Introduction to SeaWulf
HPC for CSE 416 students

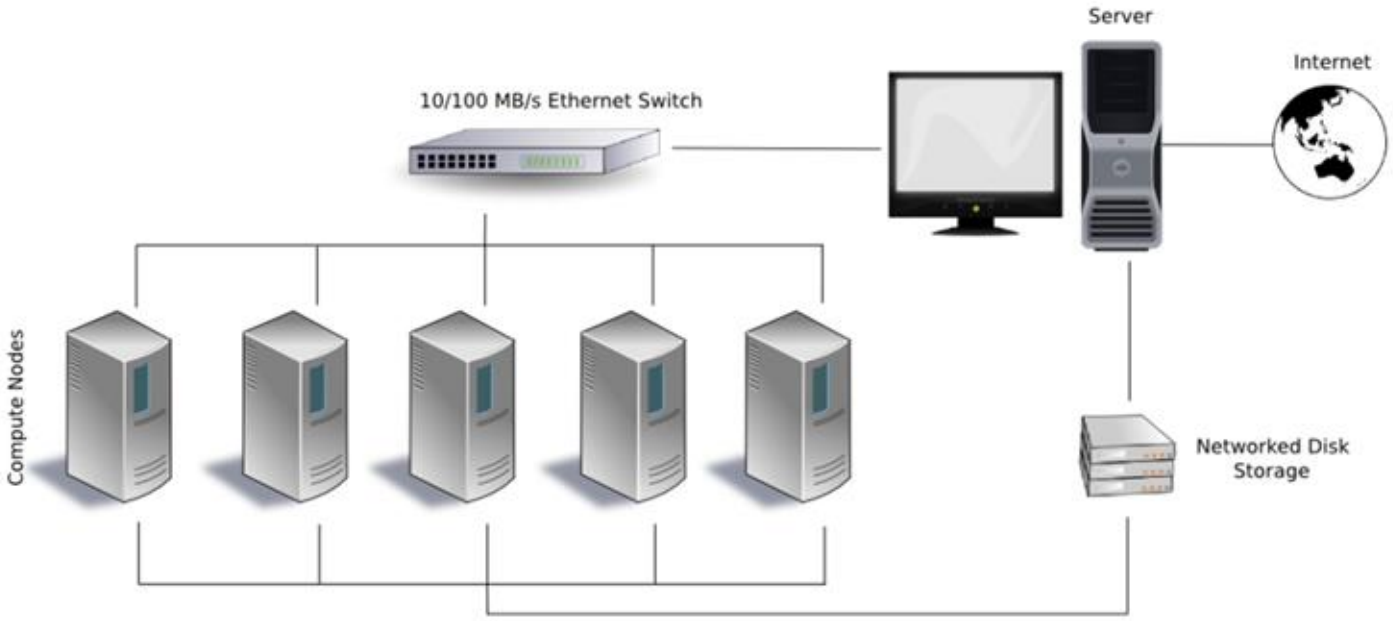
Dave Carlson, PhD

February 26, 2026

What's an HPC cluster, anyway?

A **High Performance Computing** (HPC) cluster contains multiple physically distinct computers ("nodes") that are connected over a high bandwidth network

A shared, parallel file system allows efficient access to the same data across all nodes



SeaWulf is …

- An HPC cluster dedicated to research applications for Stony Brook faculty, staff, and students

Available hardware:

- 5 **login nodes** = the entry points to the cluster

- 362 **CPU compute nodes** = where the work is done
  - 28 – 96 CPUS each
  - 128 GB – 1 TB RAM each

- 8 **GPU nodes** each with 4 Nvidia Tesla K80 GPUs
- 1 **GPU node** with 2 Tesla P100 GPUs
- 1 **GPU node** with 2 Tesla V100 GPUs
- 11 **GPU nodes** with 4 Tesla A100 GPUs
- Two **large memory nodes** each with 3 TB of RAM

How do I connect to SeaWulf?

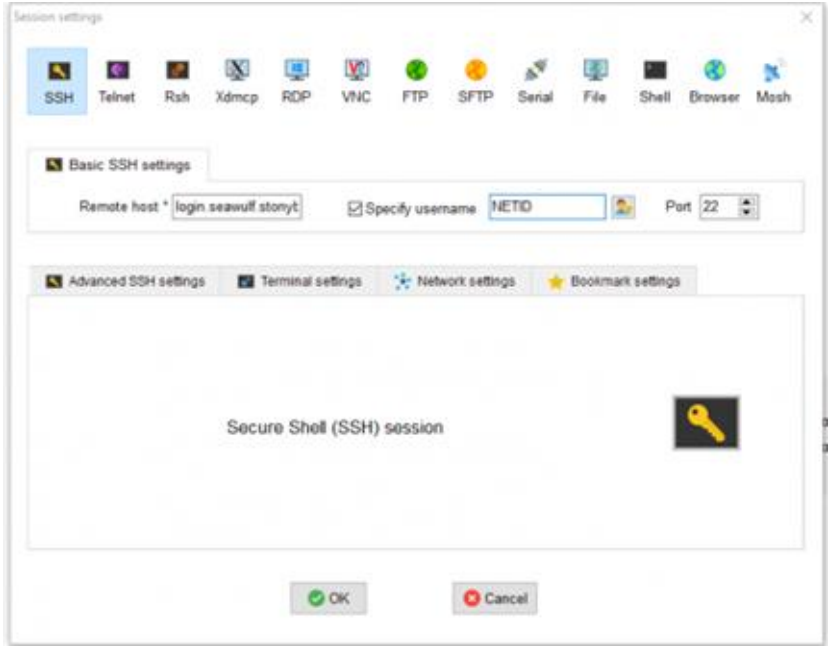
Mac & Linux users via the terminal:

ssh -X netid@**login**.seawulf.stonybrook.edu

ssh -X netid@**milan**.seawulf.stonybrook.edu

Windows users:

[Mobaxterm download link]



2-factor authentication with DUO

- Upon login, you will be prompted to receive and respond to a push notification, sms, or phone call:

```
Enter a passcode or select one of the following options:

 1. Duo Push to XXX-XXX-9515
 2. Phone call to XXX-XXX-9515
 3. SMS passcodes to XXX-XXX-9515

Passcode or option (1-3):
```

- Multiple failures to respond can lead to temporary lockout of your account

- *DUO 2FA can be bypassed if connected to SBU's VPN (GlobalProtect)*

Which login nodes should I access?

login.seawulf.stonybrook.edu provides access to:

- ❑ 28-core nodes
- ❑ All GPU nodes except A100

milan.seawulf.stonybrook.edu &
xeonmax.seawulf.stonybrook.edu provides access to:

- ❑ 40-core nodes
- ❑ 96-core nodes (AMD Milan)
- ❑ hbm-96-core nodes (Intel Sapphire Rapids)
- ❑ A100 GPU nodes

Access SeaWulf in your browser with Open OnDemand!

