuration parameter MaxArraySize.

Example:

```
#SBATCH --array=1-8
```

Specifying Memory Requirements Per Allocated CPU

Each of the infer cluster job queues (a.k.a., partitions) is configured with a default real memory size available per allocated CPU (DefMemPerCPU), expressed in MegaBytes. The current DefMemPerCPU settings are:

Queue / Partition	DefMemPerCPU
infer_q	5,696 MB (5.56 GB)
gpu_q	5,696 MB (5.56 GB)

Therefore, a single core job in the infer_q will be limited to 5,696 MB of RAM, a dual core job will be limited to 11,392 MB of RAM, and so on. If your job will require more than the default memory per allocated CPU, then you should employ one of the following methods:

(a) it is recommended that you manually specify your memory requirement using the "--mem-per-cpu" option;

(b) as an alternative workaround, you could specify a larger number if cores using "--ntasks-per-node" to give you a memory limit equal to DefMemPerCPU x (number of cores you specified);

(c) as another alternative workaround, you could specify the "--exclusive" option, which will allocate the whole node and therefore the whole node's memory.

Requesting Interactive Shell Jobs

The **srun --pty $SHELL** command can be used to request an interactive shell job on a compute node. The following examples illustrate how to request an interactive shell job with various options.

Single Core On Single Node Resource Allocation (The Default) Using Default Run Time Limit (1 hour)

```
[zorba@infer1 ~]$ srun -A myAccount --pty $SHELL

[zorba@inf001 ~]$ echo $SLURM_CPUS_ON_NODE

1

[zorba@inf001 ~]$
```

Whole Node Resource Allocation (--exclusive) Using Default Run Time Limit (1 hour)

```
[zorba@infer1 ~]$ srun -A myAccount --pty --exclusive $SHELL

[zorba@inf001 ~]$ echo $SLURM_CPUS_ON_NODE

32

[zorba@inf001 ~]$
```

Specifying a Run Time Limit In Minutes Format

```
[zorba@infer1 ~]$ srun -A myAccount --pty -t 300 $SHELL

[zorba@inf001 ~]$ squeue -l

Wed Mar 25 18:02:53 2020
             JOBID PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES NODELIST(REASON)
                46   infer_q     bash    zorba  RUNNING       0:06   5:00:00      1 inf001

[zorba@inf001 ~]$
```

Specifying a Run Time Limit In Hours:Minutes:Seconds Format

```
[zorba@infer1 ~]$ srun -A myAccount --pty -t 2:00:00 $SHELL

[zorba@inf001 ~]$ squeue -l

Wed Mar 25 18:09:37 2020
             JOBID PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES NODELIST(REASON)
                47   infer_q     bash    zorba  RUNNING       0:02   2:00:00      1 inf001

[zorba@inf001 ~]$
```

Working with Bank Accounts

Job accounting data is collected for all jobs, and resource usage by each job (nodes and elapsed time) is charged to a bank account. Each user has a default bank account, and each user may have access to additional bank accounts.

Specifying A Bank Account for Your Job

The -A option is used to specify a bank account for your job to charge against. Following are examples of using the -A option in batch jobs and for interactive shell jobs.

Specifying a Bank Account in Batch Jobs

Use the "#SBATCH -A" option in your job script, as shown in the example below. (Substitute an actual account name for the "myAccount" that is used in the example.)

```
#!/bin/bash


#SBATCH -A myAccount


#SBATCH --time=00:05:00

#SBATCH --nodes=1 --ntasks-per-node=1

#SBATCH -p infer_q


hostname
```

Specifying a Bank Account for Interactive Shell Jobs

Add the "-A" option to your interactive srun command, as shown in the example below. (Substitute an actual account name for the "myAccount" that is used in the example.)

```
zorba@infer1 ~> srun -A myAccount --pty $SHELL
```

Listing Your Bank Accounts

You may list all of the accounts to which you have access to on each HPC system using the following command:

```
zorba@infer1 ~> myaccounts
```

The above command is a shortcut for the following Slurm command:

```
zorba@infer1 ~> sacctmgr list associations user=$LOGNAME format=Account%16,Cluster
```

Add the **-l** (that is a lower case "el") or **--local** option to the myaccounts command to only list the accounts to which you have access for on the HPC system you are currently logged into:

```
zorba@infer1 ~> myaccounts -l
```

Displaying Queue Information About Jobs

The **squeue** command reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order. Some useful examples are shown below.

