
WVU-RC Documentation

Release 2023.02.20

Guillermo Avendano-Franco

Feb 20, 2023

CONTENTS

1	Introduction	5
1.1	Infrastructure and Services	6
1.2	Getting Help	7
1.3	Policies	9
1.4	Purchasing Compute Nodes	10
2	Quick Start	13
2.1	Getting Access	13
2.2	Connect to the cluster (SSH)	14
2.3	UNIX/Linux Command Line Interface	18
2.4	Terminal-based Text Editors (nano)	26
2.5	Cluster Storage	28
2.6	Environment Modules	30
2.7	Workload Manager (SLURM)	32
2.8	File Transfer (Globus)	37
2.9	Web Interface (Open On-Demand)	38
3	Basic Usage	59
3.1	Terminal-based Text Editors	59
3.2	Data Storage	68
3.3	Environment Modules	70
3.4	Workload Manager (SLURM)	73
3.5	File Transfer (Globus and SFTP)	89
3.6	Web Interface (Open On-Demand)	91
4	Advanced Usage	113
4.1	Singularity Containers	113
4.2	Conda	119
4.3	GPU Computing	127
4.4	Environment Modules	130
4.5	Jupyter Notebooks	135
4.6	HDF5: Hierarchical Data Format	136
4.7	XWindow	137
4.8	Compile Source Code	139
5	Scientific Programming	141
5.1	Fortran, C and C++	142
5.2	Python Language	146
5.3	R Language	155
5.4	Julia Language	175

5.5	MATLAB	177
5.6	Perl Language	190
5.7	Parallel Programming: OpenMP	193
5.8	Parallel Programming: MPI	193
5.9	Parallel Programming: OpenACC	193
5.10	Parallel Programming: CUDA	196
6	Software Administration	197
6.1	Editing these documents	197
6.2	Installing Packages in User Locations	201
6.3	Linear Algebra	203
6.4	Boost 1.79	211
6.5	Message Passing Interface	212
6.6	HDF5 and NetCDF	213
6.7	Fast Fourier Transforms	216
6.8	Force Field Molecular Dynamics	216
6.9	CHARM++ and NAMD	237
6.10	Density Functional Theory	254
6.11	Big Data	278
6.12	Python 3.9.7	280
6.13	Matlab N-D multithreaded matrix operations (MMX)	283
6.14	Building Julia	301
6.15	Tinker9	301
6.16	Cuda	305
7	Domain Specific Details	307
7.1	Engineering: ANSYS Products	307
7.2	Engineering: ANSYS/Forte	309
7.3	Engineering: OpenFOAM 10	317
7.4	Bioinformatics: Using Bowtie2	321
7.5	Visualization: VisIt	325
7.6	Compiling Planetary Modeling Packages	334
7.7	NAMD	345
8	Clusters Specifications	351
8.1	Mountaineer Cluster	351
8.2	Spruce Knob	352
8.3	Thorny Flat Cluster	357
8.4	Go First Data-Analytics Cluster	361
9	References	363
9.1	Common Unix commands	363
9.2	Linux Commands	364
9.3	Software Centrally Managed	365
10	Indices and tables	375



RESEARCH COMPUTING

West Virginia University Research Computing (WVU-RC) is a team inside WVU's Research Office dedicated to supporting, enabling and advancing computational research at WVU.

WVU Research Computing maintains a portfolio of infrastructures to support its mission. We maintain several High Performance Computing (HPC) **Clusters** from general purpose to specialized ones, both on premises as on the cloud. WVU Research Computing also provide a variety of other services including a large research storage facility called **DataDepot**, and a Demilitarized Zone (DMZ) for high speed data transfers called **WVU Research Exchange (REX)**.

In addition to maintaining these facilities, WVU Research Computing offers support, consulting and training in areas of High Performance Computing, Data Analysis, Machine Learning and Parallel Programming.

The table below shows our portfolio of HPC resources (past and present):

Table 1: HPC Portafolio at WVU Research Computing

Infraestructure/ HPC Clusters	Compute Nodes (total/active)	CPU Cores (total/active)	Accelerators GPUs (total/active)	Description
Mountaineer (decommissioned)	32/0	384/0	0/0	First centrally managed HPC Cluster for WVU. CPUs from Intel Westmere microarchitecture (32 nm). Decommissioned on 2018.
Spruce Knob (end-of-life decom. 2023)	176/148	3376/3036	14/5 NVIDIA GPUs Tesla K20m Tesla K20Xm	General-purpose HPC cluster first commissioned in 2017. Heterogeneous cluster with Intel processors. Sandy Bridge, Ivy Bridge, Haswell and Broadwell.
Thorny Flat (in production)	178/178	6516/6516	47 NVIDIA GPUs P6000 (21) RTX 6000 (24) A100 (2)	General-purpose HPC cluster. Intel processors with Skylake and Cascade Lake. Installed at Pittsburgh Supercomputer Center.
GoFirst (in production)				Virtual Infraestructure running on AWS. Serves Business Data Analytics (BUDA) program. Chambers College of Business and Economics.
2				CONTENTS
Dolly Sods (to be installed)				GPU Accelerated HPC Cluster.

The contents of this website can be downloaded as a single PDF here: [docs_hpc_wvu.pdf](#)

There are several websites associated with WVU-RC activities, here is a list of the most relevant ones:

The official webpage in Research Office portal about the Research Computing Division [WVU Research Computing - Research Office](#)

The legacy documentation was a Wiki website that will continue to be online for a while [WVU Research Computing - Legacy Wiki](#)

The HelpDesk ticket system [WVU Research Computing - HPC HelpDesk](#)

If your research was possible thanks to the use of our clusters please acknowledge the support using these comments:
For Spruce Knob:

“Computational resources were provided by the WVU Research Computing Spruce Knob HPC cluster,
which is funded in part by NSF EPS-1003907.”

For Thorny Flat:

“Computational resources were provided by the WVU Research Computing Thorny Flat HPC cluster,
which is funded in part by NSF OAC-1726534.”

For requesting help, create a new ticket at the Research Computing HPC Help Desk web page. You are welcome to e-mail any member of the WVU-RC team directly, but since we are not always at our desk, the ticket system will guarantee that your support question will be seen by someone currently available.

Main Responsible for Documentation and Scientific Outreach [Guillermo Avendano-Franco](#)

**CHAPTER
ONE**

INTRODUCTION

The contents of this documentation are mostly focused on our High-Performance Computing (HPC) Clusters at West Virginia University Research Computing (WVU-RC). However, WVU-RC offers a variety of technologies and services beyond HPC, the *Introduction* chapter summarizes those services and the policies regarding its usage. Purchasing nodes is also possible and described.

First-time users should go to *Quick Start*. This chapter is presented as a tutorial where the basic elements of using a computer cluster are presented and guide the user step by step from getting an account to submit a job and take the results back to his/her own desktop computer.

Once you have followed the *Quick Start* we move into a chapter *Basic Usage*, this chapter assumes that user knows at least how to enter into the cluster and submit a basic job. The chapter provides basic information that was not covered in the *Quick Start* tutorial. We include more information about the command line interface, the various text-based editors, environment modules and more details on the various elements that were skipped before.

The next chapter is *Advanced Usage*. Intended for users who feel comfortable with the Command Line Interface. Advanced users are supposed to know how to create scripts using Shell Scripting or any other interpreted language. Advanced users could be interested in installing their own software, using containers, or Conda environments.

The chapter *Scientific Programming* is intended for users with interest in develop, test and run their own programs on WVU's clusters. We introduce the basics of using compilers, build systems, parallel programming, debugging, profiling and optimization. Developers can be considered as an advanced user who programs their own codes or integrates other codes in more than a simple linear workflow.

In the chapter *Software Administration* we make reference to the various tools that can be used to monitor the global health of the clusters. In general, users do not need to have that global view but knowing how the entire cluster works can give them insights about their own role in having an effective usage of the cluster.

The next chapter, *Domain Specific Details* collects sections for various packages that are relevant for a restrict number of researchers.

Chapter *Clusters Specifications* should serve as a reference about the current configuration of the clusters, in terms of hardware, software, modules, and queues.

The final chapter *References* collect tables and references about Unix commands, PBS options, variables, it should be of good use when you know exactly what you are looking for.

1.1 Infrastructure and Services

We provide access to centrally managed computational systems, several High-Performance Computing, and data analysis clusters. We also offer Infrastructure that supports researchers such as large and secure storage, high-speed data transfer channel using a demilitarized zone (DMZ), support in areas of High-Performance Computing, Parallel programming and visualization and training on those areas via workshops and seminars.

Here we summarize the different dimensions of action for WVU-RC:

1.1.1 High-Performance Computing

We operate 2 High-Performance computing Clusters “Spruce Knob” and “Thorny Flat”.

The High-Performance Computing facilities are funded by the National Science Foundation EPSCoR Research Infrastructure Improvement Cooperative Agreement #1003907, the state of West Virginia (WVEPSCoR via the Higher Education Policy Commission), the WVU Research Corporation and faculty investments.

Mountaineer was WVU’s first shared cluster and it was a 384 core Intel high-density computing cluster based on Xeon Westmere processors. Each node had 12 cores and 48GB of RAM, providing 4GB per core average. Storage was provided by a direct-attached SAN unit with 10TB of formatted disk space, as well as a network attached storage system with 60 TB of storage capacity. Mountaineer stop operation in April 2019 when the new cluster *Thorny Flat* started.

Spruce Knob is WVU’s current HPC system. This system is 176 nodes, 3,376 core heterogeneous high-density computing cluster based on Intel Xeon Sandy Bridge, Ivy Bridge, Haswell, and Broadwell processors. Spruce Knob follows a condo model where faculty members can purchase direct access to nodes on the cluster making them part owners of the cluster.

Thorny Flat is our new generation HPC cluster, with around 108 nodes, 4208 cores and 21 Nvidia P6000 GPU cards for extra computing power.

1.1.2 Data Analysis Cluster

GoFirst Cluster is a dedicated WVU MS Business Data Analytics computing resource that allows students in the Business Data Analytics M.S. program to gain experience in a controlled, secure cloud-computing environment. *GoFirst* is built from four compute nodes running HDFS shared filesystem, to run Hadoop and Spark jobs using RStudio as a frontend interface.

1.1.3 Research Exchange

REX is a Science DMZ, or demilitarized zone, which is a dedicated “express lane” network for research data traffic within the University’s larger network. It is funded through a nearly \$487,000 cyberinfrastructure grant that WVU Research Corp. won in the 2014 year from the National Science Foundation.

REX gives Information Technology Services the ability to separate research traffic from other Internet traffic, guarantee high-speed Internet2 access for WVU researchers, and facilitate data exchanges with off-campus collaborators. The upgrades also provide WVU researchers with greater access to off-campus resources such as national scientific supercomputing centers. The grant funded the development and deployment of two Data Transfer Nodes, high-performance data transfer “depots” that will improve the ability to move large science data sets. These Data Transfer Nodes have 640TB of raw disk storage, giving researchers a high-speed storage location when transferring large data sets.

1.1.4 HPC storage

WVU-RC offers researchers access to two tiers of storage through our Data Direct Network GRIDScaler system. Our standard tier of storage is available free to all users, but users also have the option to purchase dedicated group storage on the system.

The GRIDScaler system provides access to high-speed parallel GPFS storage and currently provides over 7 GB of throughput and 1 PB of raw storage.

All users of HPC systems have access to more than 400 TB of high-speed scratch storage. Scratch storage is for the temporary storage of files and gives researchers a place to process large amounts of data. In addition, each user is provided 10 GB of home directory space and 10 GB of group storage space upon request.

Some researchers prefer to have dedicated group storage on the HPC cluster to store large amounts of data for processing without the fear of it being removed. These researchers can purchase dedicated storage for their group at \$189/TB per year. This also offers an easy way to share data between researchers in the same group.

1.1.5 Data Depot

The WVU Research Data Depot is a centrally managed, reliable, secure and fast data storage system specifically designed to meet the university's diverse research storage needs. Designed to handle all size of files, from small to very large, researchers who use this service will have access to their data both on and off campus and can also use it to collaborate with researchers outside of WVU.

ITS designed the Data Depot to be easy to use. Researchers have access to drag and drop files through an interface they are accustomed to for users of Windows, OSX or Linux file managers. Command line tools, such as sftp, and Linux based command, such as mount, can also be used to access the files on lab PCs/servers.

More information on: [Research Data Depot](#)

1.1.6 Seminars and workshops

WVU-RC supports the mission of educating users on High-Performance Computing, Parallel Programming, and Data Analysis via seminars and workshops.

1.2 Getting Help

To request the opening of an account, software installation, general support on HPC, DataDepot and most of our services please open a ticket on

[WVU Research Computing - HPC HelpDesk](#)

The screenshot shows the West Virginia University Information Technology Services website. The top navigation bar includes links for Home, Getting Started, Scheduled Maintenance, Projects/Workspaces, Services, and Knowledge Base. Below the navigation is a breadcrumb trail: Service Catalog > Research Requests > Research Computing Requests > Research Computing / High-Performance Computing. On the right side of the header, there is a search bar labeled "Search the client portal" and a user profile for Daniel Turpen. A "Request" button is located at the bottom left of the main content area.

Service Catalog / Research Requests / Research Computing Requests / Research Computing / High-Performance Computing

Research Computing / High-Performance Computing

[+ Show Help](#) [- Hide Help](#)

Request support or access from Research Computing to the high-performance computing clusters being maintained through a collaborative effort of researchers at WVU.

Subject *

[Enter a brief description about this request]

Please select a help topic: *

[dropdown menu]

Request

Fill out the subject field, select a help topic, and fill out all the mandatory fields.

This screenshot shows the same website layout as the first one, but with more form fields. It includes a "Please provide any relevant JOB IDs from Tasks on Spruce Knob or Thorny Flat." field and a "Please provide any additional details for your request." field, both represented by large text input boxes. The "Request" button is also present here.

Service Catalog / Research Requests / Research Computing Requests / Research Computing / High-Performance Computing

Research Computing / High-Performance Computing

[+ Show Help](#) [- Hide Help](#)

Request support or access from Research Computing to the high-performance computing clusters being maintained through a collaborative effort of researchers at WVU.

Subject *

[Enter a brief description about this request]

Please select a help topic: *

[dropdown menu]

Please provide any relevant JOB IDs from Tasks on Spruce Knob or Thorny Flat.

[text input box]

Please provide any additional details for your request. *

Provide details on the reason(s) for opening the ticket.

Request

Once all relevant information is provided, click on *Request* and your request will be submitted to the WVU-RC for processing.

1.3 Policies

1.3.1 Acknowledging Your Use of the WVU High Performance Computing Environment

Maintaining a first class HPC environment is expensive, both in human and material terms. Research Office requests that you acknowledge your use of the campus clusters in papers or other publications of research which benefited from the use of these campus resources. Such acknowledgements assist us in convincing people in our administration and funding agencies of the value of these resources to our community, and help us to obtain new funding for their continued maintenance and expansion.

To acknowledge your use of the clusters, we request that you use the following wording:

For Spruce Knob:

The authors acknowledge the computational resources provided by the WVU Research Computing Spruce Knob HPC cluster, which is funded in part by NSF EPS-1003907.

For Thorny Flat:

The authors acknowledge the computational resources provided by the WVU Research Computing Thorny Flat HPC cluster, which is funded in part by NSF OAC-1726534.

We would also appreciate your informing us of publications in which you acknowledge one of the clusters. In retribution we will do efforts to give visibility to your work and the computational efforts that make them possible. Contact us at helpdesk@hpc.wvu.edu and shared with us the PDF or DOI of your publication.

1.3.2 Running Jobs on Login Node

Running jobs/tasks on the login node (i.e., what you are interacting with immediately after you log in) are not permitted. When the RC admin team notices this occurring, the processes will be killed immediately to ensure system stability. If you are having trouble submitting jobs through the scheduler, please follow instructions on [Getting Help](#) and we would be happy to help.

1.3.3 Data storage

Current storage limits are specific to each cluster and directories (Home and Scratch). For information about where these directories are located and storage quotas please visit the [Disk Storage](#) page.

1.3.4 Sharing User Directories

Initially sharing user directories or scratch space is not permitted. This is for tracking user purposes. However, we recognize that research is collaborative in nature, and therefore we are attentive to request for sharing resources across user spaces for temporary or fixed time limits to get jobs completed. Please follow instructions on [Getting Help](#) and the WVU-RC team with these requests.

Users also have the option of using group storage for sharing of files. A Principal Investigator (PI) may request up to 10 GB of group storage for free. Additional storage may be purchased if desired. For more information, please visit [Persistent Group Storage](#).

1.3.5 Getting Software Installed On the Cluster

Software needed on the cluster for research work can be installed at user requests. However, we initially recommend that software, scripts, and programming libraries be installed locally in user directories as opposed to system-wide. This is generally because system-wide installs of software/libraries not supported directly by Red Hat Enterprise Linux (RHEL) may get broken during system-wide updates. If the software/library has a large enough appeal (multiple users/multiple research teams) we can and will assist in installing a system-wide version. If assistance is needed for installing software/libraries locally in your user directory please follow instructions at [Getting Help](#), the WVU-RC team we will gladly assist.

1.3.6 User Priority for Job Submission

Depending on what hostname your submitting jobs on, different priority defaults are assigned to your job submission. On Mountaineer priority for queues are set by fair share queueing. This means that user priority is assigned based on a combination of the size of the job and how much system resources you have used during the given week, with higher priority assigned to larger jobs and/or user jobs that have used fewer system resources in the week. On Spruce knob, the research team nodes are first to come first serve priority, with jobs submitted before other jobs having higher priority. For specifications associated with the standby queue and community node queues on Spruce Knob, please see the [Spruce Knob Queue](#) page. If you would like to have different priority settings for your research teams queue please follow instructions at [Getting Help](#) for the WVU-RC team attend your request.

1.3.7 Database Management on Clusters

The current set-up for the HPC clusters is for common HPC use. This means that the compute nodes where a set-up with the idea that data will be transferred to the system, computed, and then transferred off. Database management systems are not currently supported on the cluster because the compute nodes were never initially set-up to handle the data storage required. However, if your research team has a need for Managed Database systems that have large storage needs with a database management system back-end (i.e. MySQL, Oracle, etc) please [Getting Help](#) the RC HPC team and we can come up with solutions to handle these data requests.

1.3.8 X11 Forwarding - Running visualization software

Non compute intensive processes for visualization purposes can be run on the login node. These processes can include Gnuplot, R, and Matlab. However, if your visualization job requires computing data before producing graphs and figures, it is best to run these jobs through the scheduler in batch mode. Compute-intensive jobs, visualization or not, are not permitted to run on the head-node. If you have any questions about the best way to accomplish your computing goal, please follow instructions on [Getting Help](#) through the help desk and we will provide any assistance needed to fulfill your requirements.

1.4 Purchasing Compute Nodes

The West Virginia University HPC systems allow users to purchase dedicated compute node capacity for their groups. At present the following options are available for purchase. For questions or to purchase nodes please contact helpdesk@hpc.wvu.edu.