Point your browser to:

https://sn-ood.seawulf.stonybrook.edu/

See our OOD docs here:

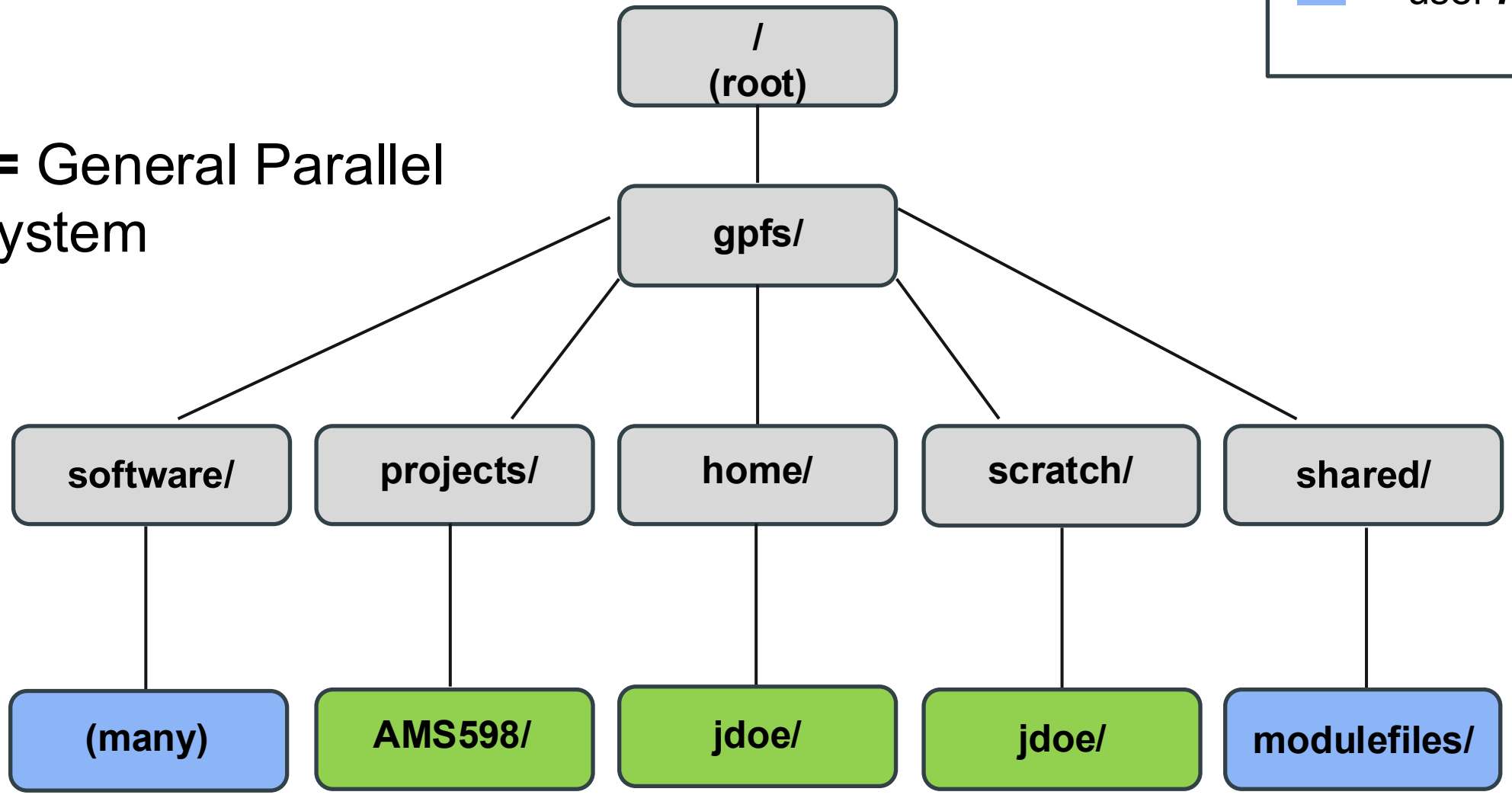
https://rci.stonybrook.edu/HPC/docs#ondemand

The SeaWulf filesystem

= user ***writeable***

= user ***readable***

gpfs = General Parallel File System



Important paths to remember

- /gpfs/home/netid = **your home directory (20 GB)**

These environment variables also point to your home directory

$HOME

~

- /gpfs/scratch/netid = **your scratch directory (20 TB for housing temporary and intermediate files)**

- /gpfs/projects/AMS530 = **your project directory (1 TB shared space accessible to all class members)**

How do I transfer files onto SeaWulf?

Mac & Linux users should use scp (secure copy) to move files to and from SeaWulf

To transfer files from your computer to SeaWulf

1. **Open terminal**
2. **scp /path/to/my/file netid@login.seawulf.stonybrook.edu:/path/to/destination/**

To transfer files from SeaWulf to your computer

1. **Open terminal**
2. **scp netid@login.seawulf.stonybrook.edu:/path/to/my/file /path/to/destination**