The following example shows the default squeue output, which includes all jobs of all users.

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                63   infer_q     bash  wgentry  R       0:17      1 inf001
                64   infer_q     bash    zorba  R       0:10      1 inf002
```

The following example shows an option to squeue to only display your own jobs by specifying your user name.

```
$ squeue -u zorba
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                64   infer_q     bash    zorba  R       0:22      1 inf002
```

The following example shows an option to squeue to only display a specific job by specifying the job ID.

```
$ squeue -j 64
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                64   infer_q     bash    zorba  R       0:29      1 inf002
```

Cancelling Jobs

The **scancel** command is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

The following example shows the use of scancel with a job ID as the argument. It illustrates how one user cannot cancel the jobs of another user, and how a cancelled job goes into a completing (CG) job state for up to 32 seconds before ending.

```
[zorba@infer1 ~]$ squeue

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                63   infer_q     bash  wgentry  R      16:34      1 inf001
                64   infer_q     bash    zorba  R      16:27      1 inf002

[zorba@infer1 ~]$ scancel 63

scancel: error: Kill job error on job id 63: Access/permission denied

[zorba@infer1 ~]$ scancel 64

[zorba@infer1 ~]$ squeue

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                64   infer_q     bash    zorba CG      16:38      1 inf002
                63   infer_q     bash  wgentry  R      16:53      1 inf001

[zorba@infer1 ~]$ squeue

             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                63   infer_q     bash  wgentry  R      17:19      1 inf001
```

Displaying Detailed Information About Running Jobs

The **scontrol show job** command is used to report detailed job information about active jobs. Some useful examples are shown below.

The following example shows the default scontrol show job output for a specified job ID.

```
$ scontrol show job=9723

JobId=9723 JobName=hello_mvapich2_gcc-18x32.sbatch
   UserId=zorba(986791) GroupId=zorba(986791) MCS_label=N/A
   Priority=4294901733 Nice=0 Account=sysadmin QOS=normal WCKey=*defwckey
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:09 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2020-03-25T21:59:35 EligibleTime=2020-03-25T21:59:35
   AccrueTime=2020-03-25T21:59:35
   StartTime=2020-03-25T21:59:35 EndTime=2020-03-25T21:59:44 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-03-25T21:59:35
   Partition=infer_q AllocNode:Sid=infer1:17858
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=inf[001-018]
   BatchHost=inf001
   NumNodes=18 NumCPUs=576 NumTasks=576 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=576,mem=3204G,node=18,billing=576
   Socks/Node=* NtasksPerN:B:S:C=32:0:*:* CoreSpec=*
   MinCPUsNode=32 MinMemoryCPU=5696M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=USER Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/hello_mvapich2_gcc-18x32.sbatch
   WorkDir=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi
   StdErr=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/slurm-9723.out
   StdIn=/dev/null
   StdOut=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/slurm-9723.out
   Power=
```

The following example shows the --details option to scontrol show job which will display slightly more detailed information.

```
$ scontrol --details show job=9723

JobId=9723 JobName=hello_mvapich2_gcc-18x32.sbatch
   UserId=zorba(986791) GroupId=zorba(986791) MCS_label=N/A
   Priority=4294901733 Nice=0 Account=sysadmin QOS=normal WCKey=*defwckey
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   DerivedExitCode=0:0
   RunTime=00:00:09 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2020-03-25T21:59:35 EligibleTime=2020-03-25T21:59:35
   AccrueTime=2020-03-25T21:59:35
   StartTime=2020-03-25T21:59:35 EndTime=2020-03-25T21:59:44 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-03-25T21:59:35
   Partition=infer_q AllocNode:Sid=infer1:17858
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=inf[001-018]
   BatchHost=inf001
   NumNodes=18 NumCPUs=576 NumTasks=576 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=576,mem=3204G,node=18,billing=576
   Socks/Node=* NtasksPerN:B:S:C=32:0:*:* CoreSpec=*
   Nodes=inf[001-018] CPU_IDs=0-31 Mem=182272 GRES=
   MinCPUsNode=32 MinMemoryCPU=5696M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=USER Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/hello_mvapich2_gcc-18x32.sbatch
   WorkDir=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi
   StdErr=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/slurm-9723.out
   StdIn=/dev/null
   StdOut=/home/zorba/slurmtests/mpitests/mvapich2/hello_mpi/slurm-9723.out
   Power=
```

Displaying Accounting Information About Jobs

The **sacct** command is used to report job or job step accounting information about active or completed jobs. Some useful examples are shown below.

The following example shows the default sacct output for a specified job ID.