1.4.1 Thorny Flat Phase 3

Phase 3 Thorny Flat nodes could be installed at the end of 2022. The available configurations are as follows:

NOTE: Due to challenges in manufacturing and availability, GPU and compute configurations have been experiencing very long lead times, price fluctuation, and unexpectedly discontinued or unavailable products. Please continue to contact us with requests for resources and we will work with our vendor to provide approximate times and configuration availability. Due to constraints imposed by NVIDIA and integrators, consumer GPUs, are not available for purchase and installation on Data-Centers.

NOTE: Please contact us for a quote on your configuration to ensure accurate pricing.

Non-GPU Compute Nodes -

- Small Memory - 92GB - \$7500 approx
- Medium Memory - 192GB - \$8500 approx
- Large Memory - 384GB - \$9500 approx
- Extra Large Memory - 768GB - \$12500 approx

1.4.2 Dolly Sods Phase 0

GPU Capable Compute Nodes (Up to 8 GPUs per Node) will go into our upoming GPU cluster Dolly Sods to be provisioned at the end of 2022. Specifications are changing, the compute node could be similar in memory configurations to those in Thorny Flat.

Listed below is the cost per GPU card. Consumer cards are no longer an option for use in datacenter systems, and server grade GPUs from the Nvidia Ampere Series listed below are available in limited configurations due to supply chain. Contact us if you're interested in pricing out a GPU node.

- NVIDIA® A10 GPU Computing Accelerator - 24GB GDDR6 - PCIe 4.0 x16 - Passive Cooler \$4000 approx
- NVIDIA® A30 GPU Computing Accelerator - 24GB HBM2 - PCIe 4.0 x16 - Passive Cooler \$5000 approx
- NVIDIA® A40 GPU Computing Accelerator - 48GB GDDR6 - PCIe 4.0 x16 - Passive Cooling \$5300 approx
- NVIDIA® A100 GPU Computing Accelerator - 80GB HBM2 - PCIe 4.0 x16 - Passive Cooling \$14000 approx

1.4.3 How long will my compute node be available?

All HPC compute nodes are purchased with a five year warranty. Compute nodes will be allowed to run for up to 7 years within the following parameters:

- The final two years of the compute node life are outside warranty and are on a best effort basis.
- If software/OS can no longer support compute equipment prior to the end of its 7 year life the HPC team may, in consultation with the HPC Policy Committee, determine that the life of compute equipment is shorter than 7 years. Should this occur the HPC team will strive to provide at least six months of notice to the HPC community before equipment is decommissioned.
- In the event that compute nodes fail outside of warranty they will not be repaired. The HPC team will attempt to keep investor queues at the purchased capacity, to the extent possible, based on the following process and guidelines.
 - Investor compute nodes that fail out of warranty will be replaced with compute nodes from the UI queue within the same generation of hardware.

- When possible compute nodes will be replaced with same or higher specification hardware. This will not be possible in all cases. In cases where this is not possible the investor will be contacted by the HPC team with available options.
- Transfer of compute nodes from the UI queue to investor queues will occur in the order in which failures occur.
- UI queue compute node availability is finite and is unlikely to be able to sustain all investor queues at full capacity for a 7 year life. As such investors should not assume that their queue will remain at full capacity for the duration of the two year life outside of warranty.

1.4.4 Renting Compute Nodes

HPC node rentals are not available at the moment.

QUICK START

This is a short tutorial intended for first-time users. We assume no familiarity with High-Performance Computing (HPC). The only requirement is basic familiarity with your own computer in order to install the SSH client needed to connect to the cluster.

We start on [Getting Access](#) on how to request an account. Once the access is granted, we proceed to [Connect to the cluster \(SSH\)](#) to install and use a SSH client that allows you to get a terminal on the HPC cluster. A Terminal and its command line interface can be the first barrier for a user only familiarized with Graphical User Interfaces (GUI), on [UNIX/Linux Command Line Interface](#) we present the most basic commands that will help you to create and manipulate folders and files. The next step is to learn how to edit files, on [Terminal-based Text Editors \(nano\)](#) we show how to use nano a very simple but effective text editor. We will use it to create the first program in FORTRAN that we will use later to explain execution on a batch system. The next section, [Cluster Storage](#), explains the several options to store data on the cluster, this is very important for first-time users as they have the tendency to rely only on the \$HOME folder a storage space that is very limited in most scientific purposes. At this point, we are ready to submit our first job. The section [Workload Manager \(SLURM\)](#) shows how to write a submission script and submit the job to the queue system. Finally, section [File Transfer \(Globus\)](#) explains how to move files in and out between the HPC cluster and your own computer.

At the end of this tutorial, you should have your account activated, connected to spruce, able to create files and folders, submitting simple jobs to the queue system and taking the data back to your own machine.

2.1 Getting Access

A High-Performance Computing (HPC) cluster is a research infrastructure intended to be used by multiple users simultaneously in order to execute calculations that are beyond the capabilities of current personal computers and workstations.

There are two steps involved in getting access to our HPC clusters, having a *WVU account* and using that *WVU account* to get an *HPC account*.

2.1.1 WVU Login Account

To gain access to WVU's Research Computing systems (including HPC systems), users need to first have a [WVU Login Account](#).

You can request a *WVU login account* for someone from outside the WVU community in order for the person to have access to specific online resources here. You will need the person's legal name, birth date, and current email address in order to make the request, which can be accomplished by filling out the Special Account Request form. Click the **LOGIN Account Ticket** button to get started.

Any employee may submit a request on behalf of a colleague elsewhere and there is no charge to request or use this service.

If you need a WVU Login account, please make a request with [Service Desk](#).

Click on **LOGIN Account Ticket** and follow instructions.

2.1.2 Research Computing Account

After getting a *WVU Login Account*, you will need to request access to Research Computing systems through the [Research Computing Help Desk](#) web page by clicking on the ‘Open a new ticket’ button and selecting ‘New User Account Request’ under the help topic. More information on [Getting Help](#)

On *Help Topic* select *New User Account Request* Enter your personal information such as Full Name and WVU’s email address. Personal email addresses (MSN, Gmail, etc) are not allowed for WVU users, for external users, it is advisable for them to use an institutional account instead of a personal one.

Enter the field for *Principal Investigator (PI)*, he or she will be the person to be contacted in order to get your account accepted.

For accounting purposes is very important that you fill a *Project Title* and a fairly complete *Project Abstract* This information is collected in order to prepare usage reports for financing institutions.

HPC users must ensure data that is covered by Federal security or privacy laws (e.g., HIPAA, ITAR, FERPA, classified information, etc.) is not stored on any WVU HPC system. These systems currently do not meet the enhanced security requirements imposed or implied by those laws or regulations. By selecting the check mark you are acknowledging you will not store ANY protected data on WVU’s HPC systems.

Please acknowledge use of these supercomputing systems (Thorny Flat and/or Spruce Knob) at WVU, which are funded in part by the National Science Foundation EPSCoR Research Infrastructure Improvement Cooperative Agreement #1003907, National Science Foundation Major Research Instrumentation Program (MRI) Award #1726534, the state of West Virginia (WVEPSCoR via the Higher Education Policy Commission) and WVU, in your publications produced using these resources.

Once the ticket is submitted, the PI will be notified for acceptance and your *HPC account* will be created. You will be notified once the account is ready for usage.

For any additional questions regarding access, please email us at helpdesk@hpc.wvu.edu.

2.2 Connect to the cluster (SSH)

An HPC cluster is a big computing infrastructure intended for concurrent usage by many users. A desktop, laptop or even workstations are intended for a single user at a time. In general, Graphical User Interfaces (GUI) consume an important amount of resources even when the user is not making use of them, that is one of the reasons why is common practice in HPC clusters to only allow remote shell access and limited capabilities for GUI applications.

Due to security reasons, HPC clusters are intended to be accessed using a secure shell, the standard secure shell nowadays is called SSH. Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network such as the internet. The typical application of SSH in HPC is to provide remote command-line login and remote command execution.

In order to connect to any of our clusters, you need a *username*, your name as a user of the cluster. You should have obtained that *username* with your *WVU Login account*

WVU’s clusters are also open to users who are currently associated with another Higher Education Institution in the state of West Virginia. These users will still need a WVU Login account for access. If you are working with someone (i.e. a PI) at WVU and need access, please have that person complete the form located [here](#). If you are not working with anyone at WVU, please send an email helpdesk@hpc.wvu.edu For this request to work the requestor must be on the campus network or using WVU’s VPN.

The next step is to use an SSH client to connect to one of our clusters.

2.2.1 Get an SSH Client

An SSH Client is a piece of software that allows you to run a remote session on a computer, over a network using a secure connection based on the SSH protocol.

An SSH client is usually come installed on MacOS and most Unix/Linux distributions so if you are using one of those Operating Systems you do not need to install anything.

Windows users will have to acquire an SSH client. PuTTY is a free implementation of SSH for Windows platforms, it comes along with an xterm terminal emulator.

Go to [PuTTY](#) to know more about the product or download it directly from [PuTTY Download](#).

PuTTY is not the only SSH client available for Windows. See for example [Comparison of SSH clients](#) for several alternatives.

2.2.2 Connecting to WVU's Clusters via SSH

Currently, WVU has two clusters available for access. Depending on which cluster you are using, the instructions for connecting slightly differ.

General SSH Connection Instructions

On MacOS and Unix/Linux, open a terminal shell and type:

```
$> ssh <username>@<hostname>
```

or:

```
$> ssh -X <username>@<hostname>
```

Where *<username>* is your *WVU Login account* username and *<hostname>* is the name of the cluster you wish to connect to. The *-X* option is used to forward X windows applications running on the server to be forwarded to your local machine. Remember that the \$> symbols above are there to indicate a command on the terminal, you should not enter those initial characters.

We currently have two clusters *Thorny Flat* and **Spruce Knob* the hostnames are:

Table 1: WVU's High-Performance Computer (HPC) Clusters

Cluster	Hostname	Status
Spruce Knob	<i>spruce.hpc.wvu.edu</i>	Operational
Thorny Flat	<i>tf.hpc.wvu.edu</i>	Operational

Spruce Knob Connectivity Instructions

To connect to Spruce Knob use the following command:

```
$ ssh <username>@spruce.hpc.wvu.edu
```

Note: Two-factor authentication is required to connect to Spruce Knob when not on WVU's Main Campus Network. More about WVU's Two-Factor Authentication system can be found [here](#)

Thorny Flat Connectivity Instructions

To connect to Thorny Flat, you will first have to connect to WVU's SSH gateway server. This gateway server will allow you to connect the Thorny Flat, which is hosted at the Pittsburgh Supercomputing Center.:

```
$ ssh <username>@ssh.wvu.edu
```

More information on WVU's Gateway Service can be found [here](#). When your account is created to for Thorny Flat, you will automatically be approved for access to WVU's SSH Gateway Service.

Note: Two-factor authentication is required to connect to WVU's Gateway Service. More about WVU's Two-Factor Authentication system can be found [here](#).

then:

```
$ ssh tf.hpc.wvu.edu
```

Note: You cannot connect directly to Thorny Flat via SSH. Executing directly from your terminal or ssh client `ssh tf.hpc.wvu.edu` will always fail.

2.2.3 Logging In

When your SSH access is granted, you will be prompted with a login message with helpful commands and updates about the cluster.

At this point, you will get a terminal prompt such as:

```
<username>@srih0001:~$
```

All the commands executed from now on are happening on a remote machine, the Spruce Knob *head node*, this is the place were most of your direct interaction with the cluster happens.

2.2.4 Logging Out

Logging out of a cluster can be done with the exit command:

```
$> exit
```

The exit command will attempt to terminate any process running on the head. In some cases, you will get an error that jobs are either currently running or currently stopped. You can view stopped jobs using the jobs command:

```
$> jobs -l  
[1]+ 3325 Stopped vim script56.py
```

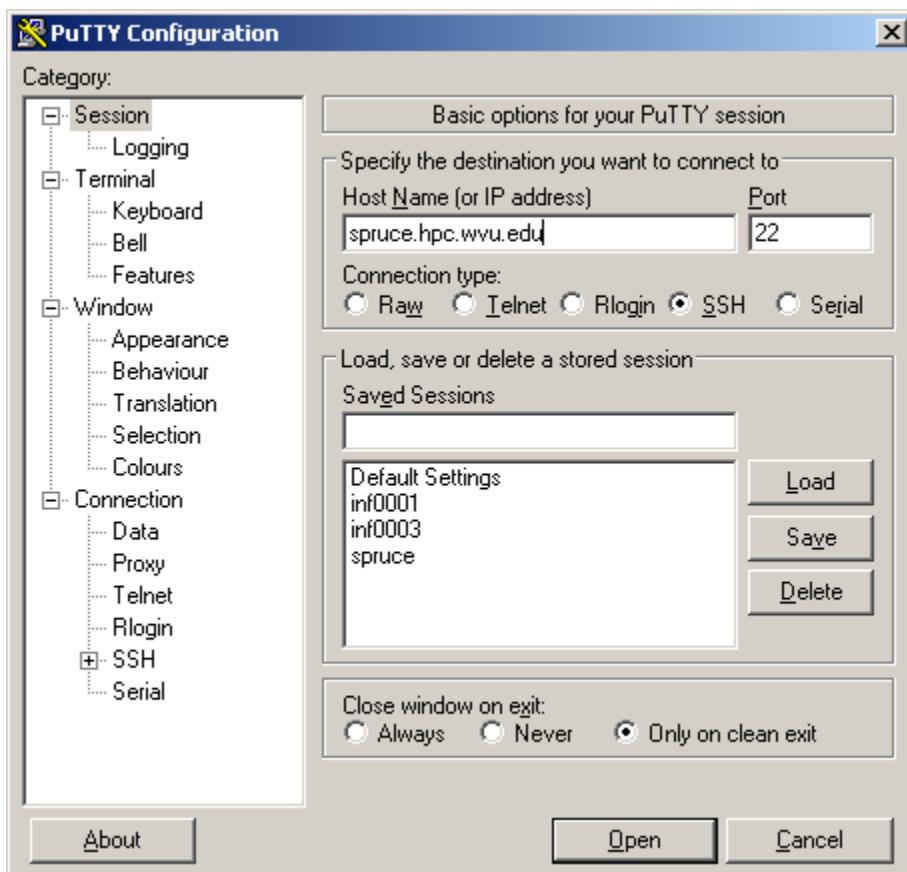
The output of jobs -l will give you the job PID number (in this case 3325) and the command (vim script56.py). To kill jobs preventing successful log out, use the kill command:

```
$> kill -s 9 3325
```

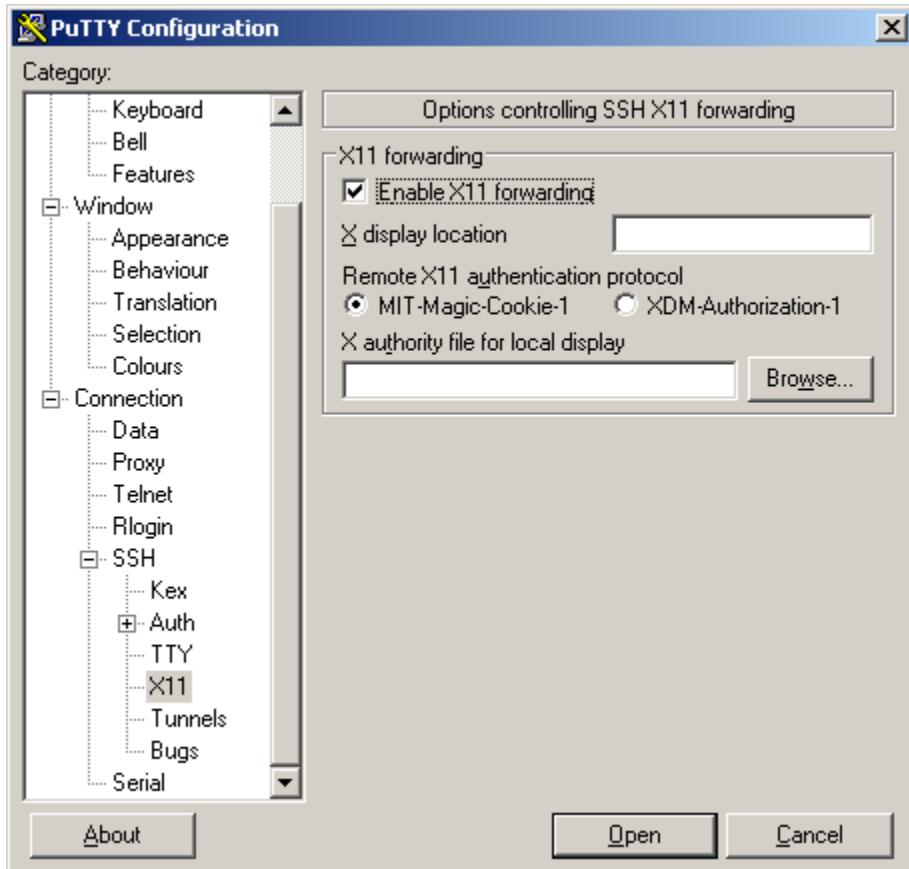
Once all jobs are terminated, the exit command will close the connection to the host. On section *Workload Manager (SLURM)* we will explain how to submit jobs on the queue system. Jobs on the queue system are not killed when you log out of the head node.

2.2.5 Putty Example

If you are using Windows and PuTTy, click on PuTTy Icon and enter the hostname



If you want to get X11 forwarding, ie remote windows popping on your local machine, enable X11 forwarding as shown below.



2.3 UNIX/Linux Command Line Interface

Most of the time, a user interacts with an HPC cluster using a Command Line Interface (CLI), also known as terminal. On a terminal, all your interaction is controlled by a program called *shell*. You can tell that you are in *shell* when you see what is called a *prompt* message, a set of characters that indicate that the *shell* is ready to receive instructions to operate.

To issue a command to the shell you type a command, occasionally followed by a few arguments. When you finish typing the full command you type the ENTER key, the command is executed and if the command is programmed to produce screen output it will appear on your terminal as it executes. Once the command is terminated you get a new *prompt* indicating that the shell is ready for new commands.

The CLI interface is a powerful way to interact with a remote computer for several reasons:

1. It takes few resources on the remote machine allowing the machine to serve tens, hundreds and in some cases even thousands of concurrent users.
2. The *shell* is far more than command reader and executor, it is actually a complete programming language. You can actually create complex sets of instructions by doing what is called *Shell Programming*; the ability to do shell programming is what we use in this document to differentiate a basic user from an advanced user..
3. Despite the learning curve being steeper, the CLI gives you far more control on the machine, so as you become more comfortable using it you will be able to do things that are too cumbersome on a Graphical User Interface (GUI).

Using a CLI is probably the biggest obstacle than beginners has to overcome before start taking advantage of a HPC cluster. Here we will offer a short straight to the point introduction to the bare minimal of managing files and folders

on the *shell*.