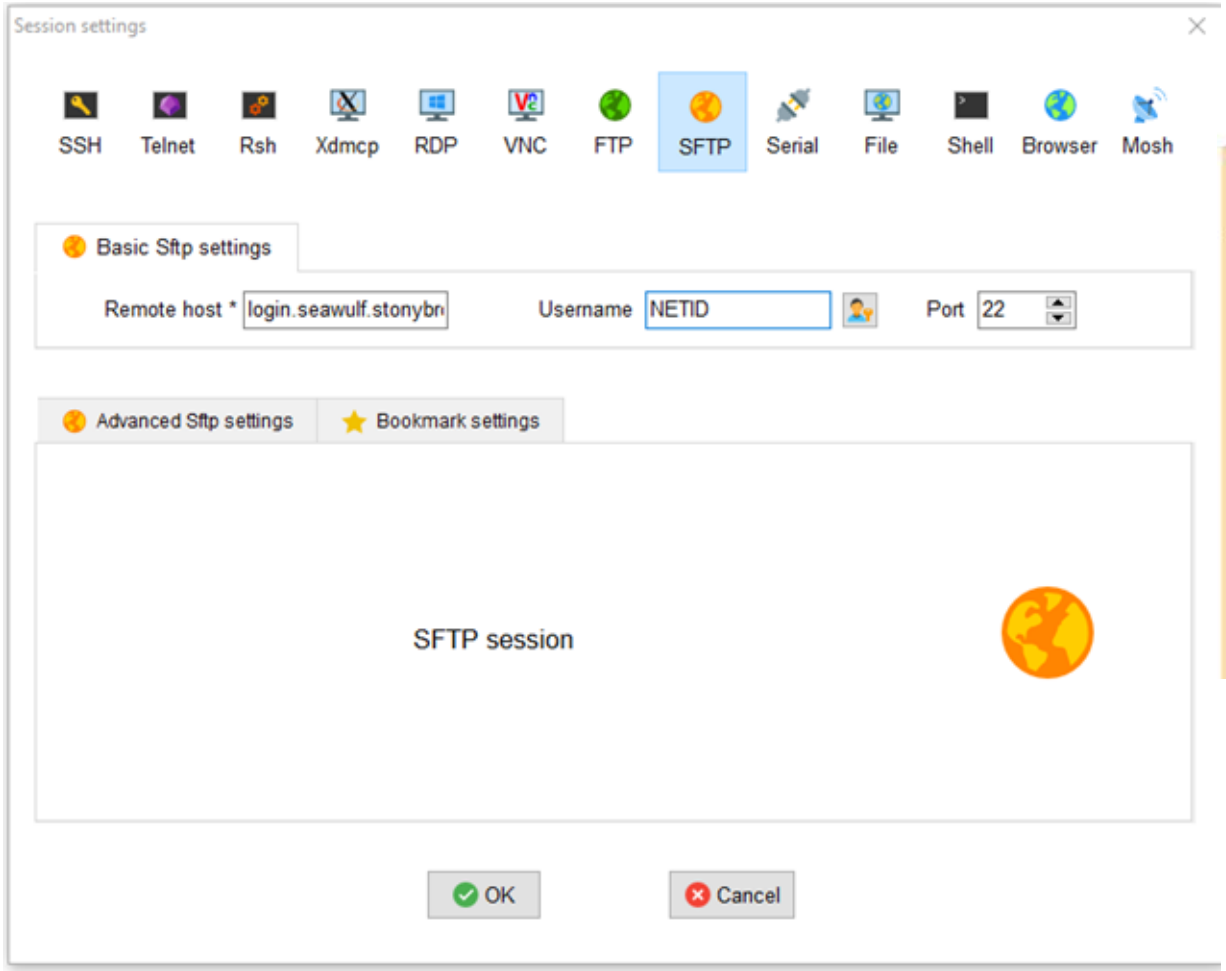
When possible, transfer archives (e.g. tarballs) or directories because:

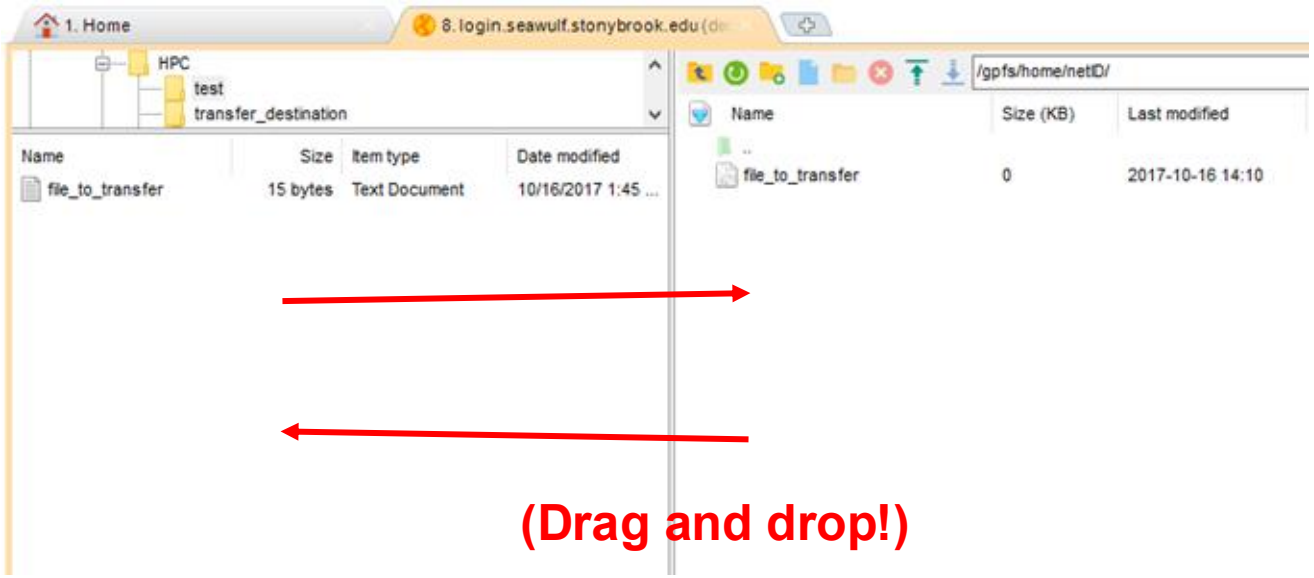
1. It's faster to transfer one large file than many small files

2. Unless connected to the SBU VPN, you may receive 1 DUO prompt for every scp command you run!

How do I transfer files onto SeaWulf?

Windows users:





(Drag and drop!)