```
$ sacct -j 64

       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
64                 bash    infer_q   sysadmin          1 CANCELLED+      0:0
```

The following example shows an option to saact to display the Job ID, formatted CPU Time used (Elapsed time * CPU), Elapsed Time, and Timelimit for a specified job ID.

```
$ sacct -j 64 -o jobid,CPUTime,Elapsed,Timelimit

       JobID    CPUTime    Elapsed  Timelimit
------------ ---------- ---------- ----------
64             00:16:38   00:16:38   01:00:00
```

Using GPU Accelerators

You must submit jobs to the gpu_q in order to access compute nodes with attached GPU accelerators. You must also specify the --gres=gpu:1 job submit option in order to get access to the GPU accelerator on the compute nodes with attached GPU accelerators.

You will want to request an interactive shell job in the gpu_q using an interactive qsub command in order to develop and compile CUDA codes. For example:

```
zorba@infer1 ~> srun -A myAccount --pty --gres=gpu:1 -p gpu_q $SHELL

zorba@inf001 ~>
```

When you request GPUs, the system will set two environment variables - it is strongly recommend that you **do not change** these:

- CUDA_VISIBLE_DEVICES
- GPU_DEVICE_ORDINAL

To your application, it will look like you have GPU 0 (CUDA_VISIBLE_DEVICES=0 and GPU_DEVICE_ORDINAL=0):

```
zorba@infer1 ~> srun -A myAccount --pty --gres=gpu:1 -p gpu_q $SHELL

zorba@inf001 ~> env | grep DEVICE

CUDA_VISIBLE_DEVICES=0
GPU_DEVICE_ORDINAL=0
```

CUDA Applications

CUDA software is available via loadable environment modules. Use the modules for the latest version of CUDA software available on the cluster, which is CUDA 10.1 as of this writing. The software packages included are: the CUDA 10.1 Toolkit (cuda10.1-toolkit); the CUDA 10.1 SDK (cuda10.1-sdk); and, the CUDA DCGM (cuda-dcgm). The modules available for these packages are:

cuda10.1/**blas** adds CUDA 10.1 BLAS (CUBLAS) libraries to your environment variables

cuda10.1/**fft** adds CUDA 10.1 FFT (CUFFT) libraries to your environment variables

cuda10.1/**nsight** adds NVIDIA CUDA 10.1 Nsight Eclipse Edition to your environment variables

cuda10.1/**profiler** adds NVIDIA CUDA 10.1 Visual Profiler to your environment variables

cuda10.1/**toolkit** adds NVIDIA CUDA 10.1 Toolkit to your environment variables

cuda-**dcgm** adds NVIDIA CUDA DCGM to your environment variables

For general usage and compilation, it is sufficient to load just the cuda<version>/toolkit module:

```
zorba@infer1 ~> srun -A myAccount --pty --gres=gpu:1 -p gpu_q $SHELL

zorba@inf001 ~> module load cuda10.1/toolkit
```

The CUDA documentation under the directory $CUDA_INSTALL_PATH/*doc* ($CUDA_INSTALL_PATH is a variable set when you load the cuda<version> /toolkit module) on the cluster and at http://docs.nvidia.com/cuda has further information on how to run compute jobs using CUDA. Examples of CUDA code can be found at http://docs.nvidia.com/cuda/cuda-samples and under the directory $CUDA_SDK (a variable set when you load the cuda<version> /toolkit module) on the cluster.

Example gpu_q Job Script

```
#!/bin/bash

# Example Slurm sbatch script for GPU application jobs in the infer cluster
# gpu_q partition.

#
# SLURM JOB SCRIPT OPTIONS:
#
# The #SBATCH lines set various SLURM directive parameters for the job.
#
# The "#" sign in front of "SLURM" at the beginning of a sbatch script line is
# not a comment; Slurm interprets lines beginning with "#SBATCH" as denoting a
# job option directive.  All other instances of lines beginning with a "#" sign
# (including, e.g., "##SBATCH" and "# SBATCH") are comments as normal.
#
# It is important to keep all #SBATCH lines together near the top of the
# sbatch script.  No commands or variable settings should occur until
# after the #SBATCH lines.