2.3.1 Files and Folders

Operating Systems from UNIX legacy, Linux and MacOS being the most prominent examples, uses the idea of *folders* and *files* to organize the data on their storage device. Differently from Windows, on UNIX there is no such idea of Drives C: or D: or letters for CD Drives or USB Keys. In UNIX every piece of data on any storage device is logically located in some place of a *filesystem tree*. Think about the *filesystem tree* where the lowest level is called *the root folder* and is indicated by /. From / you will see branches like /bin, /lib and many others, those are folders. Inside each folder, there are potentially more folders and files. The tree structure as a metaphor for storing data is very powerful and in the case of UNIX systems, that data structure is deeply exploited, even hardware devices such as sound cards, CD drives and hard drives receive a file-like entry on the rooted tree. When you insert a USB drive, Modern Linux distributions and MacOS will *mount* it automatically, meaning that it will receive a location on the tree, in the case of Linux, the mounting point is usually somewhere inside /media/, in MacOS the mounting point is /Volumes.

In the particular case of your interaction with the HPC cluster, there are two important folders that you should be aware of. They are so important that they receive special variables to tell you what they are. The \$HOME and \$SCRATCH, each user gets assigned a unique location for both folders. On them the user has the rights to create, modify and delete files and folders as he/she wants. In Linux the names of files and folders are basically arbitrary. Most linux filesystems are case sensitive, meaning that changing the file in lowercase and uppercase refers to different files. Using spaces for files and folders is allowed but discouraged as it force you to enclose the filename in quotes `` `` and `` `` for declare them or escaping the spaces like, This\ file\ has\ spaces.

2.3.2 The *echo* and *cat* commands

Your fist command will show you what those locations are. Execute:

```
$> echo $HOME
/users/<username>
$> echo $SCRATCH
/scratch/<username>
```

The first command to learn is echo. The command above uses echo to show the contents of two shell variables \$HOME and \$SCRATCH. Shell variables are ways to store information in such a way that the shell can use it when needed. Each user on the cluster receives appropriated values for those variables.

Let us explore a bit more the usage of echo. Enter this command line and execute ENTER:

```
$> echo "I am learning UNIX Commands"
I am learning UNIX Commands
```

The shell is actually able to do basic arithmetical operations, execute this command:

```
$> echo $((23+45*2))
113
```

Notice that as customary in mathematics products take precedence over addition. That is called the PEMDAS order of operations, ie “Parentheses, Exponents, Multiplication and Division, and Addition and Subtraction”. Check your understanding of the PEMDAS rule with this command:

```
$> echo $(((1+2**3*(4+5)-7)/2+9))
42
```

Notice that the exponential operation is expressed with the `**` operator. The usage of `echo` is important, otherwise, if you execute the command without `echo` the shell will do the operation and will try to execute a command called `42` that does not exist on the system. Try by yourself:

```
$> $ $(((1+2**3*(4+5)-7)/2+9))  
-bash: 42: command not found
```

As you have seen before, when you execute a command on the terminal in most cases you see the output printed on the screen. The next thing to learn is how to redirect the output of a command into a file. This will be very important later to submit jobs and control where and how the output is produced. Execute the following command:

```
$> echo "I am learning UNIX Commands" > report.log
```

With the character `>` redirects the output from `echo` into a file called `report.log`. No output is printed on the screen. If the file does not exist it will be created. If the file exists previously, the file is erased and only the new contents are stored.

To check that the file actually contains the line produced by `echo`, execute:

```
$> cat report.log  
I am learning UNIX Commands
```

The `cat` (concatenate) command displays the contents of one or several files. In the case of multiple files the files are printed in the order they are described in the command line, concatenating the output as the name of the command implies.

You can even use a nice trick to write a small text on a file. Execute the following command, followed by the text that you want to write, sat the end execute `Ctrl-D (^D)`, the *Control Key* followed by the `D` key. I am annotating below the location where `^D` should be executed:

```
$> cat > report.log  
I am learning UNIX Commands^D  
$> cat report.log  
I am learning UNIX Commands
```

In fact, there are hundreds of commands, most of them with a variety of options that change the behavior of the original command. You can feel bewildered at first by a large number of existing commands, but in fact most of the time you will be using very few of them. Learning those will speed up your learning curve.

Another very simple command that is very useful in HPC is `date`. Without any arguments, it prints the current date to the screen. Example:

```
$> date  
Mon Nov 5 12:05:58 EST 2018
```

2.3.3 Folder commands

As we mentioned before, UNIX organizes data in storage devices as a tree. The commands `pwd`, `cd` and `mkdir` will allow you to know where you are, move your location on the tree and create new folders. Later we will see how to move folders from one location on the tree to another.

The first command is `pwd`. Just execute the command on the terminal:

```
$> $ pwd  
/users/<username>
```

It is very important at all times to know where in the tree you are. Doing research usually involves dealing with a significant amount of data, exploring several parameters or physical conditions. Properly organizing all the data in meaningful folders is very important to research endeavors.

When you log into a cluster, by default you are located on your \$HOME folder. That is why most likely the command `pwd` will return that location in a first instance.

The next command is `cd`. This command is used to *change directory*. The directory is another name for *folder*. The term *directory* is also widely used. At least in UNIX the terms *directory* and *folder* are interchangeable. Other desktop operating systems like Windows and MacOS have the concept of *smart folders* or *virtual folders*, where the *folder* that you see on screen has no correlation with a directory in the filesystem. In those cases the distinction is relevant.

There is another important folder defined in our clusters, its called the scratch folder and each user has their own. The location of the folder is stored in the variable `$SCRATCH`. Note that this is internal convention and is not observed in other HPC clusters.

Use the next command to go to that folder:

```
$> cd $SCRATCH
$> pwd
/scratch/<username>
```

Notice that the location is different now, if you are using this account for the first time you will not have files on this folder. It is time to learn another command to list the contents of a *folder*, execute:

```
$> ls
$>
```

Assuming that you are using your HPC account for the first time, you will not have anything on your `$SCRATCH` folder. This is a good opportunity to start creating one folder there and change your location inside, execute:

```
$> mkdir test_folder
$> cd test_folder
```

We have use two new commands here, `mkdir` ``allows you to create folders in places where you are authorized to do so. For example your ``\$HOME and `$SCRATCH` folders. Try this command:

```
$> mkdir /test_folder
mkdir: cannot create directory '/test_folder': Permission denied
```

There is an important difference between `test_folder` and `/test_folder`. The former is a location in your current working directory (CWD), the later is a location starting on the root directory `/`. A normal user has no rights to create folders on that directory so `mkdir` will fail and an error message will be shown on your screen.

The name of the folder is `test_folder`, notice the underscore between *test* and *folder*. In UNIX, there is no restriction having files or directories with spaces but using them becomes a nuisance on the command line. If you want to create the folder with spaces from the command line, here are the options:

```
$> mkdir "test folder with spaces"
$> mkdir another\ test\ folder\ with\ spaces
```

In any case, you have to type extra characters to prevent the command line application of considering those spaces as separators for several arguments in your command. Try executing the following:

```
$> mkdir another folder with spaces
$> ls
another folder with spaces  folder  spaces  test_folder  test folder with spaces  with
```

Maybe is not clear what is happening here. There is an option for `ls` that present the contents of a directory:

```
$>ls -l
total 0
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:44 another
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:45 another folder with spaces
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:44 folder
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:44 spaces
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:45 test_folder
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:45 test folder with spaces
drwxr-xr-x 2 myname mygroup 512 Nov  2 15:44 with
```

It should be clear, now what happens when the spaces are not contained in quotes "test folder with spaces" or escaped as another\ folder\ with\ spaces. This is the perfect opportunity to learn how to delete empty folders. Execute:

```
$> rmdir another
$> rmdir folder spaces with
```

You can delete one or several folders, but all those folders must be empty. If those folders contain files or more folders, the command will fail and an error message will be displayed.

After deleting those folders created by mistake, let's check the contents of the current directory. The command `ls -1` will list the contents of a file one per line, something very convenient for future scripting:

```
$> ls -1
another folder with spaces
test_folder
test folder with spaces
```

2.3.4 Commands for copy and move

The next two commands are `cp` and `mv`. They are used to copy or move files or folders from one location to another. In the simplest case, those two commands take two arguments, the first argument is the source and the last one the destination. In the case of more than two arguments, the destination must be a directory. The effect will be to copy or move all the source items into the folder indicated as the destination.

Before doing a few examples with `cp` and `mv` let's use a very handy command to create files. The command `touch` is used to update the access and modification times of a file or folder to the current time. In case there is not such a file, the command will create a new empty file. We will use that feature to create some empty files for the purpose of demonstrating how to use `cp` and `mv`.

Lets create a few files and directories:

```
$> mkdir even odd
$> touch f01 f02 f03 f05 f07 f11
```

Now, lets copy some of those existing files to complete all the numbers up to `f11`:

```
$> cp f03 f04
$> cp f05 f06
$> cp f07 f08
$> cp f07 f09
$> cp f07 f10
```

This is good opportunity to present the ** wildcard*, use it to replace an arbitrary sequence of characters. For instance, execute this command to list all the files created above:

```
$> ls f*
f01  f02  f03  f04  f05  f06  f07  f08  f09  f10  f11
```

The *wildcard* is able to replace zero or more arbitrary characters, see for example:

```
$> ls f*1
f01  f11
```

There is another way of representing files or directories that follow a pattern, execute this command:

```
$> ls f0[3,5,7]
f03  f05  f07
```

The files selected are those whose last character is on the list [3, 5, 7]. Similarly, a range of characters can be represented. See:

```
$> ls f0[3-7]
f03  f04  f05  f06  f07
```

We will use those special character to move files based on its parity. Execute:

```
$> mv f[0,1][1,3,5,7,9] odd
$> mv f[0,1][0,2,4,6,8] even
```

The command above is equivalent to execute the explicit listing of sources:

```
$> mv f01 f03 f05 f07 f09 f11 odd
$> mv f02 f04 f06 f08 f10 even
```

2.3.5 Delete files and Folders

As we mentioned above, empty folders can be deleted with the command `rmdir` but that only works if there are no subfolders or files inside the folder that you want to delete. See for example what happens if you try to delete the folder called odd:

```
$> rmdir odd
rmdir: failed to remove `odd': Directory not empty
```

If you want to delete odd, you can do it in two ways. The command `rm` allows you to delete one or more files entered as arguments. Let's delete all the files inside odd, followed by the deletion of the folder odd itself:

```
$> rm odd/*
$> rmdir odd
```

Another option is to delete a folder recursively, this is a powerful but also dangerous option. Even if deleting a file is not actually filling with zeros the location of the data, on HPC systems the recovery of data is practice unfeasible. Let's delete the folder even recursively:

```
$> rm -r even
```

2.3.6 Summary of Basic Commands

The purpose of this brief tutorial is to familiarize you with the most common commands used in UNIX environments. We have shown 10 commands that you will probably use very often in your interactions. These 10 basic commands and one editor from the next section is all that you need to be ready for submitting jobs on the cluster.

The next table summarizes those commands.

Table 2: WVU's High-Performance Computer (HPC) Clusters

Command	Description	Examples
<code>echo</code>	Display a given message on the screen	\$> echo "This is a message"
<code>cat</code>	Display the contents of a file on screen Concatenate files	\$> cat my_file
<code>date</code>	Shows the current date on screen	\$> date Wed Nov 7 10:40:05 EST 2018
<code>pwd</code>	Return the path to the current working directory	\$> pwd /users/username
<code>cd</code>	Change directory	\$> cd sub_folder
<code>mkdir</code>	Create directory	\$> mkdir new_folder
<code>touch</code>	Change the access and modification time of a file Create empty files	\$> touch new_file
<code>cp</code>	Copy a file in another location. Copy several files into a destination directory	\$> cp old_file new_file
<code>mv</code>	Move a file in another location. Move several files into a destination directory	\$> mv old_name new_name
<code>rm</code>	Remove one or more files from the file system tree	\$> rm trash_file \$> rm -r full_folder

2.4 Terminal-based Text Editors (nano)

After learning a few commands the next thing to learn is to edit text files. As we will see later, editing small text files is a common procedure on an HPC cluster; for example, the submission scripts are relatively small text files, and most of the times, you have to make small adjustments to those scripts. Many scientific codes use text-based input files for running simulations. You need to know how to use at least one basic editor to help you with those tasks.

There are several editors available on modern UNIX/Linux machines. The most widely used are **vi**, **emacs**, and **nano**.

The standard *de facto* editor in UNIX is **vi** and a hacker favorite. The Single UNIX Specification (SUS) specifies **vi**, so every conforming system must have it.

The second most commonly used editor is **emacs**, sometimes not installed by default on many systems it is usually provided on most Linux distributions.

The third editor is **GNU nano** a very user-friendly editor. For this tutorial we will focus on **nano**, for short text files and simple edits, **nano** serves its purpose.

2.4.1 Opening GNU nano

On a terminal execute:

```
$> nano <name_of_file>
```

You will get a text application that looks like this:



2.4.2 Using GNU nano

The bottom two lines show the commands to use. The symbol ^ means press the **CONTROL KEY** (left or right) and keep it pressed until you press the command key. For example, to exit the editor we see **^X**, which tells us that we must hold the control key and then press X. The character in the command is not case sensitive so pressing **^X** is equivalent to **^x**.

Executing commands with nano will sometimes show st of subcommands; some of them will be written with an M (eg. M-C), in that case M indicates the **META** character, which corresponds to Alt on modern keyboards.

The command **^G** display a complete list of keystrokes available from the main window. To exit from the help page, execute **^X**

GNU nano offers a fairly complete set of basic utilities to edit files.

Search for text with **^W** and the text to find. The command **^_** allows you to go to a specific line number.

2.4.3 Copy, Cut and Paste

The command **^K** and **^U** only serve to cut and paste the line where the cursor is located. When you cut the text, its contents are stored in what is called the *cut buffer*. The command **^U** just takes the contents of the *cut buffer* and write its contents to the location of the cursor. As the contents of the *cut buffer* are not erased you can use **^U** to repeat the same contents as many times as you want.

For more advanced copy and paste, you need to learn how to select blocks of text before cutting them to the *cut buffer*. First, move the cursor to the start of the text you want to select, press the M-A key combination (Alt-A in modern keyboards) to mark the start, then move the cursor to the end of the section you want to select. Once you have marked the beginning and end of the text, the **^^** (Esc-6) and **^K** key combinations can be used to copy or cut it, respectively. In a command like **^^**, the meaning of that is press the *Control Key* followed by the *carret symbol* ^.

2.4.4 Quitting Nano

To quit nano, use the **^X** key combination. If the file you are working on has been modified since the last time you saved it, you will be asked to save the file first. Type **y** to save the file, or **n** to exit nano without saving the file.

2.4.5 Exercise

As an exercise, take the opportunity to write the source code of a simple parallel program, open nano and write the code below:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
```

(continues on next page)

(continued from previous page)

```

MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

// Get the name of the processor
char processor_name[MPI_MAX_PROCESSOR_NAME];
int name_len;
MPI_Get_processor_name(processor_name, &name_len);

// Print off a hello world message
printf("Hello world from processor %s, rank %d out of %d processors\n",
       processor_name, world_rank, world_size);

// Finalize the MPI environment.
MPI_Finalize();
}

```

The code is in C, a well known programming language for a variety of purposes, including scientific computing. In C, tabulations are not mandatory but helps the legibility of the code, pay special attention to semicolons at the end of each instruction. All lines starting with double slash // are comments, so you can ignore those lines if you want.

In the next chapter, we will show how to compile the code and execute it on the cluster using the queue manager.

2.5 Cluster Storage

On a desktop machine, you normally own the whole hard drive and you are free to use it up to the limit of physical drive. On a share resource like a cluster, limits need to be imposed to avoid one user taking too much space or creating too many files, that could prevent other users to store their data or could reduce the overall performance of the drive with too many files.

There are two important locations from the user perspective, \$HOME and \$SCRATCH those locations are stored on a high performance storage device and uses GPFS as their file system. Those locations are also shared between all compute nodes, so no matter on which machine you run, files on those locations are always available. In this quick start, we will describe briefly the purpose of each one of them.

2.5.1 Home Directory

Home directories on Spruce Knob and Thorny Flat are located in /users directory and they are the default login location for users. The default disk quota for each user is 10 GB. As such, we recommend that home directories be primarily used to store scripts, binary executables and all the software machinery to do your research, in cases where data is too large, the \$HOME folder should not be used for storing data as it could fill quickly. For research data, users should use Scratch space on our clusters.

The good thing about \$HOME folder, despite its limited size is that all the data is backed up on a daily basis via snapshots and tape. Users can retrieve up to 4 weeks of older files from /users/.snapshots/{date}/\$USER if needed. If files are not available in the snapshot, please contact helpdesk@hpc.wvu.edu and the HPC team will attempt to retrieve the data from tape.

Note: Users may check their storage quota via the following command:

```
quota
```

This command is actually an alias for the GPFS command to manage quotas on the filesystem, the actual command is:

```
/usr/lpp/mmfs/bin/mmlsquota --block-size M
```

The block size can be changed to K for Kilo Bytes, M for Mega Bytes, G for Giga Bytes and T for Tera Bytes.

It is important to pay attention not only to the storage size but also to the limit on the number of files that can be stored. Too many files have an impact of file system performance so its number needs to be limited. If you create too many reaching the limit you cannot create more files on that file system even if your quota size still shows allowance.

2.5.2 Scratch Directory

Scratch directories are located in `/scratch` directory and can be easily moved to use the `$SCRATCH` environment variable set-up by default for each user. You can go to your scratch folder with:

```
cd $SCRATCH
```

Scratch directories on our clusters are treated as **TRUE** scratch space: There is no user-defined quota for space. All users scratch directories share the same file space, which is set at 130 TB. When the scratch directory becomes full, files can be automatically deleted by date. To avoid loosing data, all users should use their scratch space as truly temporary storage. However, we will notify users well in advance for schedule file deletions to give users ample time to remove data to prevent data loss.

It is a good idea to create a symbolic link from your `$HOME` folder to go directly to your `$SCRATCH` folder, this command will do that:

```
ln -s $SCRATCH $HOME/scratch
```

Next time, you can just use the symbolic link:

```
cd ~/scratch
```

Note: Scratch data is NOT backed up.

2.5.3 Persistent Group Storage

Research groups may purchase long term persistent storage from the RC HPC. This storage allows users to keep data on the cluster that will not be removed. In addition, this storage utilizes the same GPFS file system so files can be written directly to this storage from jobs executing on the cluster.

Persistent storage must be purchased in 1 TB chunks. To purchase this storage, go to <https://helpdesk.hpc.wvu.edu>, select “Open a New Ticket”, and select “Research Data Depot Storage Purchase” from Help Topic. For more information, please contact helpdesk@hpc.wvu.edu.

In the past researchers were allowed to purchase space under `/group` folder. For groups who have purchased this storage, it can be accessed here:

```
cd /group/{group_name}
```

Note: Persistent group storage is NOT backed up.

Note: Persistent Group storage has been replaced by the new Research Data Depot Service

2.5.4 Archival Storage

Please see the Research Data Depot Service for archival storage options.

For additional information please contact helpdesk@hpc.wvu.edu.

2.6 Environment Modules

A shared resource like a cluster, several individuals could have different requirements about the compilers, libraries and scientific codes needed for their research. Environment Modules helps users choose which compilers, libraries and software they want to use and on which versions.

Modules can be loaded and unloaded during the same session. Modules are very useful when you need to use several scientific packages or compiling software under a variety of different compilers, libraries or versions of them.

2.6.1 Knowing the available modules

Execute this command to know which modules are available on the cluster:

```
module avail
```

This command will give a list of all modules centrally managed on the cluster. On Thonry Flat the list looks like this:

```
$ module avail

----- /usr/share/Modules/modulefiles -----
dot      module-git  module-info modules    null      use.own

----- /shared/modulefiles/tier0 -----
benchmarks/hpl/2.3_gcc48      libs/fftw/3.3.8_gcc82      libs/netcdf/4.x_intel18_
←mpi18
benchmarks/hpl/2.3_gcc82      libs/fftw/3.3.8_gcc82_ompi4  libs/netlib/3.8.0_intel18
dev/cmake/3.13                libs/fftw/3.3.8_intel18   libs/openblas/0.3.5_gcc48
dev/doxygen/1.8.15            libs/hdf5/1.10.5_gcc48   libs/openblas/0.3.5_gcc82
lang/gcc/8.2.0                 libs/hdf5/1.10.5_gcc48_ompi2  libs/xmlf90/1.5.4_gcc48
lang/go/1.12.4                libs/hdf5/1.10.5_gcc82   libs/xmlf90/1.5.4_gcc82
lang/intel/2018_u4              libs/hdf5/1.10.5_gcc82_ompi4  parallel/cuda/10.0.130
lang/intel/2019_u2              libs/hdf5/1.10.5_intel18   parallel/hwloc/1.10.1_
←gcc48
lang/intel/2019_u3              libs/hdf5/1.10.5_intel18_impi18  parallel/hwloc/1.10.1_
←gcc82
lang/java/jdk1.8.0_201          libs/libpsml/1.1.7_gcc82  parallel/hwloc/1.10.1_
←intel18
lang/julia/1.1.0                 libs/libxc/3.0.1_gcc48   parallel/hwloc/2.0.3_
←gcc82
lang/python/cpython_3.7.2_gcc82  libs/libxc/3.0.1_gcc82   parallel/hwloc/2.0.3_
←intel18
lang/python/intelpython_2.7.15   libs/libxc/3.0.1_intel18   parallel/mpich/3.3_gcc82
lang/python/intelpython_3.6.3     libs/libxc/4.2.3_intel18   parallel/mvapich2/2.3.1_
←gcc82
```

(continues on next page)

(continued from previous page)

lang/r/3.5.2	libs/libxc/4.3.4_intel18	parallel/openmpi/2.1.2_
↳ gcc48		
libs/atompaw/4.1.0.5_gcc48	libs/netcdf/4.1.1_gcc48	parallel/openmpi/2.1.6_
↳ gcc48		
libs/atompaw/4.1.0.5_intel18	libs/netcdf/4.x_gcc48	parallel/openmpi/2.1.6_
↳ gcc82		
libs/boost/1.70_gcc48_ompi216	libs/netcdf/4.x_gcc48_ompi2	parallel/openmpi/2.1.6_
↳ intel18		
libs/boost/1.70_gcc82_ompi216	libs/netcdf/4.x_gcc82	parallel/openmpi/3.1.3_
↳ intel18		
libs/boost/1.70_intel18	libs/netcdf/4.x_gcc82_ompi4	parallel/openmpi/4.0.0_
↳ gcc82		
libs/fftw/3.3.8_gcc48	libs/netcdf/4.x_intel18	parallel/ucx/1.5.0_gcc82
<hr/>		
----- /shared/modulefiles/tier1 -----		
↳ -----		
conda	singularity/2.5.2	
<hr/>		
----- /shared/modulefiles/tier2 -----		
↳ -----		
atomistic/abinit/8.10.2_gcc48	atomistic/espresso/6.4_intel18_seq	
atomistic/abinit/8.10.2_gcc82	atomistic/espresso/6.4_intel18_thd	
atomistic/abinit/8.10.2_gcc82_nompi	atomistic/gromacs/2016.6_cuda	
atomistic/abinit/8.10.2_intel18_gf	atomistic/gromacs/5.1.5_cuda	
atomistic/abinit/8.6.3_gcc48	atomistic/lammps/2018-12-12_gcc82	
atomistic/abinit/8.6.3_gcc48_ncdf411	atomistic/lammps/2018-12-12_gcc82_ompi2	
atomistic/abinit/8.8.4_gcc48	atomistic/namd/2.13_CUDA	
atomistic/abinit/8.8.4_gcc48_ncdf411	atomistic/vasp/5.4.4_intel18_seq	
atomistic/abinit/8.8.4_gcc82	atomistic/vasp/5.4.4_intel18_thd	
atomistic/amber/18_cuda	bioinformatics/emboss/6.6.0	
atomistic/amber/18_mpi	bioinformatics/gatk/4.1.0	
atomistic/amber/18_openmp	matlab/2018r2	
atomistic/elk/5.2.14_intel18	visual/graphviz/2.40.1_gcc82	
<hr/>		
----- /shared/modulefiles/tier3 -----		
↳ -----		
general_gcc82	general_intel18	

As you can see each module looks like following a tree representation, very much like a filesystem where different versions of the same software are clustered together and codes for similar purposes are under the same low level name.

For this exercise, lets start with the compiler.

2.7 Workload Manager (SLURM)

The workload manager is the software tool that makes a computer cluster to appear and work like a single entity rather than like a simple aggregate of computers on a network. WVU clusters now uses SLURM as workload manager and users must be familiar with a few commands to effectively use an HPC cluster.

2.7.1 Concepts

Before introducing the basic commands on SLURM we need to understand some concepts used here

Compute Nodes

A High-Performance Compute cluster (HPC cluster) is made of a collection of computers, the term used for each computer is “node”. These nodes are linked together through a fast network such as Gigabit Ethernet or Infiniband. In the particular case of Thorny Flat we use the Omni-Path Architecture (OPA) as the fast network fabric.

A cluster typically hosts multiple types of nodes:

- **management nodes:** These computers run management services, databases, monitoring tools, reporting applications, provisioning tools and other related services used by system administrators. Normal users have no direct access to these nodes.
- **login nodes:** These computers are the machines where users login into through SSH to submit jobs and check results. Users must never use login nodes for any intense computations
- **storage nodes:** These computers host user files in possibly multiple filesystems. We use dedicated storage systems running distributed filesystems such as GPFS.
- **compute nodes:** These are the computers where jobs run and where the computations actually take place. In the context of SLURM these machines are simply referred as “nodes”

Partitions

In SLURM, a partition is a set of compute nodes grouped logically based on either physical properties of the hardware or job scheduling policies. Compute nodes can belong to several partitions making the term partition a bit misleading. In other resource managers partitions are called “queues” which is a more appropriate term. In general compute nodes are grouped based on common features shared by the nodes such as the presence of GPUs or node memory (RAM). Another reason to create partitions is to manage jobs that run for a day from those that could run for up to a week.

Jobs

Jobs are the atomic structure of a workload manager like SLURM. A Job is made of one or more sequential steps, each step consisting in one or multiple parallel tasks that could be dispatched to multiple CPU cores on a single node or to several nodes on the cluster.

Sockets, CPU cores and Hyperthreading

On a desktop computer or laptop you will find a single processor also called Central Processing Unit (CPU). The CPU is the main chip responsible for most computational calculations taking place on the machine. Different from Desktop computers and laptops, on HPC compute nodes it is often the case to find two or four CPU chips. Each CPU is located into what is called a socket. A dual socket node is then a node with two CPU chips. Those CPUs are in general identical and the Operating System will distribute the workload among them.

Modern CPUs are made of multiple cores. A CPU core is a completely functional processing unit and several CPU cores are printed on a single chip. We call this CPUs multicore and almost all CPUs today are multicore.

Some CPUs are capable of “logically dividing” each CPU core into two hardware threads, a technology called Hyperthreading. Hardware threads are designed to hide the latencies of the memory and feed the compute units fast enough to keep them busy all the time. Hyperthreading can be activated or deactivated depending on the cluster, on the workload. Depending on the code running on the node hyperthreading can benefit or harm the performance.

2.7.2 The roles of a workload manager

A workload manager like SLURM serves two main roles: **Resource Manager** and **Job Scheduler**.

Resource Manager

The role of resource manager is to collect information about all the computers in the cluster, their characteristics and current state. A human equivalent for a resource manager is a mix of an accountant and a manager. The resource manager role is mainly responsible for gluing a HPC cluster to appear to users as a single entity rather than a pile of computers. A resource manager provides tools to execute tasks of a HPC cluster with several nodes, and nodes with several cores as simple to use as an individual computer. Consider for example executing the command to know the name of the computer where the command is executed, on a single machine you run:

```
$> hostname
trcis001.hpc.wvu.edu
```

If we want to execute the same command on 3 machines, we can use SLURM and execute:

```
$> srun -N3 hostname
srun: job 3410 queued and waiting for resources
srun: job 3410 has been allocated resources
tcocm102.hpc.wvu.edu
tcocm101.hpc.wvu.edu
tcocm100.hpc.wvu.edu
```

The command has been executed on 3 machines, the cluster is used as a single entity and we are not interested exactly on which machines the command runs as far as it executes on 3 different nodes. The whole purpose of using a HPC cluster is to have many computers to run and not being concerned exactly on which machine or machines the actual execution takes place.

When the amount of resources requested by all the jobs from all the users exceeds the amount of resources available we need a system to prioritize the execution of the different jobs. That is the role of a Job Scheduler.

Job Scheduler

The algorithms behind the prioritization of jobs can become fairly sophisticated. The resources available on a cluster are permanently changing and jobs are submitted permanently. The job scheduler has several objective functions including the maximal utilization of the cluster but also fairness among the users, preventing one user from monopolizing the cluster.

SLURM is a workload manager that takes both roles in its architecture. From the user point of view all that you need to know is a handful of SLURM commands. The SLURM commands that you will learn in this section will allow you to:

- Submit jobs to the cluster, both for interactive and non interactive jobs.
- Monitor the list of jobs running on the system
- Learn the status and extra information for a particular job
- Cancel jobs that have been submitted and they are either running or waiting in queue
- List the partitions on the cluster and their state

2.7.3 Gathering cluster information

The *sinfo* command on SLURM can be used to get an overview of the resources offered by the cluster. By default, *sinfo* lists the partitions that are available.

On WVU clusters, partitions with the prefix “comm” are community resources. Any HPC user can submit jobs to those partitions and the partitions were created differentiating the amount of RAM (small, medium [med], large and extra large [xl]), the walltime policy for the partition (day or week) and two community partitions with GPU nodes, one for interactive jobs (comm_gpu_inter) and another for non-interactive jobs running for upto a week (comm_gpu_week). The default queue is marked with a star (*) and it is called *standby*. Most compute nodes belong to this partition and jobs can run on it for up to 4 hours. The *standby* partition should be used preferentially except if you are certain that 4 hours is not enough time to complete the job.

The command *sinfo* will list all the partitions and the state of the nodes for each of them. A more summarized version can be obtained with the argument -s

PARTITION	AVAIL	TIMELIMIT	NODES(A/I/O/T)	NODELIST
standby*	up	4:00:00	82/82/3	167 taicm[001-009],tarcl100,tarcs[100,200-206,300-304],tbdcx001,tbmcs[001-011,100-103],tbpcm200,tbpcs001,tcbcx100,tcctx100,tcgcx300,tcocm[100-104],tcocs[001-064,100],tcocx[001-003],tcscm300,tjscl100,tjscm001,tmmcm[100-108],tngcm200,tpmcm[001-006],tsacs001,tsdcl[001-002],tsscl[001-002],ttmcm[100-101],tzec1[100-107],tzecs[100-115]
comm_small_day	up	1-00:00:00	57/8/0/65	tcocs[001-064,100]
comm_small_week	up	7-00:00:00	57/8/0/65	tcocs[001-064,100]
comm_med_day	up	1-00:00:00	1/4/0/5	tcocm[100-104]
comm_med_week	up	7-00:00:00	1/4/0/5	tcocm[100-104]
comm_xl_week	up	7-00:00:00	2/1/0/3	tcocx[001-003]
comm_gpu_inter	up	4:00:00	8/3/0/11	tbegq[200-202],tbmqq[001,100],tcogq[001-006]
comm_gpu_week	up	7-00:00:00	6/0/0/6	tcogq[001-006]
aei0001	up	infinite	0/8/1/9	taicm[001-009]
alromero	up	infinite	10/4/0/14	tarcl100,tarcs[100,200-206,300-304]
be_gpu	up	infinite	1/2/0/3	tbegq[200-202]
bvpopp	up	infinite	0/1/0/1	tbpcs001
cedumitrescu	up	infinite	0/0/1/1	tcctx100
cfb0001	up	infinite	0/1/0/1	tcbcx100
cgriffin	up	infinite	1/0/0/1	tcgcx300
chemdept	up	infinite	0/4/0/4	tbmcs[100-103]
chemdept-gpu	up	infinite	1/0/0/1	tbmqq100
cs00048	up	infinite	0/1/0/1	tcscm300
jaspeir	up	infinite	0/2/0/2	tjscl100,tjscm001
jbmertz	up	infinite	11/6/0/17	tbmcs[001-011,100-103],tbmqq[001,100]
mamclaughlin	up	infinite	0/9/0/9	tmmcm[100-108]
ngarapat	up	infinite	0/1/0/1	tngcm200

(continues on next page)

(continued from previous page)

pmm0026	up	infinite	0/6/0/6 tpmcm[001-006]
sbs0016	up	infinite	0/2/0/2 tsscl[001-002]
spdifazio	up	infinite	0/2/0/2 tsdcl[001-002]
tdmusho	up	infinite	0/2/0/2 ttmc[m100-101]
vyakberman	up	infinite	1/0/0/1 tsacs001
zbetienne	up	infinite	0/24/0/24 tzec1[100-107],tzecs[100-115]
zbetienne_large	up	infinite	0/8/0/8 tzec1[100-107]
zbetienne_small	up	infinite	0/16/0/16 tzecs[100-115]

Now you know the partitions on the cluster and based on your knowledge of the job you can decide on which partition submit your job. Now we will learn about the kinds of jobs that can be submitted and how to submit jobs.

2.7.4 Job Submission

The main purpose of using an HPC cluster is the execution of jobs. In particular jobs that due their characteristics are impractical to be executed on a normal desktop computer or laptop. Such is the case of jobs that could take several hours or use significant amount of resources like multiple CPU cores or memory.

As we learn above an HPC has a variety of computers with particular purposes. Computationally intense calculation must only take place on compute nodes. Login nodes, the computers you first reach when connected to the cluster should be spared from any intense workload as these computers serve several other users and running on them will slow the machine and prevent others from executing effectively even the most simple commands. Short post processing tasks are acceptable on login nodes. As a rule of thumb, if a task takes more than one core or last for more than a few minutes it should run on a compute node instead of a login node.

There are two kind of jobs that can be executed on an HPC cluster, interactive and non-interactive jobs. Interactive jobs are those where you receive resources for you to use in real time, very similar to the way you use your own computer. Interactive jobs are a good solution when you want to learn the steps needed to achieve the results you need. Later on you can write those steps in the form of scripts and let the computer to execute them in your absence.

Non-interactive jobs are the solution to jobs that take hours to execute or to run several jobs on the cluster. In non-interactive jobs you prepare a script, a recipe, indicating the computer, step by step, how to get the results that will allow you to take decisions later on or producing the final results for that level in your research.

Regardless of running, interactive or non-interactive jobs, SLURM, as workload manager, will decide on which machines (compute nodes) the jobs will run and will give you the tools to monitor the status of the jobs submitted. It is time to learn the basics of submitting interactive and non-interactive jobs.

A very simple way of launching an interactive job is using the command `srun`:

```
trcis001:~$ srun --pty bash
srun: job 22432 queued and waiting for resources
srun: job 22432 has been allocated resources
tzecs115:~$
```

Notice that `srun` is actually taking a double function. From one side is creating a new job (In the case above the job with ID=22432) followed by a remote terminal session on the machine assigned to the job. In the example above the job is requesting default values for all parameters. The partition is set to `standby` which offers a walltime of 4 hours. No selecting any number of nodes or cores will automatically assign a single core on a single machine.

In the case of needing more resources, maybe a different partition or number of cores add extra arguments to the command line:

```
trcis001:~$ srun -p standby -t 40:00 -c 4 --pty bash
```

In the example above we are explicitly selecting *standby* as partition, 40 minutes of walltime and 4 cores on a single compute nodes. The last argument in the srun command line must be the command to be executed. In this case, a bash session once logged into the assigned compute node.

The following is an example of a request for interactive job asking for 1 GPU 8 CPU cores for 2 hours:

```
trcis001:~$ srun -p comm_gpu_inter -G 1 -t 2:00:00 -c 8 --pty bash
```

You can verify the assigned GPU using the command *nvidia-smi*:

```
trcis001:~$ srun -p comm_gpu_inter -G 1 -t 2:00:00 -c 8 --pty bash
srun: job 22599 queued and waiting for resources
srun: job 22599 has been allocated resources
tbegqq200:~$ nvidia-smi
Wed Jan 18 13:27:01 2023
+-----+
| NVIDIA-SMI 515.43.04      Driver Version: 515.43.04      CUDA Version: 11.7      |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC  | | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M.  |
| |          |          |           |           |          |          MIG M. |
+=====+
|  0  NVIDIA A100-PCI... Off | 00000000:3B:00.0 Off |          0 | | |
| N/A   28C     P0    31W / 250W |      0MiB / 40960MiB |      0%  Default |
|          |          |           |           |          Disabled |
+-----+
+-----+
| Processes:                               |
| GPU  GI  CI      PID  Type  Process name        GPU Memory  |
|       ID  ID          |          |             Usage  |
+=====+
|  No running processes found            |
+-----+
```

The command above shows a NVIDIA A100 as the GPU assigned to us during the lifetime of the job.

In SLURM an interactive job can be launched with the command *salloc*:

```
trcis001:~$ salloc -N3
salloc: Pending job allocation 3506
salloc: job 3506 queued and waiting for resources
salloc: job 3506 has been allocated resources
salloc: Granted job allocation 3506
Loading git version 2.29.1 : dev/git/2.29.1
trcis001:~$
```

The command *salloc* will allocate resources (e.g. nodes or CPU cores), possibly with a set of constraints (e.g. number of processor per node or amount of memory per node). *salloc* will allocate the resources and spawn a shell in which the srun command is used to launch parallel tasks. Notice that *salloc* will return a shell on the same machine where the command *salloc* was executed.

2.8 File Transfer (Globus)

Due to a large amount of data users usually need to transfers to/from HPC systems, reliable, secure, and highly optimized transfer methods needs to be utilized. As a result, WVU offers dedicated Data Transfer Nodes (DTNs) at each location it supports. Transfers to our systems should only be done through one of these DTNs for several reasons:

- DTNs are dedicated to transferring data only and built specifically for allowing users to transfers data as quickly, efficiently, and securely as possible. Only DTN's are connected to [WVU's Science DMZ network](#) which is a dedicated network to science/research data across campus.
- Users who transfer data using a login node can put the login node under strain causing it to slow down affecting other users and even possibly cause the node to fail.

WVU's recommended method for transferring data is [Globus Online](#). The other supported method is sftp.

2.8.1 Data Transfer Nodes

Research Computing offers dedicated servers for transferring files at each one of its supported systems/locations (i.e. Spruce Knob, Thorny Flat, and the Data Depot). The following table describes the services available at each location as well as the associated Globus Endpoint Name or sftp Hostname.

	Spruce Knob	Thorny Flat	Data Depot	MIX Google Drive
Globus Endpoint	wvu#hpcdtn	wvu#thornydtn	wvu#datadepot	wvu#GoogleDrive
sftp Hostname	data.hpc.wvu.edu	tf-data.hpc.wvu.edu	datadepot-sftp.hpc.wvu.edu	NA

2.8.2 Using Globus Online

Globus Online is the preferred method for transferring files to, from, and between Research Computing Resources. WVU is also a [Globus Subscriber](#) which provides additional services over the basic/free subscription. Globus Online offers the following advantages over traditional transfer methods (i.e. scp, sftp, rsync):

- Auto performance tuning to ensure the data is transferred as quickly as possible. One can expect a speedup of at least 2x over traditional transfer methods.
- Safe transfers by ensuring data integrity using checksum methods.
- Transfers are automatically restarted after a failed or stopped connection.
- Ability to only transfer files that have yet to be transferred (similar to rsync).
- Transfers are done in the background so users do not need to remain logged into the system.

WVU's Globus Subscription adds the following features:

- Ability to archive/transfer data with unlimited storage to user's [Google Drive MIX Account](#).
- Sharing of data with others inside and outside the university.
- Sharing of data from personal workstations via [Globus Connect Personal](#).

Review <https://www.globus.org/researchers/getting-started> for a step by step guide on use Globus Online.

For more details, including written instructions, please visit this [page](#)

Note: For a video on how to utilize Globus Online with Google Drive, please see <https://www.youtube.com/watch?v=tDdVsNVK3ko&feature=youtu.be>.

2.8.3 Alternative Transfer Methods

To transfer files to Research Computing systems without Globus Online, users may choose to utilize sftp to the hostnames in the above table. Basic instructions on how to use SFTP can be found [here](#). For users who are more comfortable with Graphical User Interfaces, you may want to utilize one of the following clients:

- <https://winscp.net/eng/index.php>
- <https://filezilla-project.org/>

2.9 Web Interface (Open On-Demand)

The term Interactive Scientific Computing consists on using a computer in a similar way as you use a handheld calculator. On a handheld calculator, you type some input and you expect that input to be processed right away to produce results. The result does not need to return immediately, but you expect that calculations are done take an amount of time that allow a human to wait before a new command is submitted.

Interactive Scientific Computing is different from the way we use to work in High Performance Computing (HPC). On a HPC supercomputer, you use the computer with a queue system on what we called non-interactive computing. Supercomputers are large computing devices usually build as clusters of individual computers. Supercomputers in many cases are used by tens or hundreds of users at the same time. For the typical use of Supercomputers, the computations are programmed in advance, the user submit jobs expecting that those jobs start being executed sometime in the future and produce the result that will be analyzed later on.

Despite of the usual usage of a HPC supercomputing, you could have strong motivations for use a computer far more powerful than your own desktop or laptop. Your research have scaled to a point where your desktop computer or laptop is not longer capable of managing the task, you need specialized software packages and you do not want to spent time compiling or installing software and you would like to rely on software that is already present on the HPC cluster.

There are several ways to a HPC cluster for Interactive Computing. You can execute interactive computing directly from the terminal or using a web interface such as Open-On Demand and one interactive environments such as Jupyter or RStudio. We will describe both alternatives.

2.9.1 Interactive Computing from the Terminal

First, connect to the cluster using the instructions described on [here:ref:qs-connect::](#). Once on the head node you can request an interactive session with:

```
trcis001:~$ module load sched/slurm/22.05
trcis001:~$ srun --pty bash
```

This is the simplest command for requesting an interactive session. Under this command, the partition selected will be `standby` with a walltime of 4 hours. The job will be allocated on one compute node and one core on that node for the execution.

You can request more cores for your interactive job, for example:

```
trcis001:~$ srun -n 4 --pty bash
```

On Spruce Knob you can request 16 cores for an entire node. There are a few nodes with 20 and 24 cores but requesting with those numbers will reduce the chances of getting a node for your execution inmediately. On Thorny Flat most nodes have 40 cores. A way of requesting an entire node could be:

```
$> qsub -I -n
```

Requesting more than 1 core does not necessarily mean that your code will actually use those extra cores. It all depends on your code being able to work in parallel directly or indirectly. Directly means that the code uses some sort of parallelization such as multithreading such as pthreads or OpenMP, or multiprocessing such as MPI, or any other parallelization in languages such as R or Python that offer libraries for explicit parallelism. Indirectly parallelization means that underlying libraries such as FFTW or OpenBLAS could have been compiled with multithreading support meaning that codes that use those libraries could take advantage of parallelism.

You can also request GPUs for an interactive job, in that case you have to explicitly request a queue such as *comm_gpu_inter* that offers compute nodes with GPU cards and explicitly request the number of GPUs that you want to use. For example if you want to use one GPU card for your execution use:

```
trcis001:~$ srun --pty -p comm_gpu_inter -N 1 -n 8 -G 1 bash
```

After you submit your request for interactive jobs, you will have to wait a few minutes before you are assigned a compute node. After getting access to the compute node, you can load modules and execute commands as you were on a machine. There are several text-based environments for interactive computing.

One is R and you can access it with:

```
$> module load lang/r/4.2.0_gcc112
$> R

R version 4.2.0 (2022-04-22) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> sessionInfo()
R version 4.2.0 (2022-04-22)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Red Hat Enterprise Linux Server 7.9 (Maipo)

Matrix products: default
BLAS/LAPACK: /gpfs20/shared/software/libs/openblas/0.3.19_gcc112/lib/libopenblas_
  ↵skylakexp-r0.3.19.so

locale:
[1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8       LC_COLLATE=en_US.UTF-8
[5] LC_MONETARY=en_US.UTF-8   LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8      LC_NAME=C
[9] LC_ADDRESS=C              LC_TELEPHONE=C
```

(continues on next page)

(continued from previous page)

```
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats      graphics   grDevices utils      datasets  methods   base

loaded via a namespace (and not attached):
[1] compiler_4.2.0
>
```

Another text-based interactive environment is IPython, to use it execute for example:

```
$> module load lang/python/cpython_3.10.1_gcc112
Loading gcc version 11.2.0 : lang/gcc/11.2.0
Loading python version cpython_3.10.1_gcc112 : lang/python/cpython_3.10.1_gcc112

$> ipython
Python 3.10.1 (main, Dec 28 2021, 19:48:41) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.30.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import platform

In [2]: platform.machine()
Out[2]: 'x86_64'

In [3]: platform.version()
Out[3]: '#1 SMP Thu Mar 25 21:21:56 UTC 2021'

In [4]: platform.platform()
Out[4]: 'Linux-3.10.0-1160.24.1.el7.x86_64-x86_64-with-glibc2.17'

In [5]: platform.uname()
Out[5]: uname_result(system='Linux', node='trcis001.hpc.wvu.edu', release='3.10.0-1160.
        ↪24.1.el7.x86_64', version='#1 SMP Thu Mar 25 21:21:56 UTC 2021', machine='x86_64')

In [6]: platform.system()
Out[6]: 'Linux'

In [7]: platform.processor()
Out[7]: 'x86_64'
```

Matlab is another software that offers interactive environment. Usually MATLAB is accessed on a graphical interface but MATLAB can also work from a text-based interface:

```
$> module load lang/gcc/9.3.0 matlab/2021a
Loading gcc version 9.3.0 : lang/gcc/9.3.0

$> matlab -nodesktop
MATLAB is selecting SOFTWARE OPENGL rendering.
```

← < M A T L A B (R) > →

(continues on next page)

(continued from previous page)

```
Copyright 1984-2021 The MathWorks, Inc.  
R2021a (9.10.0.1602886) 64-bit (glnxa64)  
February 17, 2021  
  