Using the module system to access software

Useful module commands:

module avail

module load

module list

module unload

module purge

```
[decarlson@cn-mem ~]$ module avail
------------------------------------------------------------------------ /cm/local/modulefiles ------------------------------------------------------------------------
cluster-tools/8.1  cmd  dot  freeipmi/1.5.7  ipmitool/1.8.18  lua/5.3.4  module-git  module-info  null  openldap  openmpi/mlnx/gcc/64/3.1.1rc1  shared  shared.06DEC2019

------------------------------------------------------------------------ /cm/shared/modulefiles ------------------------------------------------------------------------
cuda10.0/blas/10.0.130      cuda10.1/fft/10.1.243       cuda80/nsight/8.0.61    cuda90/profiler/9.0.176  cuda91/toolkit/9.1.85    cudnn7.6-cuda10.1/7.6.5.32  netcdf/gcc/64/4.5.0
cuda10.0/fft/10.0.130       cuda10.1/nsight/10.1.243    cuda80/profiler/8.0.61  cuda90/toolkit/9.0.176   cuda92/blas/9.2.88       default-environment         netperf/2.7.0
cuda10.0/nsight/10.0.130    cuda10.1/profiler/10.1.243  cuda80/toolkit/8.0.61   cuda91/blas/9.1.85       cuda92/fft/9.2.88        gcc5/5.5.0                  openmpi/cuda/64/3.1.4
cuda10.0/profiler/10.0.130  cuda10.1/toolkit/10.1.243   cuda90/blas/9.0.176     cuda91/fft/9.1.85        cuda92/nsight/9.2.88     hpcx/2.4.0                  pgi/64/19.10
cuda10.0/toolkit/10.0.130   cuda80/blas/8.0.61          cuda90/fft/9.0.176      cuda91/nsight/9.1.85     cuda92/profiler/9.2.88   hwloc/1.11.8                slurm/17.11.12
cuda10.1/blas/10.1.243      cuda80/fft/8.0.61           cuda90/nsight/9.0.176   cuda91/profiler/9.1.85   cuda92/toolkit/9.2.88    maui/3.3.1                  torque/6.1.1

------------------------------------------------------------------------ /gpfs/shared/modulefiles ------------------------------------------------------------------------
2.1.0                                  expat/2.2.5                                intel/compiler/64/2018/18.0.3      MaxQuant_test/1.6.10.43          qgis/2.18.15
abinit/8.10.3                          exprtk/1.0                                 intel/compiler/64/2019/19.0.0      meme/5.1.1                       qt/4.8.6
abinit/8.10.3-mvapich                  faststructure/1.0                          intel/compiler/64/2019/19.0.3      mesa/13.0.5                      qt/5.0.2
acml/gcc-int64/64/5.3.1                FastTreeMP/2.1.10                          intel/compiler/64/2019/19.0.4      mesa/18.3.3                      qt/5.1.1
acml/gcc-int64/fma4/5.3.1              fenics/2017.1.0                            intel/compiler/64/2019/19.0.5      metis/metis-5.1.0                qt/5.2.1
acml/gcc-int64/mp/64/5.3.1             fftw2/openmpi/gcc/64/double/2.1.5          intel/compiler/64/2020/20.0.0      mgltools/1.5.6                   qt/5.3
acml/gcc-int64/mp/fma4/5.3.1           fftw2/openmpi/gcc/64/float/2.1.5           intel/compiler/64/2020/20.0.1      mmseqs/2-R11                     qt/5.4
acml/gcc/64/5.3.1                      fftw3/mvapich2/3.3.8                       intel/compiler/64/2020/20.0.2      mono/2.0                         qt/5.5
acml/gcc/fma4/5.3.1                    fftw3/mvapich2/3.3.8-float                 intel/mkl/64/2017/0.098            motioncor2/1.3.0                 qt/5.6.3
acml/gcc/mp/64/5.3.1                   fftw3/openmpi/gcc/64/3.3.7                 intel/mkl/64/2018/18.0.0           mpc/1.0.2                        qt/5.7
acml/gcc/mp/fma4/5.3.1                 fish/3.0.2                                 intel/mkl/64/2018/18.0.1           mpfr/3.1.5                       qt/5.8
afni/17.2.05                           fltk/1.3.0                                 intel/mkl/64/2018/18.0.2           mpi4py/3.0.2                     qt/5.9.0
amber/16                               fmriprep/20.0.7                            intel/mkl/64/2018/18.0.3           mpi4py/3.0.3                     qt/5.9.4
anaconda/2                             fmriprep/dependencies                      intel/mkl/64/2019/19.0.0           mpich/gcc/3.2.1                  qt/5.10.1
anaconda/3                             forwardGenomics/1.0                        intel/mkl/64/2019/19.0.3           mricrogl/1.0.20180623            quantum-espresso/gcc/6.2.0
ants/2.3.1                             freesurfer/5.3.0-HCP                       intel/mkl/64/2019/19.0.4           mricron/1.0.20180614             quantum-espresso/qe-6.0
apr-util/1.6.1                         freesurfer/6.0.0                           intel/mkl/64/2019/19.0.5           msm/3.0fsl                       quantum-espresso/qe-6.3MaX
apr/1.6.3                              freetype/2.9                               intel/mkl/64/2020/20.0.0           msprime/0.6.2                    quantum-espresso/qe-6.3MaX-v2
arcs/1.1.0                             fsl/5.0.6                                  intel/mkl/64/2020/20.0.1           MultiNest/3.10                   quantum-espresso/qe-6.4-24or28core
arm/allinea_studio/forge/20.0.3        fsl/5.0.10                                 intel/mkl/64/2020/20.0.2           multinest/3.10                   quantum-espresso/qe-6.4-40core
arm/allinea_studio/licenseserver/20.0.3 fsl/6.0.0                                 intel/mpi/32/2017/0.098            mummer/4.0.0-beta2               quantum-espresso/qe-d3q
arm/allinea_studio/reports/20.0.3      fsl/6.0.4                                  intel/mpi/64/2017/0.098            mvapich2/gcc/64/2.2rc1           qwt/6.1.3
armadillo/7.4.2                        fsl_fix/1.06.15                            intel/mpi/64/2018/18.0.0           mvapich2/gcc7.1/64/2.2rc1        R/3.3.2
ase/3.13.0                             gatk/4.0.0                                 intel/mpi/64/2018/18.0.1           namd/2.9                         R/3.4.2
atlas/3.10.3                           gatk/4.0.2                                 intel/mpi/64/2018/18.0.2           namd/2.13                        R/3.4.3
atom/1.34.0                            gc/7.2                                     intel/mpi/64/2018/18.0.3           nccl/2.3.5                       R/3.4.4
autodock-vina/1.1.2                    gcam/5.1.3                                 intel/mpi/64/2019/19.0.0           ncurses/6.0                      R/3.5.2
autodock/4.2.6                         gcc-stack                                  intel/mpi/64/2019/19.0.3           ncview/2.1.2                     R/3.6.0
autodock/intel/4.2.6                   gcc/6.5.0                                  intel/mpi/64/2019/19.0.4           ncview/2.1.7                     R/3.6.2
automake/1.14                          gcc/7.1.0                                  intel/mpi/64/2019/19.0.5           netcdf/gcc/4.7.1                 R/4.0.2
BayeScan/2.1                           gcc/7.2.0                                  intel/mpi/64/2020/20.0.0           netcdf/gcc/4.7.1-gcc-4.8.5       ra/0.2.1
bcl2fastq/2.20                         gcc/8.1.0                                  intel/mpi/64/2020/20.0.1           netcdf/gcc/4.7.1-gcc-6.5.0       raxml-ng/0.9.0
beagle/3.1.2                           gcc/9.2.0                                  intel/mpi/64/2020/20.0.2           netcdf/gcc/4.7.1-gcc-7.1.0       raxml-ng/intel/0.9.0
bimbam/1.0                             gcj/4.8.2                                  intel/tbb/64/2018/18.0.2           netcdf/gcc/4.7.3-parallel        ray/0.8.7
binutils/2.27                          gdal/2.2.3                                 intel/tbb/64/2018/18.0.3           netcdf/gcc/fortran/4.5.2-parallel rclone/1.36
binutils/2.30                          gdb/7.11                                   intel/tbb/64/2019/19.0.0           netcdf/intel/64/4.1.1            rclone/1.45
binutils/devel/2.27-27                 gdb/8.2.1                                  intel/tbb/64/2019/19.0.3           netcdf/intel/64/4.7.4-parallel   rDock/2013.1
biopandas/0.2.4                        GenomeAlignmentTools/1.0                   intel/tbb/64/2019/19.0.4           netcdf/rpm/64/4.1.1-3            readline/6.2.10
bioperl-run/1.006900                   gensim/3.4.0                               intel/tbb/64/2020/20.0.2           NeuroElf/1.1                     readline/7.0
biopython/1.69                         geos/3.7.0                                 internal/condamodule_template.txt  nibabel/1.2.0                    readline/8.0
blacs/openmpi/gcc/64/1.1patch03        geospatial/1.0                             internal/condamoduletest/1.0       nlopt/2.4.2                      reditools/2.0
blas/10.0                              gflags/2.2.0                               internal/condamoduletest/1.9.1     nmon/16                          reditools/prerequisites/2.0
blas/gcc/64/3.7.0                      git/2.12.2                                 iozone/3_434                       numactl/2.0.11                   relion/3.0
blas/gcc/64/3.8.0                      glew/2.1.0                                 ipyrad/2.2.0                       nwchem/6.8                       relion/3.0-beta
```