# SPECIFYING THE ACCOUNT

#SBATCH -A myAccount
```

```
#
# REQUESTING RESOURCES:
#
# You will need to edit the --time resource limit line
# and the --nodes and --ntasks-per-node resource limits line
# to suit the requirements of your job.
#
# NOTE: --ntasks-per-node is equivalent to number of cores per node (and
#       therefore comparable to ppn in Torque), unless also specifying
#       a --cpus-per-task setting greater than 1.
#
# You may optionally request whole nodes by uncommenting the --exclusive line.
#
# You may optionally request memory required per node or memory required per
# allocated CPU by uncommenting and editing the --mem or --mem-per-cpu line.
#
# NOTE: The --mem parameter would generally be used if whole nodes are
#       allocated to jobs, while the --mem-per-cpu parameter would generally
#       be used if individual processors are allocated to jobs.


# Set the time, which is the maximum time your job can run in HH:MM:SS

#SBATCH --time=00:05:00


# Set the number of nodes, and the number of tasks per node (up to 32 per node)
#
# NOTE: The maximum number of tasks per node that can be requested will be
#       affected by the cpus per task setting if this setting is specified
#       with --cpus-per-task greater than 1.
#
# To access GPUs on GPU Nodes, also request a generic resource of type gpu.

#SBATCH --nodes=1 --ntasks-per-node=1 --gres=gpu:1


# If needed, request whole nodes by uncommenting the "#SBATCH --exclusive" line
# below

##SBATCH --exclusive


# If needed, request memory required per node or memory required per allocated
# CPU, in MegaBytes
#
# NOTE: Memory required per node would generally be used if whole nodes are
#       allocated to jobs, while memory required per allocated CPU would
#       generally be used if individual processors are allocated to jobs.
#
# NOTE: The commented out examples below show the currently configured default
#       values.

##SBATCH --mem=182272
##SBATCH --mem-per-cpu=5696


# Set the partition to submit to (a partition is equivalent to a queue)

#SBATCH -p gpu_q


# Optional: If modules are needed, source modules environment
#
# NOTE: This step is normally only needed for users whose default shell is
#       csh or tcsh.
#
# (Uncomment the following line if you wish to use it.)
# . /etc/profile.d/modules.sh

# Optional: Reset all loaded modulefiles to the normal defaults
# (Uncomment the following line if you wish to use it.)
# module reset
```

```
# Load any modules you require:
#
# NOTE: The following example module "null" does nothing; replace the module
#       name with one you wish to load.
#
# (Uncomment the following line if you wish to use it.)
# module load null
module load cuda10.1/toolkit


# Below here enter the commands to start your job

/apps/sysadm/Tesla_T4/cuda10.1/bin/x86_64/linux/release/cudaOpenMP
```

Example gpu_q Job Script for Multiple Serial GPU Processes Run in Parallel

NOTE: The following job example requires a cluster with at least two gpu_q nodes.