To get started, type doc.  
For product information, visit www.mathworks.com.  
>>
```

2.9.2 Interactive Computing from a web interface

For taking advantage of WVU's High Performance Computing cluster for interactive scientific computing another alternative is from a web browser. On this lesson you will not have to learn Linux commands, you just need to execute one for the purpose of downloading all the materials for the tutorials but beyond that your interaction will take place on a friendly web interface. You do not have to manually submitting jobs or editing submission scripts, these are tasks very important for HPC but they will delegated for other lesson.

We will be using a tool, a web-based client portal, that hides all that complexity and allow you to start using powerful computers for your research from a web interface, with minimal effort and fast learning curve.

Several technologies are involved here and it is important to understand how those different pieces are interconnected.

Open OnDemand is a web-based client, based on the Ohio Supercomputer Center's proven "OSC On Demand" platform, that enables HPC centers to install and deploy advanced web and graphical interfaces for their users. HPC resources are accessible from a web browser without the user having to install any special software or plugin.

The path for this tutorial is as follows. First we will demonstrate how to access the open on demand portal. Next we will create Jupyter and RStudio sessions and opening a terminal and a file manager.

2.9.3 Accessing the Dashboard

First, go to [Thorny Flat On Demand Dashboard](#)

The first page you will see is asking for your credentials



Central Authentication Service

Requesting access for service:
WVU Services

Username

Password

LOGIN

[Need help?](#) [Manage your Login account.](#)

West Virginia University is an Equal Opportunity/Affirmative Action institution.

Copyright 2020 West Virginia University - [Privacy Policy](#)

After entering your credentials and using your DUO authentication you will land on the Open On Demand Dashboard:



RESEARCH COMPUTING

OnDemand provides an integrated, single access point for all of your HPC resources.

From this dashboard you can launch interactive jobs, open terminals and access a file manager, we will see each of those operations in the next sections.

2.9.4 Interactive applications

From the dashboard go to *Interactive Apps*. There are several options there, we will show 2 apps that are currently ready for being used. Jupyter Notebooks and RStudio.

Jupyter

For Jupyter click on *Interactive Apps > Jupyter Notebook*. A form is shown with all the options available to create the Jupyter session.

A good starting point is to select *CPython 3.7.4* as the Python version, select *standby* as the queue and *4 hours* as the wall time. There are options for alternative Python versions, queues and walltimes. A short description of each options is shown on the form.

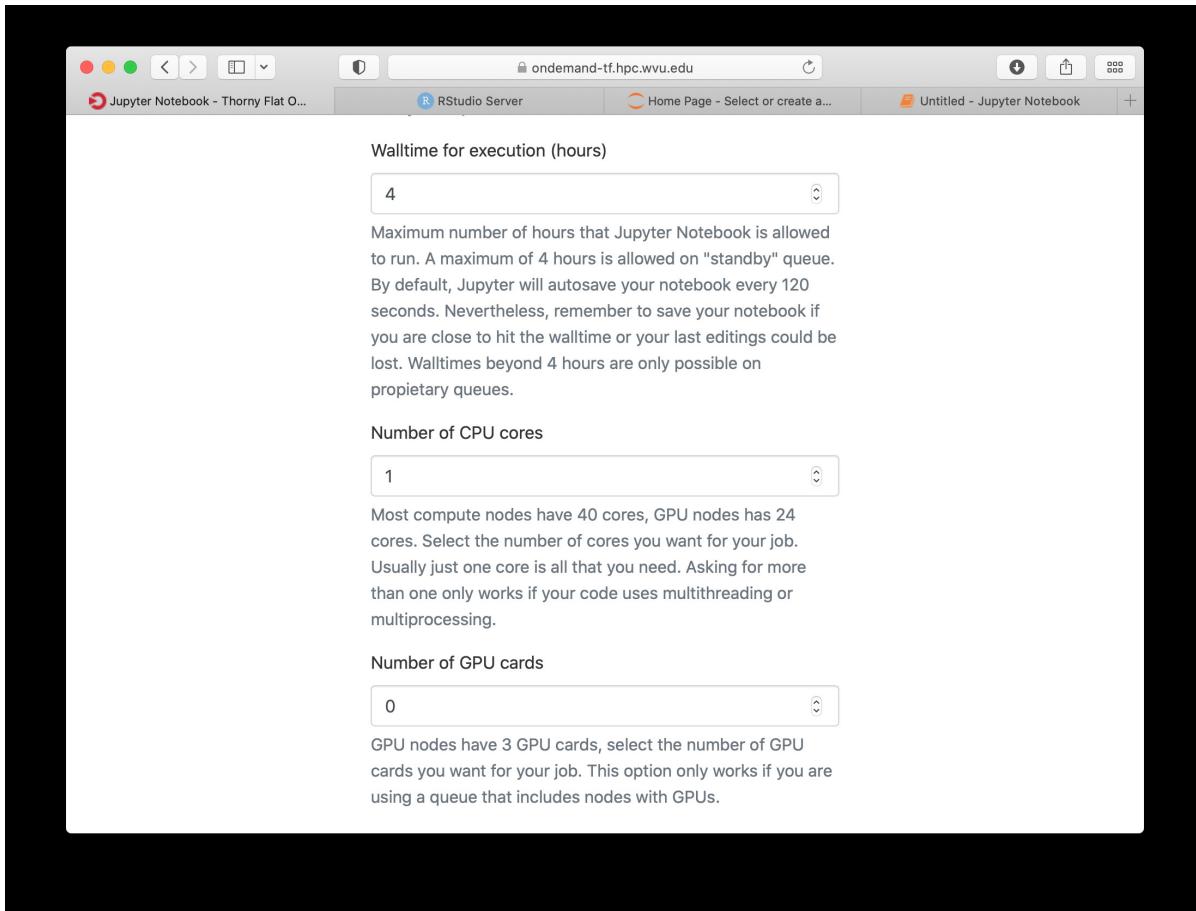
The screenshot shows a web browser window with the URL `ondemand-tf.hpc.wvu.edu`. The page title is "Jupyter Notebook - Thorney Flat O...". The main content area is titled "Jupyter Notebook" and contains the following information:

- Python Version:** Set to "CPython 3.10.1 GCC 11.2".
- Queue for Torque/Moab:** Set to "standby".
- Description:** "This app will launch a Jupyter Notebook server on a compute node."
- Queue Options:**
 - debug: max 1 hour, CPU only
 - standby: max 4 hours, CPU Only

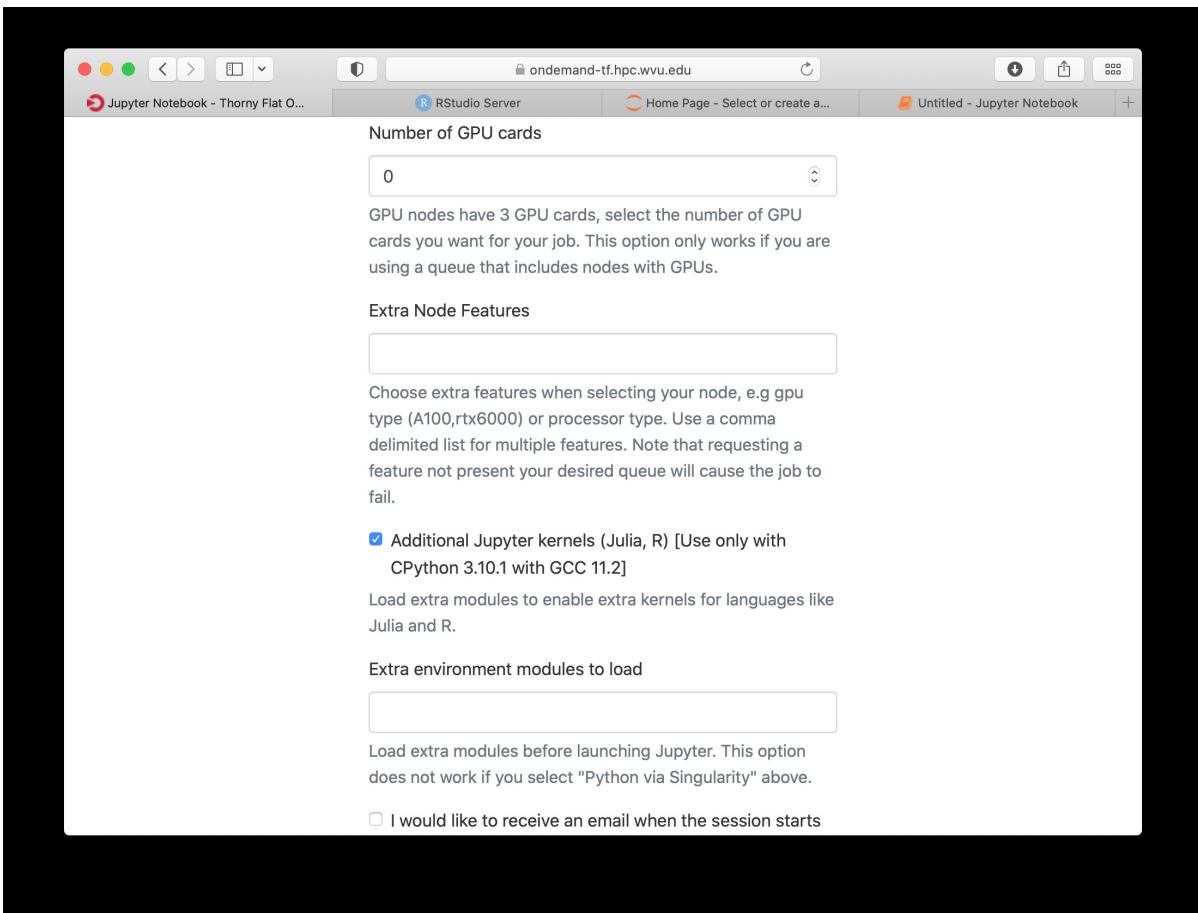
On the left sidebar, under "Interactive Apps", the "Jupyter Notebook" option is highlighted. Other listed options include ABAQUS, Thorney Desktop, Fermi-Lat, MATLAB, and RStudio Server.

The next fields on the form ask for the number of cores, GPU cards, extra modules and the singularity image in case you have selected that as your Python version.

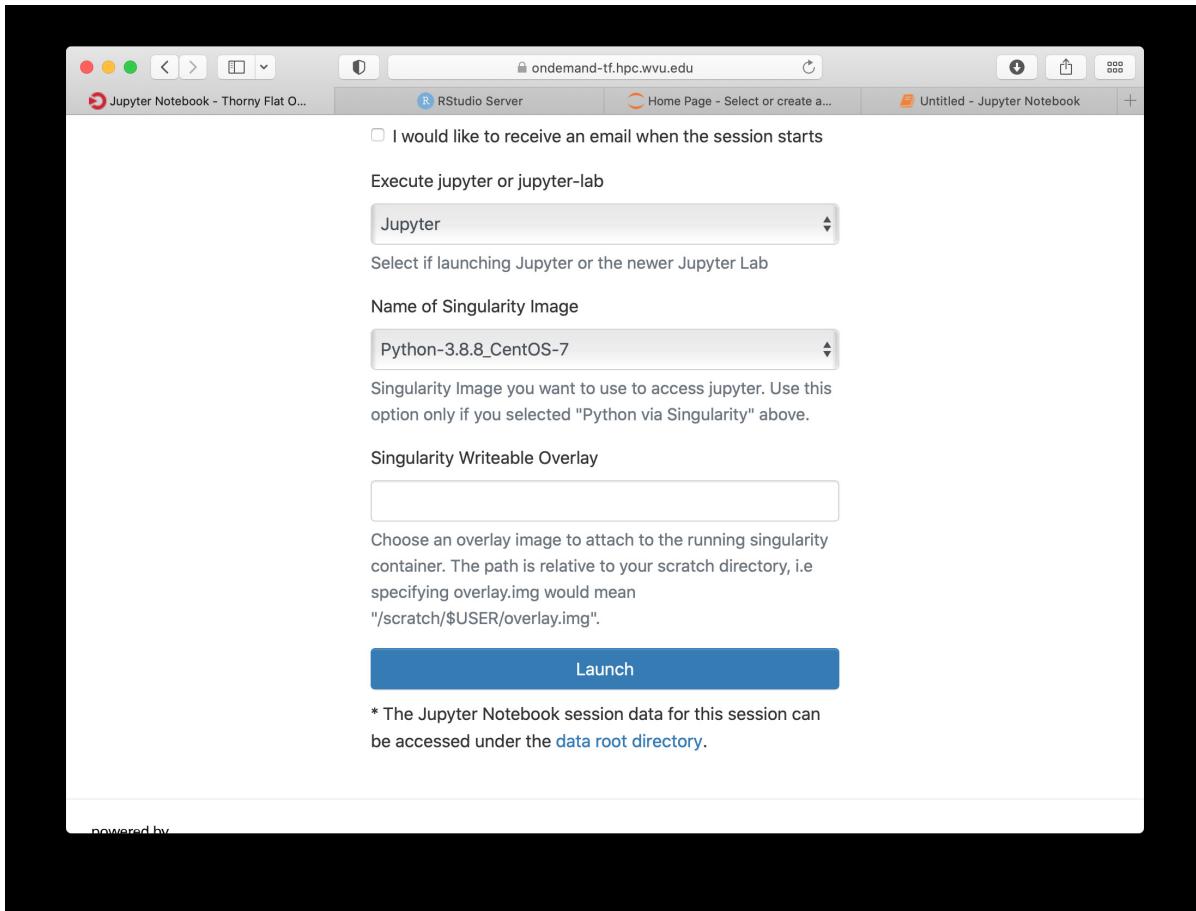
Notice that taking advantage of multiple cores depends on your code being able to use those cores. In the case of Python that usually means that your code is using *multiprocessing* module or you are using *numpy* with multi-threading capabilities. The usage of multiple cores is not something that happens automatically so if you are not sure asking for one core is enough. A similar situation happens with GPUs noticing that only the queue *comm_gpu_inter* give you access to GPUs for community nodes.



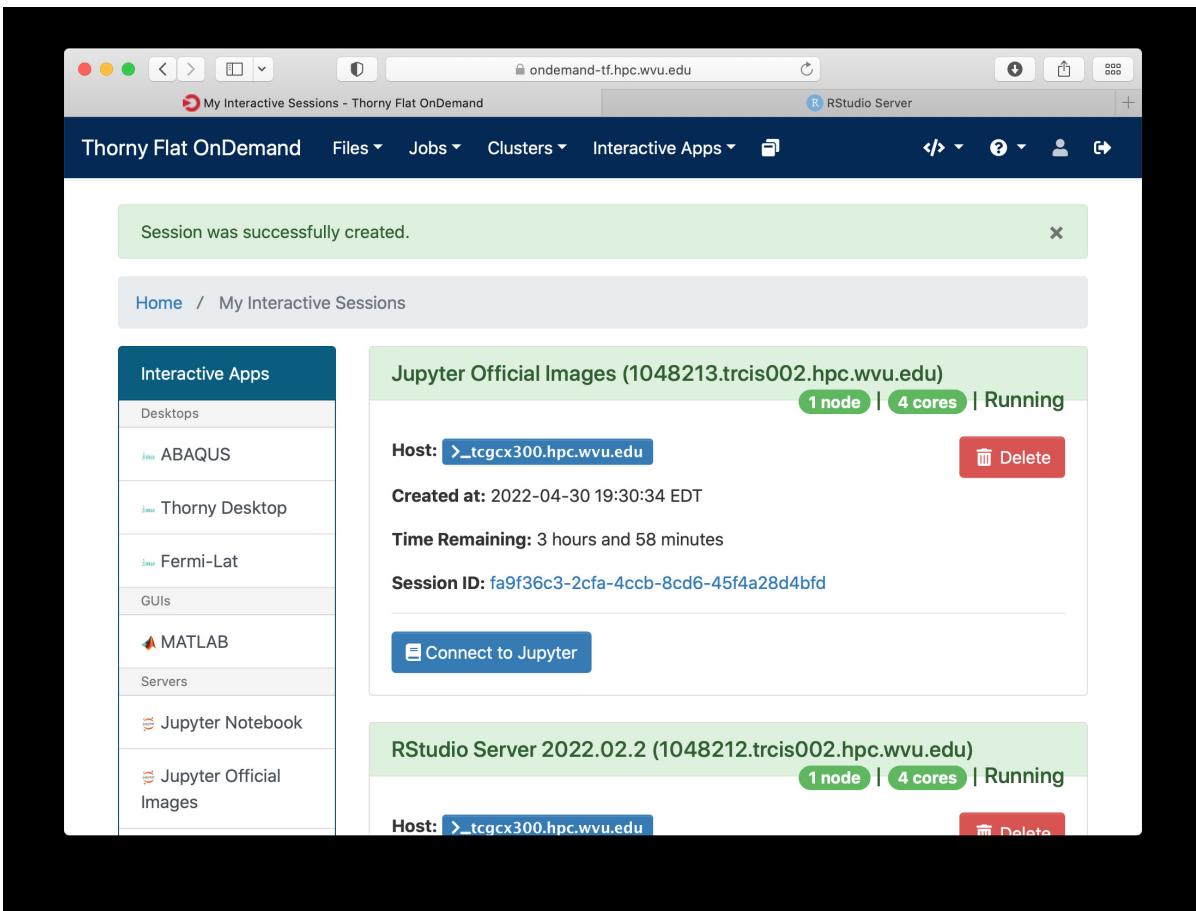
Once you have customize the parameters for your Jupyter session, click on *Launch*. Open On demand will launch a new interactive session and when the interactive session is launched you will get a button to connect to the Jupyter notebook.



The Jupyter session is launched on a compute node. The Jupyter interface shows as file manager where you can select a notebook to launch, upload one from your local computer or create a new Notebook, go to *New > Python 3* to create a new Jupyter notebook with Python 3 as kernel.



The new notebook give you entries for typing Python instructions that are executed when you type *SHIFT-ENTER*



The screenshot shows a Jupyter Notebook interface running on a Mac OS X system. The title bar indicates the session is on 'ondemand-tf.hpc.wvu.edu'. The notebook tab is titled 'Untitled - Jupyter Notebook'. The code cell contains the following Python script:

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t)) # white noise 1
nse2 = np.random.randn(len(t)) # white noise 2

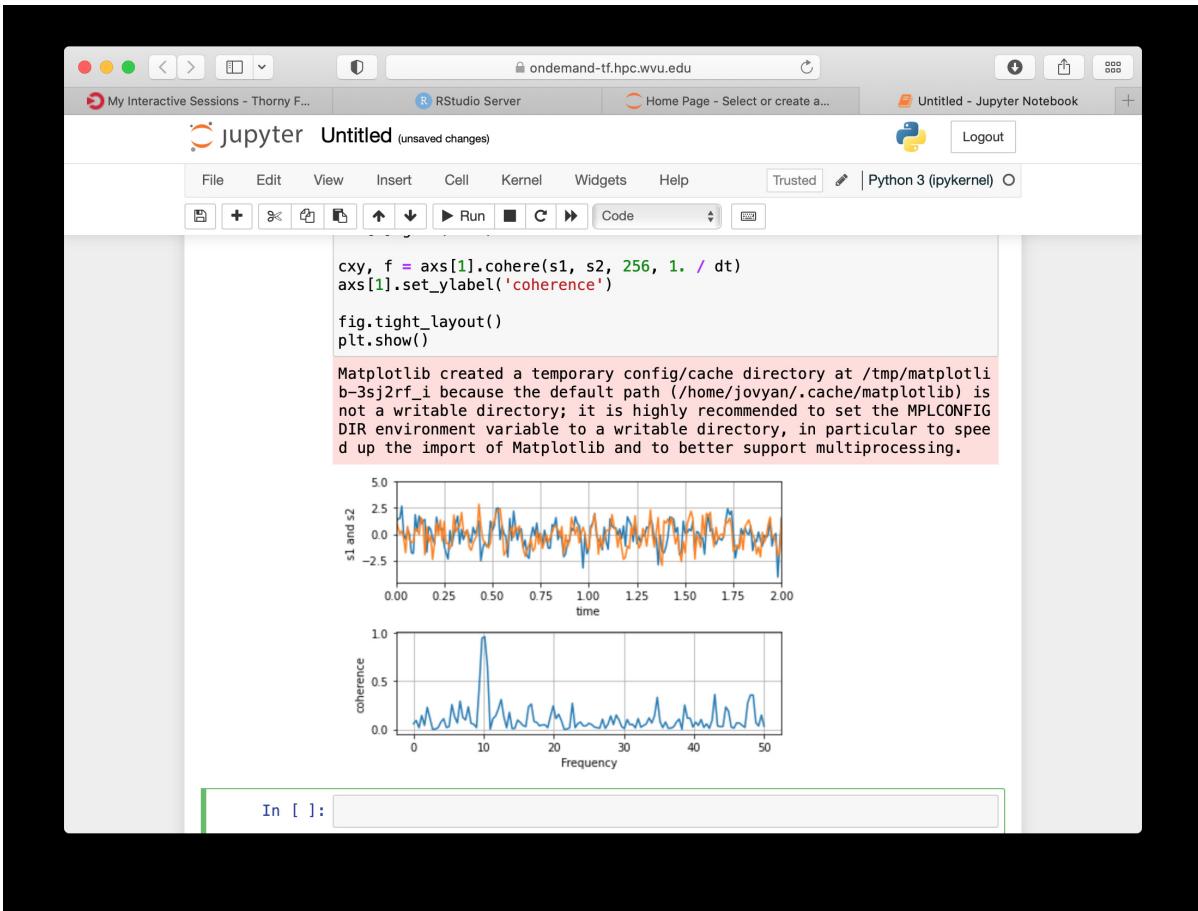
# Two signals with a coherent part at 10Hz and a random part
s1 = np.sin(2 * np.pi * 10 * t) + nse1
s2 = np.sin(2 * np.pi * 10 * t) + nse2

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, s1, t, s2)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('time')
axs[0].set_ylabel('s1 and s2')
axs[0].grid(True)

cxy, f = axs[1].cohere(s1, s2, 256, 1. / dt)
axs[1].set_ylabel('coherence')

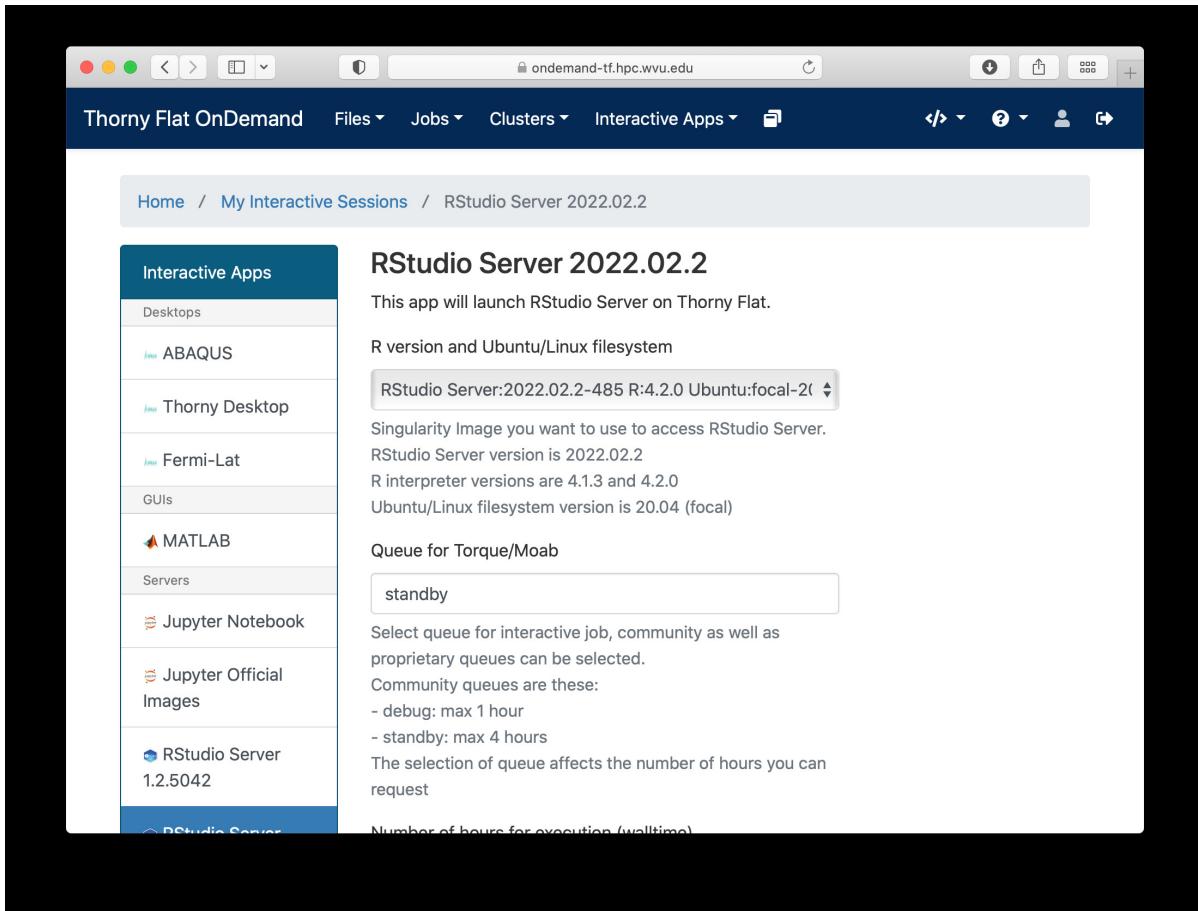
fig.tight_layout()
plt.show()

Matplotlib created a temporary config/cache directory at /tmp/matplotlib/b-3si2rf i because the default path (/home/joyvan/.cache/matplotlib) is
```