Using the module system to access software

System default python

```
[decarlson@milan2 ~]$ python
Python 3.9.21 (main, Jun 27 2025, 00:00:00)
[GCC 11.5.0 20240719 (Red Hat 11.5.0-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'pandas'
>>>
[decarlson@milan2 ~]$ module load anaconda/3
(base) [decarlson@milan2 ~]$ python
Python 3.10.18 | packaged by conda-forge | (main, Jun  4 2025, 14:45:41) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>>
(base) [decarlson@milan2 ~]$
```

Load a module!

Newer python version with more useful packages now available!

How do I do work on SeaWulf?

Computationally intensive jobs should *not* be done on the login node!

- To run a job, you must submit a batch script to the Slurm Workload Manager
- A batch script is a collection of bash commands issued to the scheduler, which which distributes your job across one or more compute nodes
- The user requests specific resources (nodes, cpus, job time, etc.), while the scheduler places the job into the queue until the resources are available

Example Slurm Job Script

All jobs submitted through a job scheduling system using scripts

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --time=05:00
#SBATCH --partition=short-40core
#SBATCH --job-name=parallel_job
#SBATCH --output=squared_numbers.txt

# load required modules

module load anaconda/3
module load gnu-parallel/6.0

# parallelize the calculation across each input, running 40 processes at a time

parallel --jobs 40 python number_square.py {} ::: {1..100}
```

SBATCH Flags
- Specify # nodes, CPUs, running time, queue, and email options

Load modules – add programs to your path and set important environment variables

Execute your script or command

How do I execute my Slurm script?

Jobs are submitted via the Slurm Workload Manager using the “sbatch” command

```
[decarlson@login1 iqtree]$
[decarlson@login1 iqtree]$
[decarlson@login1 iqtree]$
[decarlson@login1 iqtree]$
[decarlson@login1 iqtree]$
[decarlson@login1 iqtree]$ module load slurm/17.11.12
[decarlson@login1 iqtree]$ sbatch gnu_parallel_example.slurm
Submitted batch job 356633
[decarlson@login1 iqtree]$
```

This is your job ID.

Let’s try submitting a job!

```
module load slurm

cp -r /gpfs/projects/samples/gnu_parallel/python_example/ ~

cd python_example

sbatch parallel.slurm
```

Useful Slurm commands

sbatch <script> = submit a job

scancel <job id> = cancel a job

squeue = get job status

sinfo = get info on node/queue status and utilization

(see the following for a full list of Slurm commands)

https://slurm.schedmd.com/archive/slurm-21.08.8/man_index.html

What queue should I submit to?

How many cores do you need?

How many nodes do you need?

How much time do you need?

- ❑ **Only jobs using MPI should request more than 1 node!**
- ❑ **Use a "shared" queue if you don't need all the resources on a node**
- ❑ **There is often a tradeoff between resource usage and wait time!**
- ❑ **Don't wait until the last minute to submit jobs when you have a deadline!**

Queues accessed from `milan1` and `milan2`:

Queue	CPU Architecture	Vector/Matrix Extension	CPU Cores per Node	GPUs per Node	Node Memory[1]	Default Runtime	Max Runtime	Max Nodes	Min Nodes	Max Simultaneous Jobs per User	Multiple Users per Node
debug-40core	Intel Skylake	AVX512	40	0	192 GB	1 hour	1 hour	8	n/a	n/a	No
short-40core	Intel Skylake	AVX512	40	0	192 GB	1 hour	4 hours	8	n/a	4	No
short-40core-shared	Intel Skylake	AVX512	40	0	192 GB	1 hour	4 hours	4	n/a	n/a	Yes
medium-40core	Intel Skylake	AVX512	40	0	192 GB	4 hours	12 hours	16	6	1	No
long-40core	Intel Skylake	AVX512	40	0	192 GB	8 hours	48 hours	6	n/a	3	No
long-40core-shared	Intel Skylake	AVX512	40	0	192 GB	8 hours	24 hours	3	n/a	n/a	Yes
extended-40core	Intel Skylake	AVX512	40	0	192 GB	8 hours	7 days	2	n/a	3	No
extended-40core-shared	Intel Skylake	AVX512	40	0	192 GB	8 hours	3.5 days	1	n/a	n/a	Yes

See our documentation on SeaWulf's queues

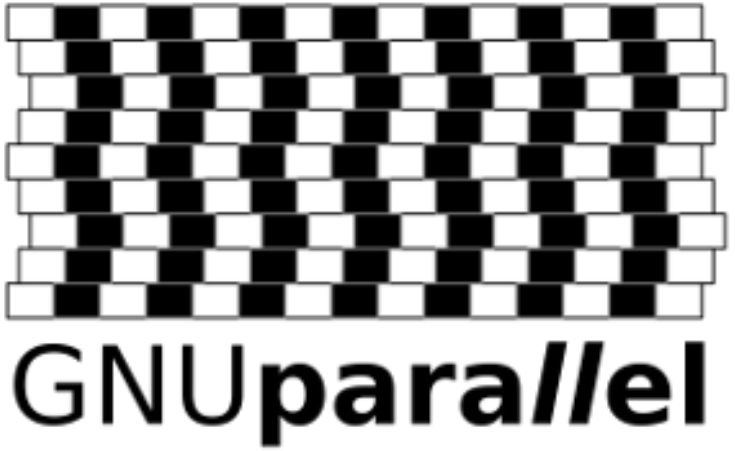
Use a ”shared” queue when you don’t need a full node