```
#!/bin/bash


#SBATCH -A myAccount

#SBATCH --time=00:05:00

#SBATCH --nodes=2 --ntasks-per-node=1 --gres=gpu:1

#SBATCH -p gpu_q


module load cuda10.1/toolkit


srun --nodes=1 --ntasks=1 --gres=gpu:1 --exclusive nvidia-smi -L &

srun --nodes=1 --ntasks=1 --gres=gpu:1 --exclusive nvidia-smi -L &

wait
```

Machine Learning Applications

Machine Learning software is available via loadable environment modules. The software packages included, listed by the modules available for those packages, are:

bazel An open-source build and test tool similar to Make, Maven, and Gradle. It uses a human-readable, high-level build language. Bazel supports projects in multiple languages and builds outputs for multiple platforms.

chainer-py37-cuda10.1-gcc A flexible framework for neural networks, designed to write complex architectures simply and intuitively.

cub-cuda10.1 A flexible library of cooperative threadblock primitives and other utilities for CUDA kernel programming.

dynet-py37-cuda10.1-gcc A neural network library developed by Carnegie Mellon University and many others. It is written in C++ (with bindings in Python) and is designed to be efficient when run on either CPU or GPU, and to work well with networks that have dynamic structures that change for every training instance.

fastai-py37-cuda10.1-gcc A library that simplifies training fast and accurate neural nets using modern best practices.

gpytorch-py37-cuda10.1-gcc A Gaussian process library implemented using PyTorch.

horovod-mxnet-py37-cuda10.1-gcc A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

horovod-pytorch-py37-cuda10.1-gcc A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

horovod-tensorflow-py37-cuda10.1-gcc A distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.

keras-py37-cuda10.1-gcc A high-level neural networks Python API, capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

ml-pythondeps-py37-cuda10.1-gcc Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.

mxnet-py37-cuda10.1-gcc A deep learning framework designed for both efficiency and flexibility. It allows a mix of symbolic and imperative programming and contains a dynamic dependency scheduler that automatically parallelizes any operations on the fly.

nccl2-cuda10.1-gcc Version 2 of NCCL, an implementation of multi-GPU and multinode collective communication primitives that are performance optimized for NVIDIA GPUs.

opencv3-py37-cuda10.1-gcc An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

protobuf3-gcc Version 3 of Protocol Buffers, a language-neutral, platform-neutral extensible mechanism for serializing structured data.

pytorch-py37-cuda10.1-gcc An optimized tensor library for deep learning using GPUs and CPUs. It provides tensor computation (like NumPy) with strong GPU acceleration, and deep neural networks built on a tape-based autograd system. It now includes Caffe2.

tensorflow-py37-cuda10.1-gcc An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.

tensorflow2-py37-cuda10.1-gcc An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.

tensorrt-cuda10.1-gcc A platform for high-performance deep learning inference designed by NVIDIA and built on CUDA. It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.

theano-py37-cuda10.1-gcc A Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

xgboost-py37-cuda10.1-gcc An optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the gradient boosting framework.

The machine learning environment modules automatically load additional environment modules as dependencies. For example:

```
zorba@infer1 ~> srun -A sysadmin --pty --gres=gpu:1 -p gpu_q $SHELL

zorba@inf001 ~> module list

Currently Loaded Modules:
  1) DefaultModules   3) site/infer/easybuild/setup   5) slurm/slurm/19.05.5
  2) shared           4) gcc/9.2.0

zorba@inf001 ~> module add tensorflow-py37-cuda10.1-gcc

zorba@inf001 ~> module list

Currently Loaded Modules:
  1) DefaultModules                          9) openblas/dynamic/0.2.20
  2) shared                                 10) protobuf3-gcc/3.8.0
  3) site/infer/easybuild/setup             11) cudnn7.6-cuda10.1/7.6.5.32
  4) gcc/9.2.0                              12) hdf5_18/1.8.21
  5) slurm/slurm/19.05.5                    13) cuda10.1/toolkit/10.1.243
  6) python37                               14) nccl2-cuda10.1-gcc/2.5.6
  7) ml-pythondeps-py37-cuda10.1-gcc/3.2.3  15) gcc5/5.5.0
  8) keras-py37-cuda10.1-gcc/2.3.1          16) tensorflow-py37-cuda10.1-gcc/1.15.2

zorba@inf001 ~> module add openmpi-geib-cuda10.1-gcc

zorba@inf001 ~> module list

Currently Loaded Modules:
  1) DefaultModules                         10) protobuf3-gcc/3.8.0
  2) shared                                 11) cudnn7.6-cuda10.1/7.6.5.32
  3) site/infer/easybuild/setup             12) hdf5_18/1.8.21
  4) gcc/9.2.0                              13) nccl2-cuda10.1-gcc/2.5.6
  5) slurm/slurm/19.05.5                    14) tensorflow-py37-cuda10.1-gcc/1.15.2
  6) python37                               15) hpcx/2.4.0
  7) ml-pythondeps-py37-cuda10.1-gcc/3.2.3  16) gcc5/5.5.0
  8) keras-py37-cuda10.1-gcc/2.3.1          17) cuda10.1/toolkit/10.1.243
  9) openblas/dynamic/0.2.20                18) openmpi-geib-cuda10.1-gcc/3.1.4



zorba@inf001 ~> python
```

```
Python 3.7.5 (default, Jan 29 2020, 22:53:01)
[GCC 9.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> import tensorflow as tf

2020-03-25 23:11:35.745189: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudart.so.10.1

>>> hello = tf.constant('Hello, TensorFlow!')