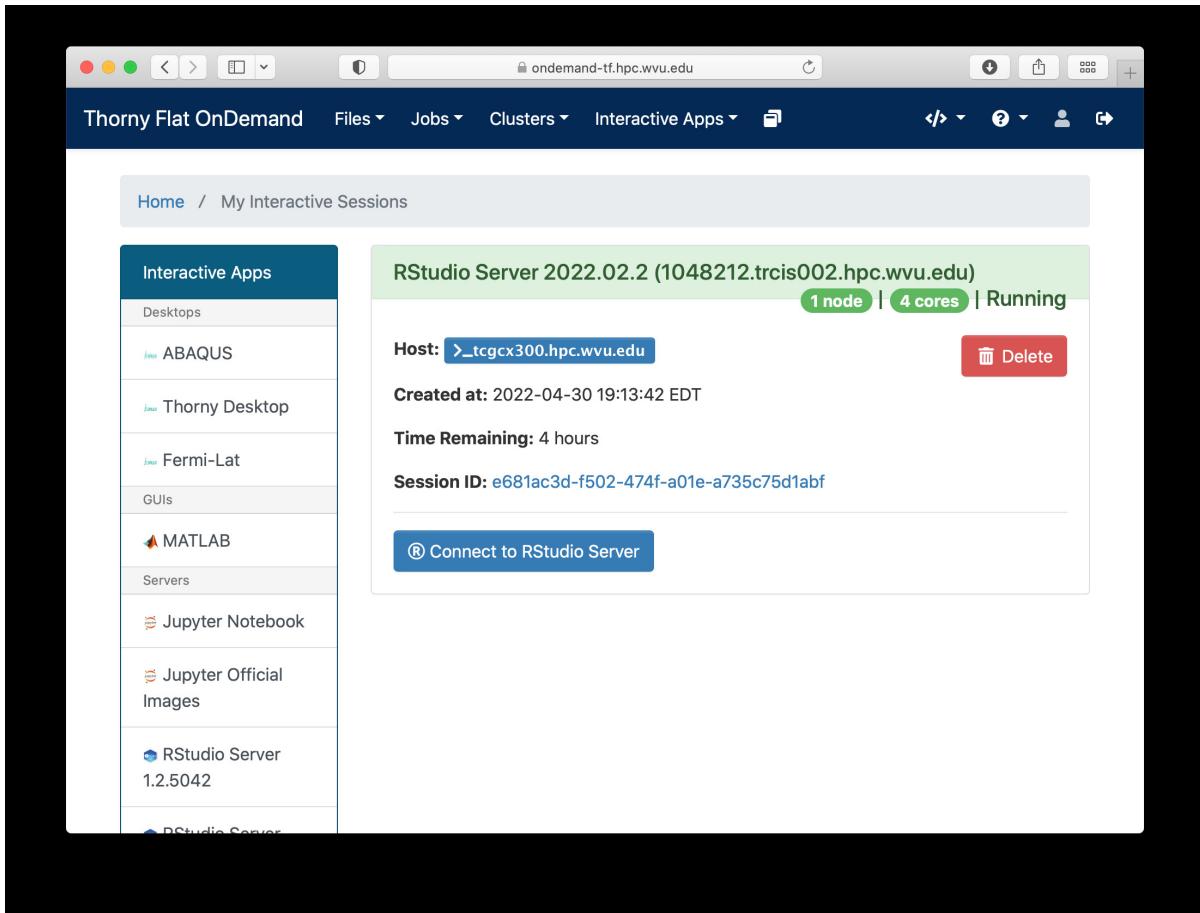


RStudio

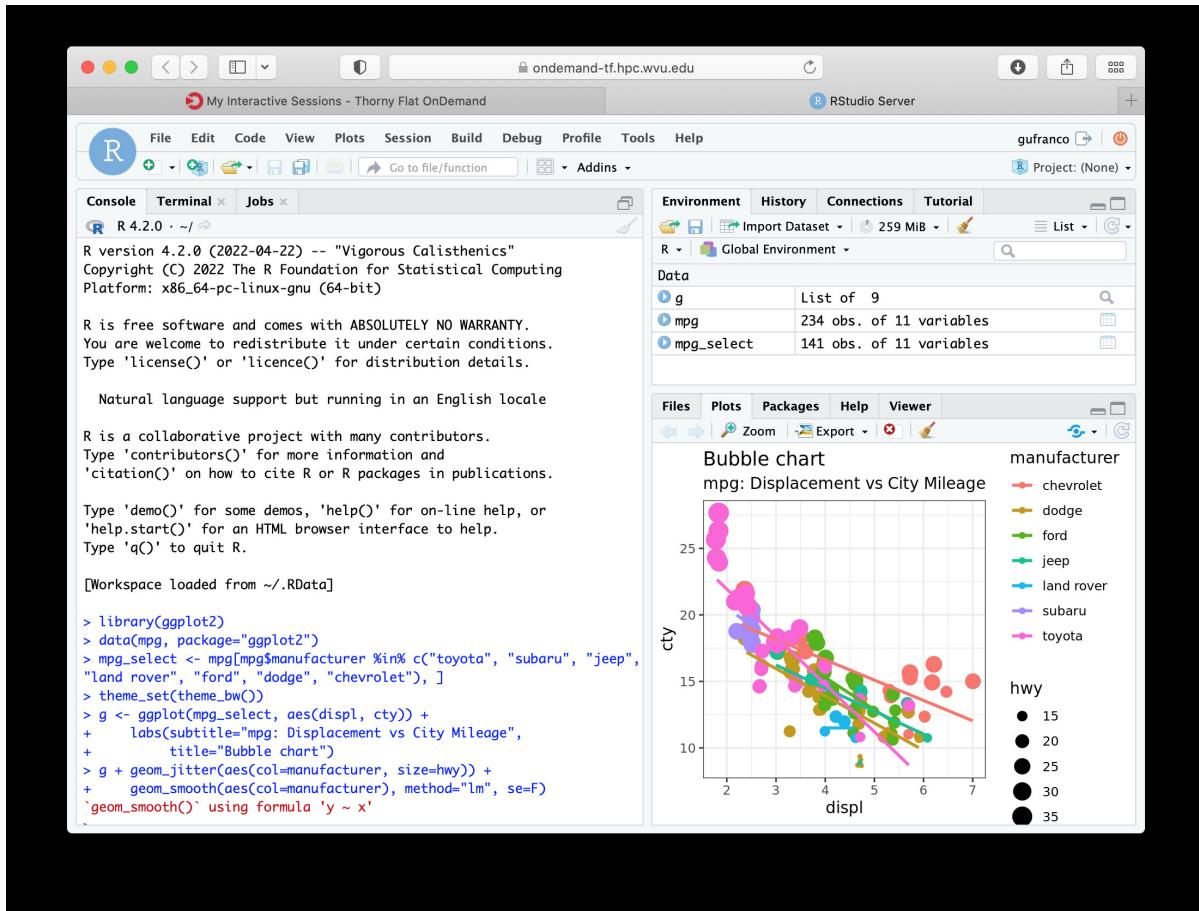
Another popular interactive environment is RStudio, select *Interactive Apps > RStudio*. The options in the form are very few. Select a queue such as *standby*, 4 hours of wall time and 1 core.



When you click on Launch, Open on Demand will create a new interactive session on Thony Flat and when the job starts execution, a button appears to open the session on a new tab.

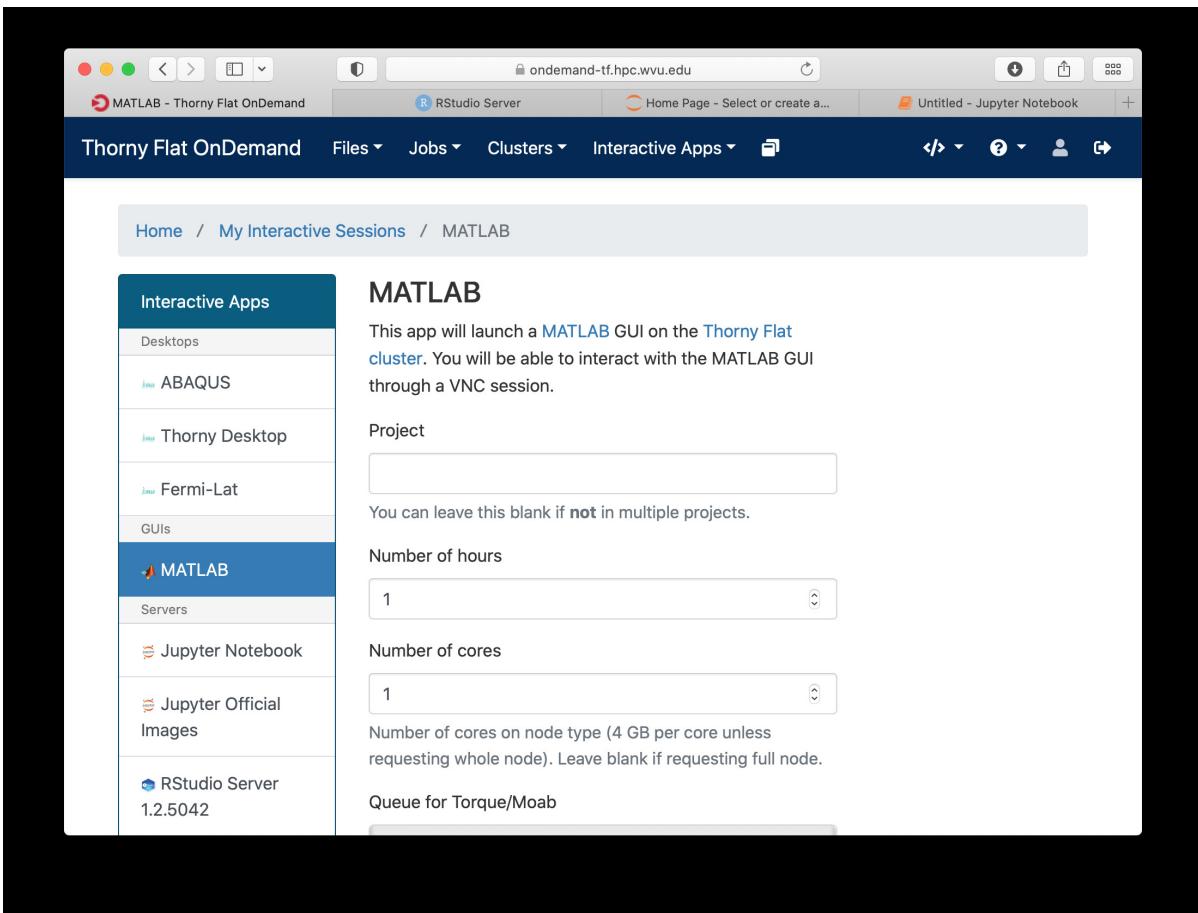


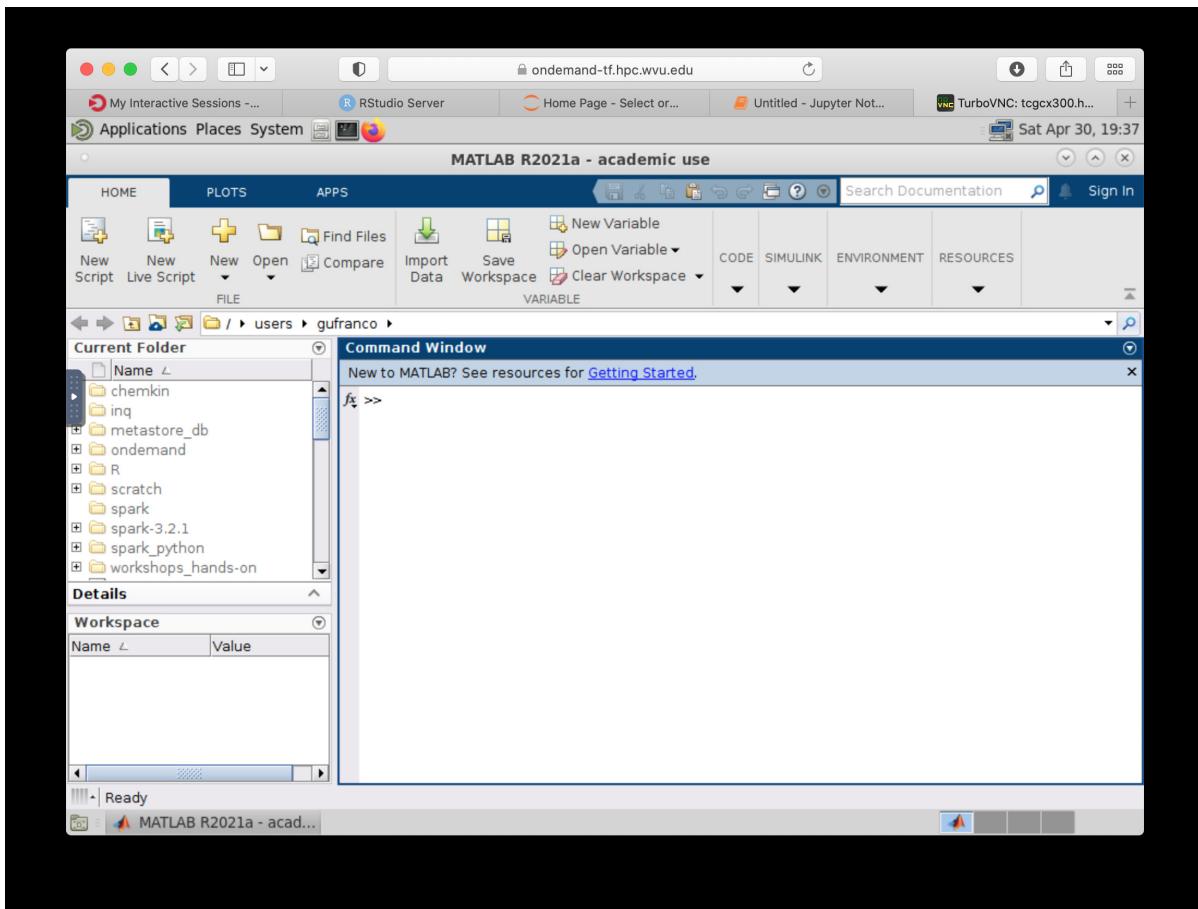
The interface for RStudio shows the commands on the left and variables and plots on the right.

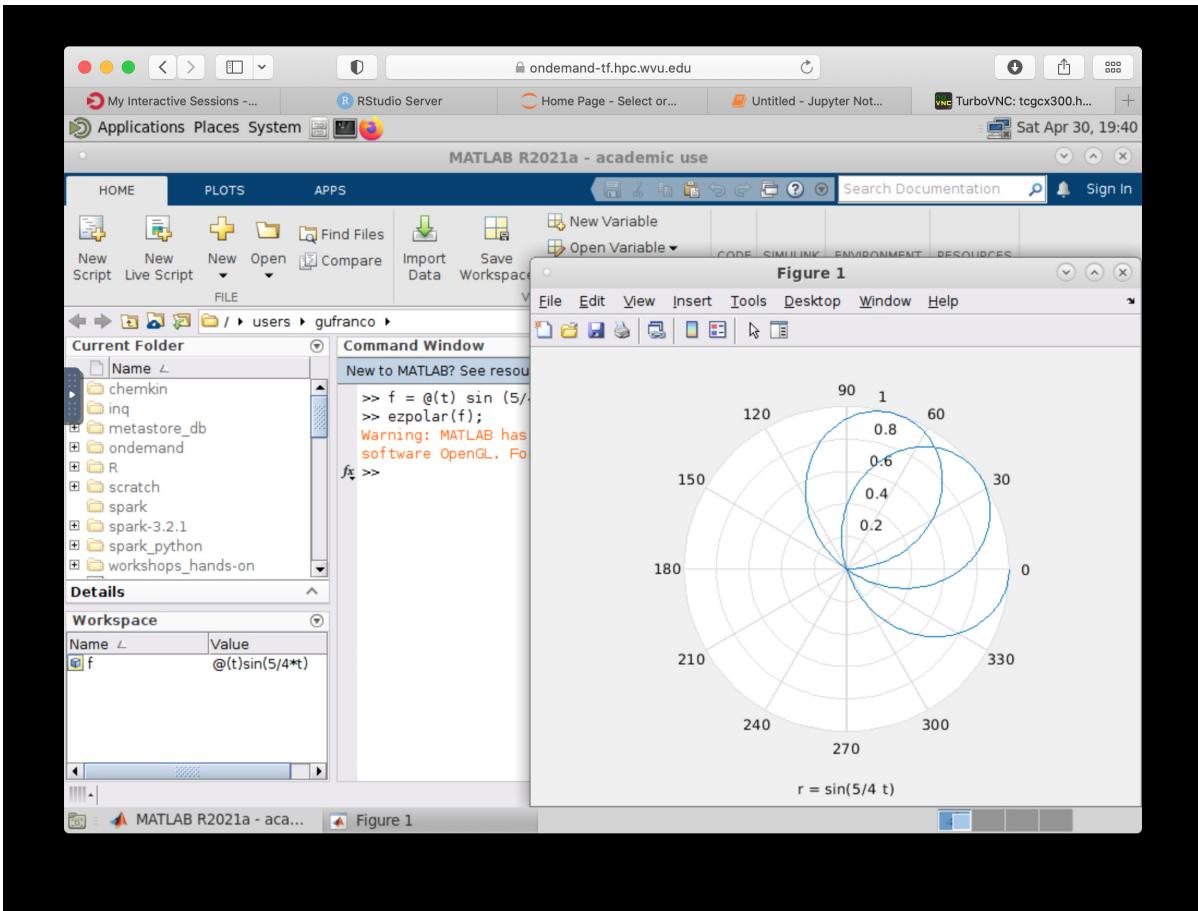


2.9.5 MATLAB

MATLAB is a numeric computing environment and programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. MATLAB can run using Open On-Demand via an app that executes a virtual desktop on a browser tab.



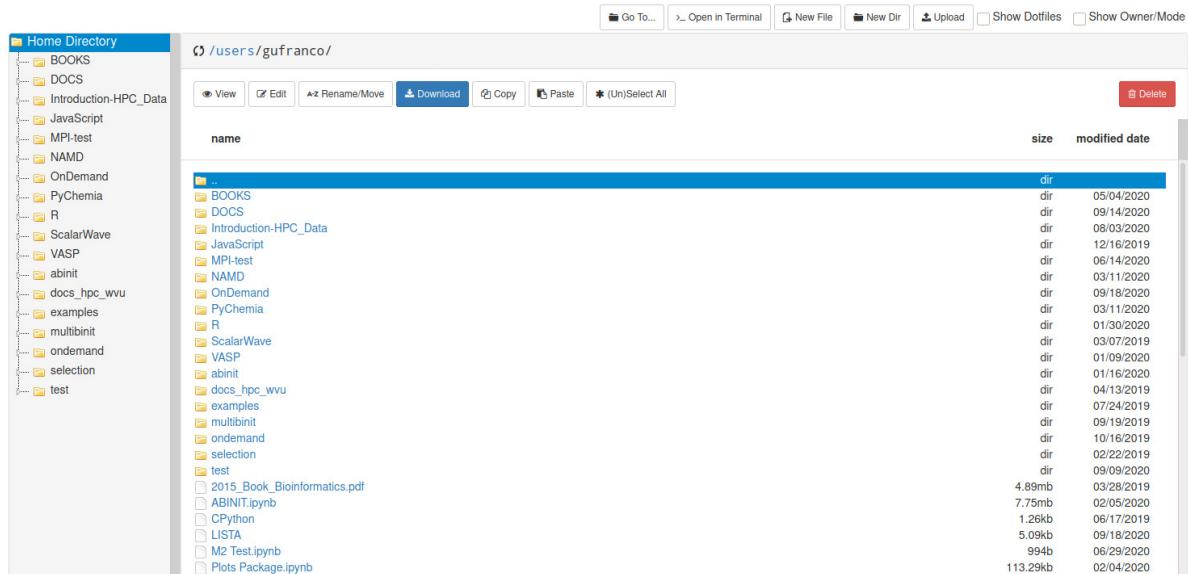




2.9.6 File Manager

The Open On demand dashboard also offers a simple but useful File Manager that give you options to view, edit, download and rename files. It is a simple way to see plots and download individual files to your local computer.