```
#!/usr/bin/env bash

#SBATCH --job-name=test_job
#SBATCH --output=avail_resources.txt
#SBATCH --time=00:05:00
#SBATCH -p short-40core-shared
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=10g



echo "Available CPUs:  $SLURM_JOB_CPUS_PER_NODE"
echo "Available Memory (MB): $SLURM_MEM_PER_NODE"
```

Parallel processing on the cluster

- **Parallelization *within a single compute node***
  - Lots of ways of doing this
  - Some tasks easily parallelized with simple tools (e.g., “Embarrassingly Parallel” tasks)
  - Language-specific options (e.g., Python’s Multiprocessing library)
  - OpenMP for multithreaded, shared memory tasks in C/C++/Fortran
- **Parallelization across *multiple nodes***
  - No communication – Slurm Array
  - Communication between processes needed – Requires the use of MPI

Parallel processing on a single node with GNU Parallel



- Perfect for “embarrassingly parallel” situations
- Available as a module: gnu-parallel/6.0
- Can easily take in a series of inputs (e.g., files or values) and run a command on each input simultaneously
- Lots of tutorials and resources available on the web!

https://www.gnu.org/software/parallel/parallel_tutorial.html (thorough!!)
https://www.msi.umn.edu/support/faq/how-can-i-use-gnu-parallel-run-lot-commands-parallel (many practical examples)

Parallel processing with GNU Parallel

```
#!/usr/bin/env bash

#SBATCH --nodes=1
#SBATCH --time=05:00
#SBATCH --partition=short-40core
#SBATCH --job-name=parallel_job
#SBATCH --output=squared_numbers.txt

# load required modules

module load anaconda/3
module load gnu-parallel/6.0

# parallelize the calculation across each input, running 40 processes at a time

parallel --jobs 40 python number_square.py {} ::: {1..100}
```

Optional flags to control behavior

This command will be run on each input

The input values to parallelize over

{} = placeholder for each input

Parallel processing on a single node with Python's multiprocessing library


```
#!/usr/bin/env python

import numpy as np
import multiprocessing as mp

# define a function for the calculation
def square_me(num):
        result = int(np.square(num))
        return(result)

# spawn a pool of parallel workers

p = mp.Pool(processes=40)

# using a pool of 40 parallel workers, loop over the values of 1 through 100 and apply the math function to each
for num in range(1,101):
        results = p.apply_async(square_me, [num])
        print(f'The square of {num} is {results.get()}')


# close the pool of workers
p.close()
```

Brief introduction to parallelization options for python:

https://www.anyscale.com/blog/parallelizing-python-code

Multithreading on a single node with OpenMP

OpenMP

- Framework for parallelization (multithreading) in a **shared-memory (single node)** context for C, C++, and Fortran
- Typically involves creating multiple threads within a single process instead of spawning multiple processes
- Useful when communication between parallel tasks is required
- Implemented in most modern compilers (GCC, Intel, LLVM, etc.)
- Resource usage controlled at runtime by environment variables

Check out this comprehensive tutorial!

Multithreading on a single node with OpenMP

Estimating pi with C and OpenMP

```c
#include <stdio.h>
#include <time.h>
#include <omp.h>
#include <stdint.h>

#define NPTS 1000000000

void main() {
   uint64_t i;
   double a,b,c,pi,dt,mflops;
   struct timespec tstart,tend;
   clock_gettime(CLOCK_REALTIME,&tstart);
   a = 0.5;
   b = 0.75;
   c = 0.25;
   pi = 0;
   #pragma omp parallel for reduction(+:pi)
   for (i = 1; i <= NPTS; ++i)
      pi += a/((i-b)*(i-c));
   clock_gettime(CLOCK_REALTIME,&tend);
   dt = (tend.tv_sec+tend.tv_nsec/1e9)-(tstart.tv_sec+tstart.tv_nsec/1e9);
   mflops = NPTS*5.0/(dt*1e6);
   printf("NPTS = %ld, pi = %f, threads = %d\n",NPTS,pi,omp_get_max_threads());
   printf("time = %f, estimated MFlops = %f\n",dt,mflops);
   }
```

```
#!/usr/bin/env bash

#SBATCH --job-name=openmp_pi
#SBATCH --output=openmp_pi.log
#SBATCH --nodes=1
#SBATCH --time=05:00
#SBATCH -p short-40core