>>> sess = tf.Session()

2020-03-25 23:13:04.000512: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcuda.so.1
2020-03-25 23:13:04.029811: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1639] Found device 0 with
properties:
name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
pciBusID: 0000:12:00.0
2020-03-25 23:13:04.029865: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudart.so.10.1
2020-03-25 23:13:04.034132: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcublas.so.10
2020-03-25 23:13:04.037064: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcufft.so.10
2020-03-25 23:13:04.037850: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcurand.so.10
2020-03-25 23:13:04.041087: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcusolver.so.10
2020-03-25 23:13:04.042968: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcusparse.so.10
2020-03-25 23:13:04.051854: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudnn.so.7
2020-03-25 23:13:04.053007: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1767] Adding visible gpu
devices: 0
2020-03-25 23:13:04.053530: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions
that this TensorFlow binary was not compiled to use: AVX512F
2020-03-25 23:13:04.056691: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1639] Found device 0 with
properties:
name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
pciBusID: 0000:12:00.0
2020-03-25 23:13:04.056745: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudart.so.10.1
2020-03-25 23:13:04.056767: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcublas.so.10
2020-03-25 23:13:04.056781: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcufft.so.10
2020-03-25 23:13:04.056794: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcurand.so.10
2020-03-25 23:13:04.056818: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcusolver.so.10
2020-03-25 23:13:04.056838: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcusparse.so.10
2020-03-25 23:13:04.056856: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudnn.so.7
2020-03-25 23:13:04.057889: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1767] Adding visible gpu
devices: 0
2020-03-25 23:13:04.057921: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudart.so.10.1
2020-03-25 23:13:04.553245: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1180] Device interconnect
StreamExecutor with strength 1 edge matrix:
2020-03-25 23:13:04.553296: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1186] 0
2020-03-25 23:13:04.553310: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1199] 0: N
2020-03-25 23:13:04.555082: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1325] Created TensorFlow device (
/job:localhost/replica:0/task:0/device:GPU:0 with 14154 MB memory) -> physical GPU (device: 0, name: Tesla T4,
pci bus id: 0000:12:00.0, compute capability: 7.5)

>>> sess.run(hello)

b'Hello, TensorFlow!'

>>> a = tf.constant(10)

>>> b = tf.constant(32)

>>> sess.run(a+b)

42

>>> exit()
```

Using Singularity Containers

Singularity is a container solution for HPC environments, developed at Lawrence Berkeley National Laboratory. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data, using your choice of Linux distribution bases or leveraging other container formats (such as Docker Hub). You can make and customize containers locally using your own installation of Singularity, and then run those containers on the infer cluster.

Singularity documentation is available at https://www.sylabs.io/docs/, and a good quick start guide from that site is at https://www.sylabs.io/guides/3.5/user-guide/quick_start.html. The following examples build upon that quick start guide.

Accessing the Singularity Software

Singularity version 3.5.3 (as of this writing) is installed and available via a loadable environment module. Load the containers/singularity module in order to use the latest installed version:

```
zorba@infer1 ~> module load containers/singularity

zorba@infer1 ~> module list

Currently Loaded Modules:
  1) DefaultModules   3) site/infer/easybuild/setup   5) slurm/slurm/19.05.5
  2) shared           4) gcc/9.2.0                    6) containers/singularity/3.5.3
```

Using Pre-Built Images

You can download and use pre-built images from an external resource like Singularity Hub or Docker Hub. Here are examples of building a Singularity container by downloading a pre-built image; interacting with that container image via a shell; executing commands within that container image; and, finally, running that container image:

```
zorba@infer1 ~> module load containers/singularity

zorba@infer1 ~> singularity build hello-world.simg shub://vsoch/hello-world

WARNING: 'nodev' mount option set on /tmp, it could be a source of failure during build process
INFO: Starting build...
 59.75 MiB / 59.75 MiB [==============================================================] 100.00% 84.45 MiB/s 0s
INFO:    Creating SIF file...
INFO:    Build complete: hello-world.simg

zorba@infer1 ~> singularity shell hello-world.simg

Singularity> id

uid=986791(zorba) gid=986791(zorba) groups=986791(zorba),16521(arc.arcadm),7208062(nis.role.orchestra.opc),
7228715(arc.openondemand),7302633(org.dept.0358),7302634(org.orgn.035800),7304292(org.mgmt.m252),7304425(org.
srmgmt.s45),7632148(bi.cui.admin),7863760(arc.sysadmin)

Singularity> cat /etc/lsb-release

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.6 LTS"

Singularity> exit

exit


zorba@infer1 ~> singularity exec hello-world.simg cat /etc/os-release

NAME="Ubuntu"
VERSION="14.04.6 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04.6 LTS"
VERSION_ID="14.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"


zorba@infer1 ~> ./hello-world.simg

RaawwWWWWWRRRR!!  Avocado!
```

Building Images From Scratch On Your Local System

Building Singularity containers from scratch requires root priviledges. You must therefore install Singularity on a local system on which you have sudo privileges in order to build your own container images, which you can then copy up to the cluster and run. The following example shows how to build a Singularity container image from scratch using a Singularity recipe text file:

```
$ cat centos7.def

BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum

# If you want the updates (available at the bootstrap date) to be installed
# inside the container during the bootstrap instead of the General Availability
# point release (7.x) then uncomment the following line
UpdateURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/updates/$basearch/

%post
        yum -y install vim-minimal procps util-linux

%runscript
        cat /etc/os-release


$ sudo singularity build centos7.simg centos7.def


WARNING: 'nodev' mount option set on /tmp, it could be a source of failure during build process
INFO: Starting build...
INFO: Skipping GPG Key Import
INFO: Adding owner write permission to build path: /tmp/rootfs-a550a1f8-6d58-11ea-b0d4-548028656c89
INFO: Running post scriptlet
+ yum -y install vim-minimal procps util-linux

...

Complete!
INFO: Adding runscript
INFO: Creating SIF file...
INFO: Build complete: centos7.simg


$ ./centos7.simg

NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

Submitting Jobs To Run Singularity Container Images

Below are examples of running Singularity container images from interactive shell jobs and from batch jobs.

Running Singularity Container Images From Interactive Shell Jobs

```
zorba@infer1 ~> srun -A myAccount --pty $SHELL

zorba@inf151 ~> module load containers/singularity

zorba@inf151 ~> ./hello-world.simg

RaawwWWWWRRRR!! Avocado!
```

Running Singularity Container Images From Batch Jobs

```
zorba@infer1 ~> cat centos7.slurm

#!/bin/bash

#SBATCH -A myAccount


#SBATCH --time=00:05:00

#SBATCH --nodes=1 --ntasks-per-node=1

#SBATCH -p infer_q


module load containers/singularity

./centos7.simg


zorba@infer1 ~> sbatch centos7.slurm

Submitted batch job 127571


zorba@infer1 ~> cat slurm-127571.out

NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

Graphical Interface to Slurm

The **sview** command launches a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

NOTE: Your DISPLAY variable must be set on the login node for the sview command to succeed. The DISPLAY variable is automatically set on a login node when you log in using appropriate ssh options, such as with **ssh -XY**. In addition, the local machine from which you are logging into a login node must be running a X server in order for the sview graphical interface to display.

Resource Usage Limits

There are currently no resource usage limits as of this writing (these are subject to change):

- Maximum Nodes Per Job on the infer cluster: N/A
- Maximum Cores Per User in the infer_q partition: N/A

Getting Support

Request ARC support via VT 4Help at Advanced Research Computing (ARC) Help, which will submit a support request ticket.

If your support request is in regards to job problems, then please include the **job number** and the **location of your sbatch job script**.

The cluster includes popular HPC applications as well as applications used by individual VT labs and groups. Please submit a support ticket if additional applications are needed.