# load a gcc module
module load gcc/12.1.0

# Environment variable to set how many threads we'll be using

export OMP_NUM_THREADS=40

# compile the code
gcc /gpfs/projects/samples/pi/openmp_pi.c -o openmp_pi -fopenmp

# execute the code
./openmp_pi
```

Load a compiler module

Specify # of threads

Compile w/ –fopenmp

Can I use multiple nodes for a single job?

MVAPICH

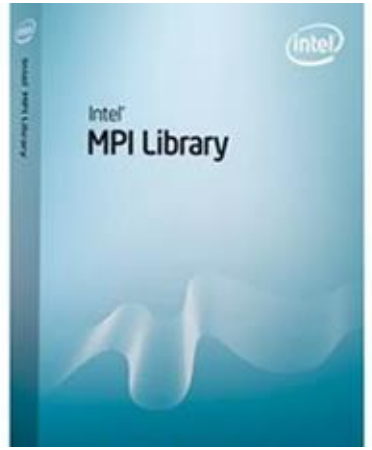
Yes! (…well…maybe)

Message Passing Interface (MPI) facilitates communication between processes within or among nodes (**distributed memory**)

Open MPI

Multiple “flavors” of MPI are available on SeaWulf

- **Mvapich**, **Intel mpich, OpenMPI**



Can I use multiple nodes for a single job?

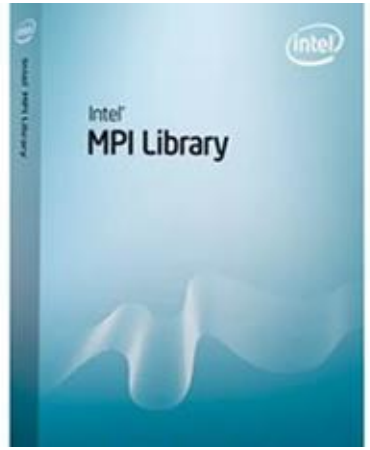
MVAPICH

To use MPI:

- Write code with MPI functions
- Compile with MPI wrapper

Open MPI

Language	GCC command	Mvapich wrapper
C	gcc	mpicc
C++	g++	mpicxx
Fortran	gfortran	mpif90

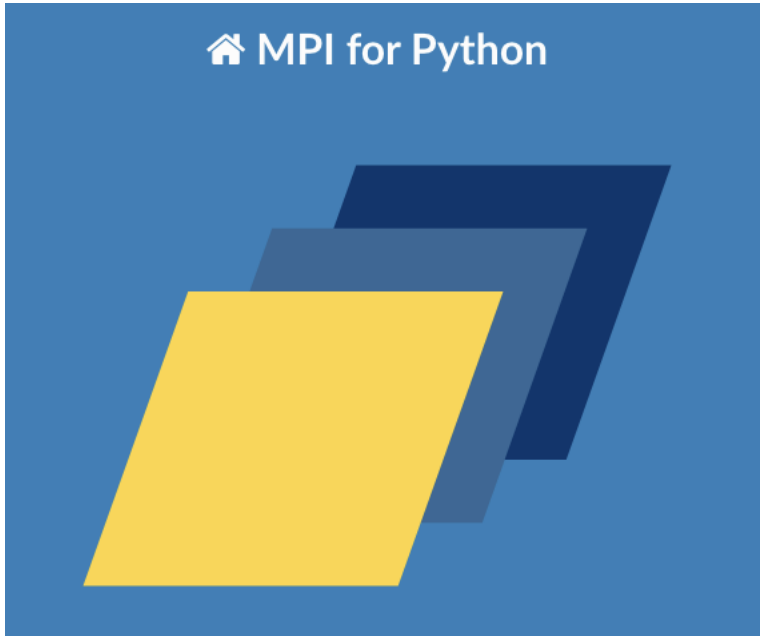


- Specify resource requirements in your Slurm script
- Execute code with mpiexec or mpirun

Can I use multiple nodes for a single job?

To use MPI *with python*:

- Use MPI4py: mpi4py.readthedocs.io
- Write code with MPI functions
- Specify resource requirements in your Slurm script
- Execute code with mpiexec or mpirun



Example MPI Job Submission Script

```
#!/usr/bin/env bash

#SBATCH --job-name=mpi_pi
#SBATCH --output=mpi_pi.log
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=28
#SBATCH --time=05:00
#SBATCH -p short-28core

# load a gcc module
module load mvapich2/gcc12.1/2.3.7

# set env variables which may help performance

export MV2_HOMOGENEOUS_CLUSTER=1
export MV2_ENABLE_AFFINITY=0

#export HWLOC_COMPONENTS=-gl

# compile the code with the mpi compiler wrapper
mpicc /gpfs/projects/samples/pi/mpi_pi.c -o mpi_pi

# execute the code with MPI
mpirun ./mpi_pi
```

Specify resource usage (nodes and MPI tasks)

Load an MPI module

Compile with MPI

Execute with MPI

Estimating pi with C and MPI

Parallelization FAQS:

Part 1: embarrassingly parallel tasks

Part 2: OpenMP & MPI

MPI Hello World

```
#!/usr/bin/env bash

#SBATCH --job-name=mpi4pytest
#SBATCH --output=mpi4pytest
#SBATCH --ntasks-per-node=8
#SBATCH --nodes=2
#SBATCH --time=05:00
#SBATCH -p short-40core


module load mpi4py/latest

mpirun -np 16 python /gpfs/projects/samples/python/mpi-hello-world.py
```

```
#!/usr/bin/env python

from mpi4py import MPI

comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()

print(f"Hello world from process {comm.rank} of {comm.size} from node {name}!")
```

“Hello from process 1 on node 1”
“Hello from process 2 on node 1”
“Hello from process 3 on node 1”
“Hello from process 4 on node 1”
…

mpirun

“Hello from process 1 on node 2”
“Hello from process 2 on node 2”
“Hello from process 3 on node 2”
“Hello from process 4 on node 2”
…

Hybrid OpenMP + MPI

Parallelization possibilities for a 2 40-core node job:

- ❑ 80 MPI tasks (40 per node)
- ❑ 1 MPI task per node + 40 OMP threads per task
- ❑ 2 MPI tasks per node + 20 OMP threads per task
- ❑ 4 MPI tasks per node + 10 OMP threads per task
- ❑ … etc.

Why combine OpenMP + MPI?

- ❑ Reduce memory footprint
- ❑ Reduce computational penalty for inter-node communication
- ❑ Take advantage of non-uniform memory access (NUMA)

Potential benefits are code-dependent—you should test your codes!

Hybrid OpenMP + MPI job

```
#!/usr/bin/env bash

#SBATCH --job-name=hybrid_pi
#SBATCH --output=hybrid_pi.log
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --time=05:00
#SBATCH -p short-40core

# load an mpi module
module load mvapich2/gcc12.1/2.3.7


# set the number of OpenMP threads to the number of cores per task

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

echo "Running pi calculation with $SLURM_NTASKS tasks using $OMP_NUM_THREADS threads for each task"

# compile the code with the mpi compiler wrapper
mpicc /gpfs/projects/samples/pi/hybrid_pi.c -o hybrid_pi -fopenmp

# execute the code with MPI
time mpirun ./hybrid_pi
```

Specify resource usage (nodes, MPI tasks, CPUs per task)

Load an MPI module

Specify number of threads

Compile with MPI + openmp flags

Execute with MPI

Need to troubleshoot? Use an interactive job!

Example:

```
[decarlson@dg-mem ~]$ srun -N 1 -p short-40core --pty bash
srun: job 293816 queued and waiting for resources
srun: job 293816 has been allocated resources
[decarlson@dn029 ~]$
```

”srun”: allocate a compute node in the short-40core queue

“--pty bash” run the bash shell on the compute node

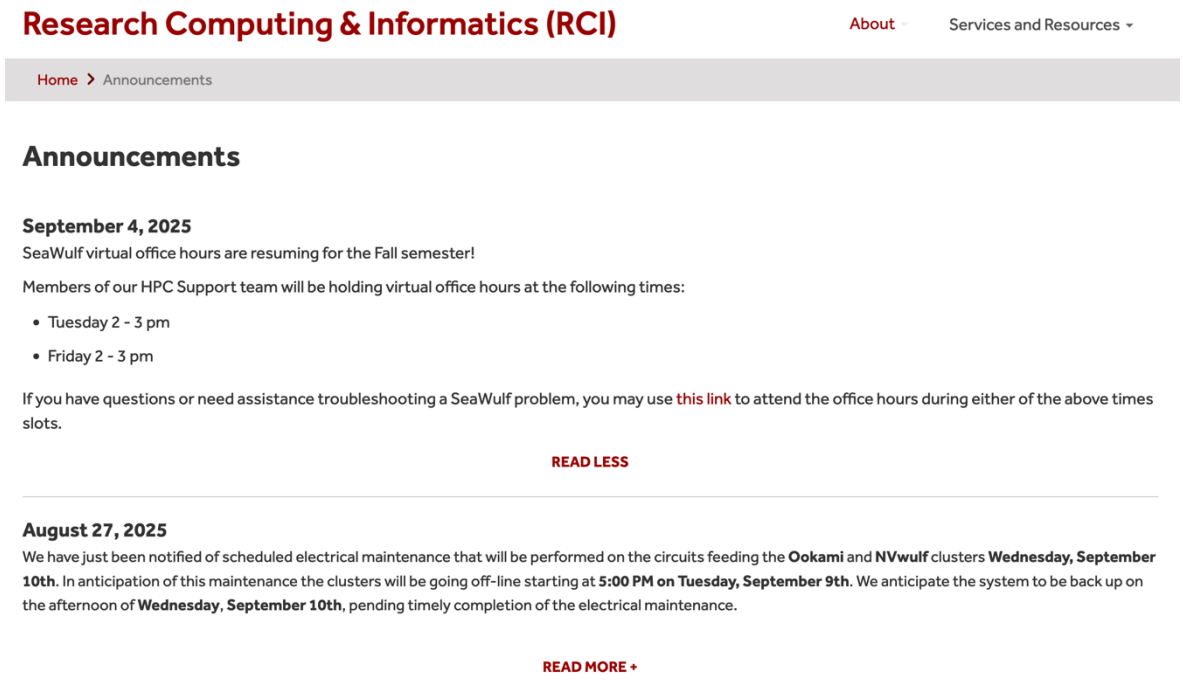
Once a node is available, you can issue commands on the command line

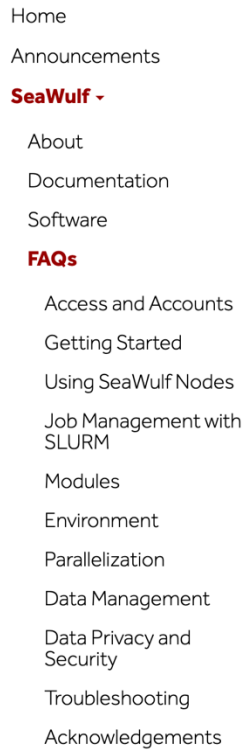
Good for troubleshooting,

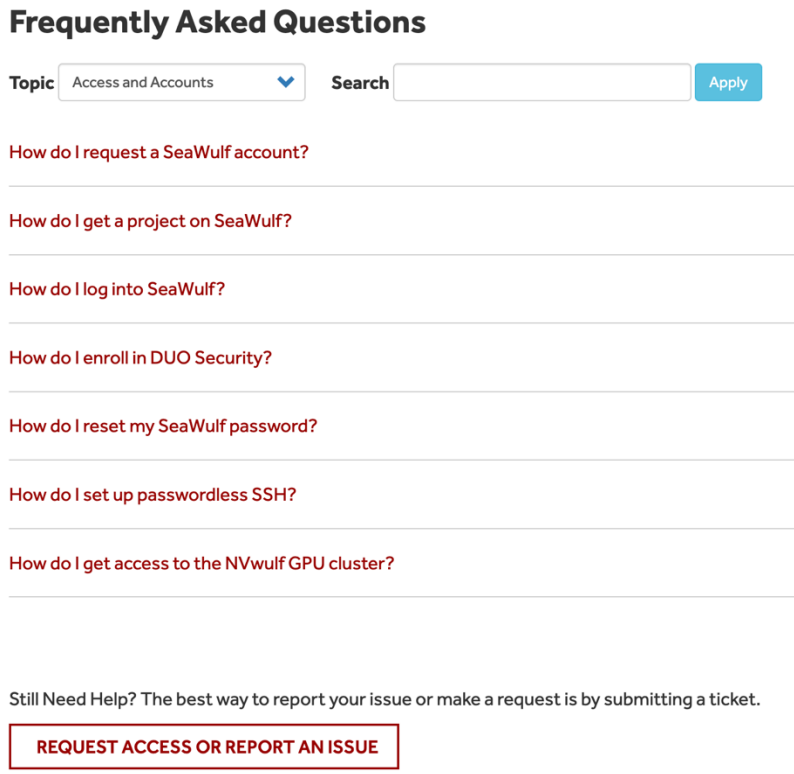
Inefficient once your code is working

Need more help or information?

Check out our documentation: https://rci.stonybrook.edu/hpc







Submit a ticket: https://iacs.supportsystem.com


QUESTIONS?