To access the File Manager on the Dashboard, go to *Files > Home Directory*. The File Manager is shown as



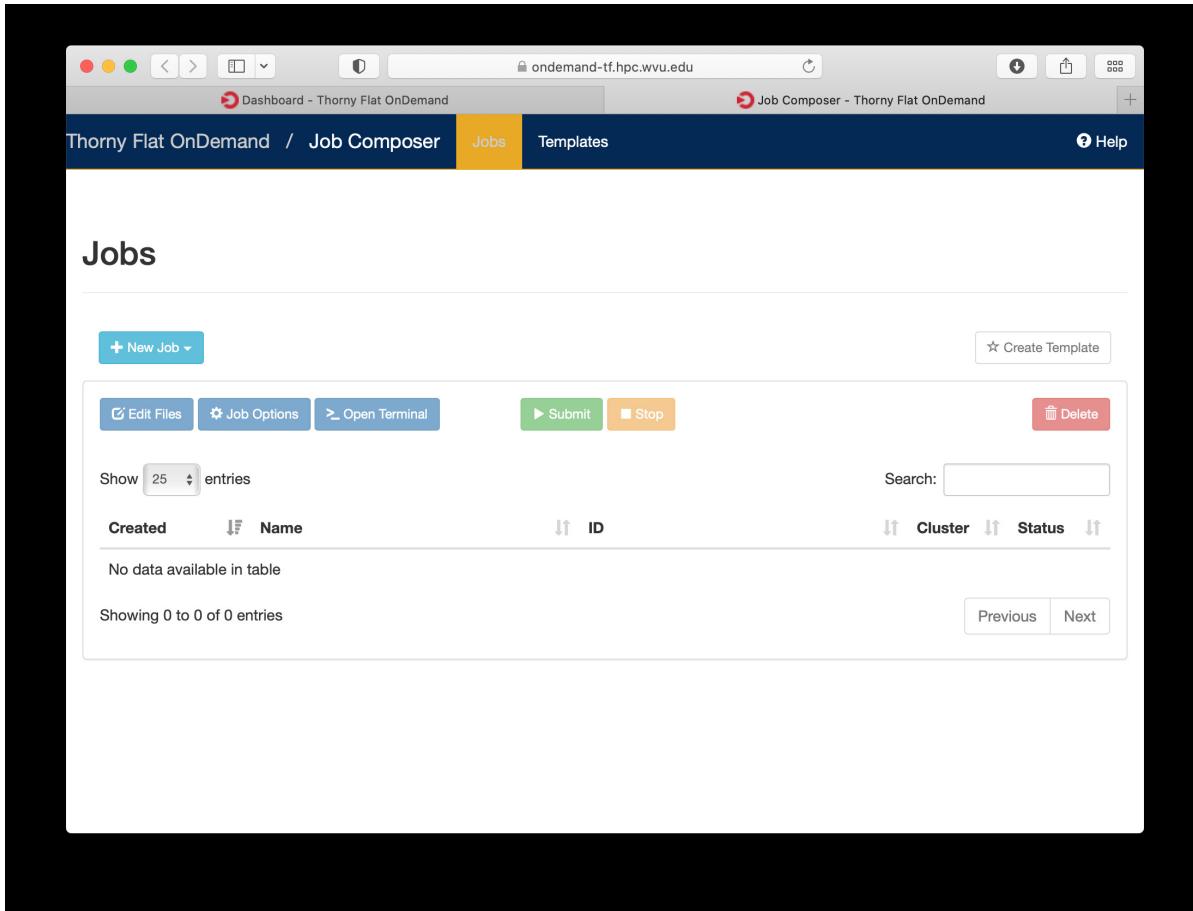
2.9.7 Job manager

Another useful tool integrated with the Dashboard is the Job Manager, you can see the jobs currently submitted on the cluster. Go to *Jobs > Active Jobs* to access the Job Manager

ID	Name	User	Account	Time Used	Queue	Status	Cluster
1048212.trcis002.hpc.wvu.edu	sys/dashboard/ sys/RStudio_2022	gufranco		00:27:11	standby	Running	Thorny Flat
1048213.trcis002.hpc.wvu.edu	sys/dashboard/ sys/Jupyter_Official	gufranco		00:10:19	standby	Running	Thorny Flat
1048214.trcis002.hpc.wvu.edu	sys/dashboard/ sys/bc_matlab	gufranco		00:05:17	standby	Running	Thorny Flat

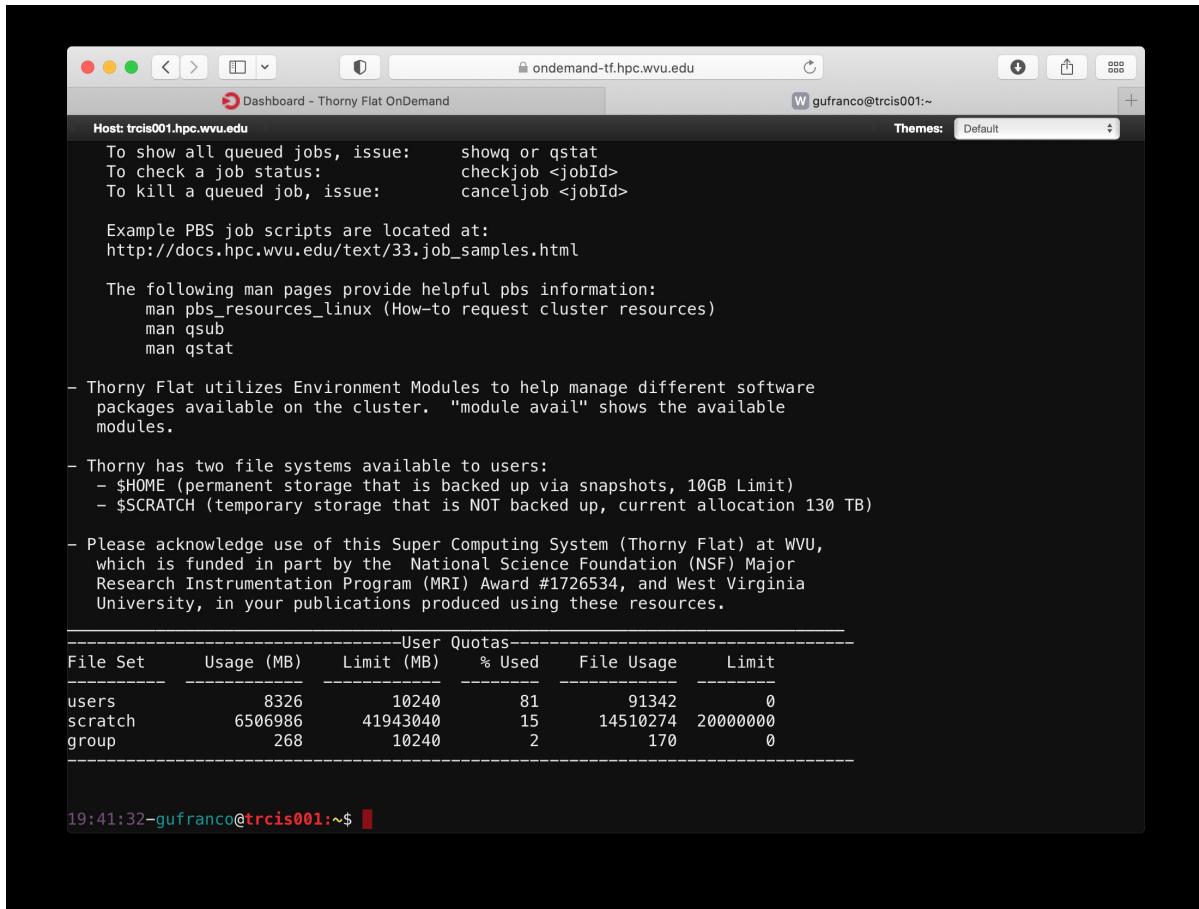
2.9.8 Job composer

Another useful tool integrated with the Dashboard is the Job Composer, you can create jobs and submit them to the cluster via a web interface. Go to *Jobs > Jobs Composer* to access the tool.



2.9.9 Terminal

Finally a Terminal session can be opened from the dashboard, the terminal runs on the head node exactly as a normal connection to the cluster via SSH. It has, however a short limit of time to be alive if no command is typed on it. The terminal is intended for quick commands only.



BASIC USAGE

This chapter provides information useful by users from all levels, but specially tailored for beginners or those who will use the cluster only sporadically.

We start on *Terminal-based Text Editors* a more in-deep exploration of the 3 main text editors in Unix/Linux: nano, emacs and vim. The next section

3.1 Terminal-based Text Editors

There are several terminal-based editors available on our clusters. We will focus our attention on three of them: nano, emacs, and vim. Your choice of an editor depends mostly on how much functionality do you want from your editor, how many fingers do you want to use for a given command and the learning curve to master it. For HPC users the editor is an important choice. Most of your time you are on the terminal executing commands or editing files, being those input files, submission scripts or the output of your calculations.

Let's review those three editors to give you the opportunity to have an informed choice.

3.1.1 Nano

Nano is a small, free and friendly editor with commands that usually manage using the Control (CTRL) combined with some other key.

You can start editing a file using a command line like this:

```
nano myfile.f90
```

There are several commands available, the list below comes from the help text. When you see the symbol “^” it means to press the Control (CTRL) key, the symbol “M-” is called Meta, but in most keyboards is identified with the (Alt) key.

^G (F1)	Display this help text
^X (F2)	Close the current file buffer / Exit from nano
^O (F3)	Write the current file to disk
^J (F4)	Justify the current paragraph
.	
^R (F5)	Insert another file into the current one
^W (F6)	Search for a string or a regular expression
^Y (F7)	Move to the previous screen
^V (F8)	Move to the next screen
.	
^K (F9)	Cut the current line and store it in the cutbuffer

(continues on next page)

(continued from previous page)

^U	(F10)	Uncut from the cutbuffer into the current line
^C	(F11)	Display the position of the cursor
^T	(F12)	Invoke the spell checker, if available
.		
^_	(F13) (M-G)	Go to line and column number
^\ \\	(F14) (M-R)	Replace a string or a regular expression
^[^]	(F15) (M-A)	Mark text at the cursor position
.	(F16) (M-W)	Repeat last search
.		
M-^	(M-6)	Copy the current line and store it in the cutbuffer
M-}		Indent the current line
M-{		Unindent the current line
.		
^F		Move forward one character
^B		Move back one character
^Space		Move forward one word
M-Space		Move back one word
^P		Move to the previous line
^N		Move to the next line
.		
^A		Move to the beginning of the current line
^E		Move to the end of the current line
M-((M-9)	Move to the beginning of the current paragraph
M-)	(M-0)	Move to the end of the current paragraph
M-\	(M-)	Move to the first line of the file
M-/	(M-?)	Move to the last line of the file
.		
M-]		Move to the matching bracket
M--	(M-_)	Scroll up one line without scrolling the cursor
M-+	(M-=)	Scroll down one line without scrolling the cursor
.		
M-<	(M-,)	Switch to the previous file buffer
M->	(M-.)	Switch to the next file buffer
.		
M-V		Insert the next keystroke verbatim
^I		Insert a tab at the cursor position
^M		Insert a newline at the cursor position
^D		Delete the character under the cursor
^H		Delete the character to the left of the cursor
M-T		Cut from the cursor position to the end of the file
.		
M-J		Justify the entire file
M-D		Count the number of words, lines, and characters
^L		Refresh (redraw) the current screen
.		
M-X		Help mode enable/disable
M-C		Constant cursor position display enable/disable
M-O		Use of one more line for editing enable/disable
M-S		Smooth scrolling enable/disable
M-P		Whitespace display enable/disable
M-Y		Color syntax highlighting enable/disable

(continues on next page)

(continued from previous page)

M-H	Smart home key enable/disable
M-I	Auto indent enable/disable
M-K	Cut to end enable/disable
M-L	Long line wrapping enable/disable
M-Q	Conversion of typed tabs to spaces enable/disable
M-B	Backup files enable/disable
M-F	Multiple file buffers enable/disable
M-M	Mouse support enable/disable
M-N	No conversion from DOS/Mac format enable/disable
M-Z	Suspension enable/disable

The most basic usage is to edit a file, and exit from the editor with CTRL-X. Nano ask you if you want to save the file, you answer “Y” and offers you a name. Simply press ENTER and your file is saved.

3.1.2 Emacs

Emacs is an extensible, customizable, open-source text editor. Together with Vi/Vim is one the most widely used editors in Linux environments. There are a big number of commands, customizations and extra modules that can be integrated with Emacs. We will just go briefly covering the basics.

The number of commands for Emacs is large, here the basic list of commands for editing, moving and searching text.

Entering Emacs

:: emacs

Leaving Emacs

suspend Emacs (or iconify it under X) C-z
exit Emacs permanently C-x C-c

Files

read a file into Emacs C-x C-f
save a file back to disk C-x C-s
save all files C-x s
insert contents of another file into this buffer C-x i
replace this file with the file you really want C-x C-v
write buffer to a specified file C-x C-w
toggle read-only status of buffer C-x C-q

Incremental Search

```
search forward C-s
search backward C-r
regular expression search C-M-s
reverse regular expression search C-M-r
select previous search string M-p
select next later search string M-n
exit incremental search RET
undo effect of last character DEL
abort current search C-g
Use C-s or C-r again to repeat the search in either direction. If
Emacs is still searching, C-g cancels only the part not matched.
```

Motion

```
entity to move over backward forward
character C-b C-f
word M-b M-f
line C-p C-n
go to line beginning (or end) C-a C-e
sentence M-a M-e
paragraph M-{ M-}
page C-x [ C-x ]
sexp C-M-b C-M-f
function C-M-a C-M-e
go to buffer beginning (or end) M-< M->
scroll to next screen C-v
scroll to previous screen M-v
scroll left C-x <
scroll right C-x >
scroll current line to center, top, bottom C-l
goto line M-g g
goto char M-g c
back to indentation M-m
```

Killing and Deleting

```
entity to kill backward forward
character (delete, not kill) DEL C-d
word M-DEL M-d
line (to end of) M-0 C-k C-k
sentence C-x DEL M-k
sexp M-- C-M-k C-M-k
kill region C-w
copy region to kill ring M-w
```

```
kill through next occurrence of char M-z char
yank back last thing killed C-y
replace last yank with previous kill M-y
```

Marking

```
set mark here C-@ or C-SPC
exchange point and mark C-x C-x
set mark arg words away M-@
mark paragraph M-h
mark page C-x C-p
mark sexp C-M-@
mark function C-M-h
mark entire buffer C-x h
```

Query Replace

```
interactively replace a text string M-%
using regular expressions M-x query-replace-regexp
Valid responses in query-replace mode are
```

```
replace this one, go on to next SPC or y
replace this one, don't move ,
skip to next without replacing DEL or n
replace all remaining matches !
back up to the previous match ^
exit query-replace RET
enter recursive edit (C-M-c to exit) C-r
```

Formatting

```
indent current line (mode-dependent) TAB
indent region (mode-dependent) C-M-\ 
indent sexp (mode-dependent) C-M-q
indent region rigidly arg columns C-x TAB
indent for comment M-;
insert newline after point C-o
move rest of line vertically down C-M-o
delete blank lines around point C-x C-o
join line with previous (with arg, next) M-^
delete all white space around point M-\ 
put exactly one space at point M-SPC
fill paragraph M-q
set fill column to arg C-x f
set prefix each line starts with C-x .
```

```
set face M-o
```

Case Change

```
uppercase word M-u  
lowercase word M-l  
capitalize word M-c  
uppercase region C-x C-u  
lowercase region C-x C-l
```

3.1.3 Vi/Vim

The third editor widely supported on Linux systems is “vi”. Over the years since its creation, vi became the *de-facto* standard Unix editor. The Single UNIX Specification specifies vi, so every conforming system must have it.

vi is a modal editor: it operates in either insert mode (where typed text becomes part of the document) or normal mode (where keystrokes are interpreted as commands that control the edit session). For example, typing i while in normal mode switches the editor to insert mode, but typing i again at this point places an “i” character in the document. From insert mode, pressing ESC switches the editor back to normal mode.

Vim is an improved version of the original vi, it offers

Here is a summary of the main commands used on vi. On Spruce when using “vi” you are actually using “vim”.

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text:

```
vi filename edit filename starting at line 1  
vi -r filename recover filename that was being edited when the system crashed
```

To Exit vi

Usually, the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file. Note: The cursor moves to bottom of the screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key:

```
:x<Return> quit vi, writing out modified file to file named in original invocation  
:wq<Return> quit vi, writing out modified file to file named in original invocation  
:q<Return> quit (or exit) vi  
:q!<Return> quit vi even though latest changes have not been saved for this vi call
```

Moving the Cursor

Unlike many of the PC and MacIntosh editors, the mouse does not move the cursor within the vi editor screen (or window). You must use the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in vi and should be avoided. If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two. In the table below, the symbol ^ before a letter means that the <Ctrl> key should be held down while the letter key is pressed:

j or <Return> [or down-arrow]	move cursor down one line
k [or up-arrow]	move cursor up one line
h or <Backspace> [or left-arrow]	move cursor left one character
l or <Space> [or right-arrow]	move cursor right one character
0 (zero) ← cursor)	move cursor to start of current line (the one with the ← cursor)
\$	move cursor to end of current line
w	move cursor to beginning of next word
b	move cursor back to beginning of preceding word
:0<Return> or 1G	move cursor to first line in file
:n<Return> or nG	move cursor to line n
:\$<Return> or G	move cursor to last line in file

Screen Manipulation

The following commands allow the vi editor screen (or window) to move up or down several lines and to be refreshed.

- ^f move forward one screen
- ^b move backward one screen
- ^d move down (forward) one half screen
- ^u move up (back) one half screen
- ^l redraws the screen
- ^r redraws the screen, removing deleted lines

Adding, Changing, and Deleting Text

This command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

- u UNDO WHATEVER YOU JUST DID; a simple toggle

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

- i** insert text before cursor, until <Esc> hit
- I** insert text at beginning of current line, until <Esc> hit
- a** append text after cursor, until <Esc> hit
- A** append text to end of current line, until <Esc> hit
- o** open and put text in a new line below current line, until <Esc> hit
- O** open and put text in a new line above current line, until <Esc> hit

Changing Text

The following commands allow you to modify text.

- r** replace single character under cursor (no <Esc> needed)
- R** replace characters, starting with current cursor position, until <Esc> hit
- cw** change the current word with new text, starting with the character under cursor, until <Esc> hit
- cNw** change N words beginning with character under cursor, until <Esc> hit; e.g., c5w changes 5 words
- C** change (replace) the characters in the current line, until <Esc> hit
- cc** change (replace) the entire current line, stopping when <Esc> is hit
- Ncc** or **cNc** change (replace) the next N lines, starting with the current line, stopping when <Esc> is hit

Deleting Text

The following commands allow you to delete text.

- x** delete single character under cursor
- Nx** delete N characters, starting with character under cursor
- dw** delete the single word beginning with character under cursor
- dNw** delete N words beginning with character under cursor; e.g., d5w deletes 5 words
- D** delete the remainder of the line, starting with current cursor position
- dd** delete entire current line
- Ndd** delete N lines, beginning with the current line; e.g., 5dd deletes 5 lines
- dNd** same as Ndd

Cutting and Pasting Text

The following commands allow you to copy and paste text.

yy copy (yank, cut) the current line into the buffer

Nyy copy (yank, cut) the next N lines, including the current line, into the buffer

yNy same as Nyy

p put (paste) the line(s) in the buffer into the text after the current line

Searching Text

A common occurrence in text editing is to replace one word or phrase by another. To locate instances of particular sets of characters (or strings), use the following commands.

/string search forward for occurrence of string in text

?string search backward for occurrence of string in text

n move to next occurrence of search string

N move to next occurrence of search string in opposite direction

Determining Line Numbers

Being able to determine the line number of the current line or the total number of lines in the file being edited is sometimes useful.

:.= returns line number of current line at bottom of screen

:= returns the total number of lines at bottom of screen

^g provides the current line number, along with the total number of lines, in the file at the bottom of the screen

Saving and Reading Files

These commands permit you to input and output files other than the named file with which you are currently working.

:r filename<Return> read file named filename and insert after current line (the line with cursor)

:w<Return> write current contents to file named in original vi call

:w newfile<Return> write current contents to a new file named newfile

:12,35w smallfile<Return> write the contents of the lines numbered 12 through 35 to a new file named smallfile

:w! prevfile<Return> write current contents over a pre-existing file named prevfile

3.2 Data Storage

Scratch is designed to be a high-performance, low-latency, parallel file system and it is uniquely built for HPC systems. That means that data is allocated in parallel on several virtual disks intended to be read and write in parallel from several computers. In order for the scratch system to perform its best the files should be balanced across all disks and frequently used files should also be balanced.

In this page we will offer some tricks that will help you regain control of your scratch space in preparation to transferring your files to other locations.

3.2.1 Knowing my current usage

First we should know what we have in terms of used space and number of files

```
$ /usr/lpp/mmfs/bin/mmlsquota --block-size M gpfs01:scratch
                                Block Limits
  ↵ File Limits
Filesystem Fileset      type          MB      quota      limit   in_doubt   grace |  ↵
  ↵ files   quota    limit in_doubt   grace Remarks
gpfs01     scratch    USR        1943448  62914560  68157440       235    none |  ↵
  ↵ 557657 24000000 25000000      172      none
```

You can also get this information in GigaBytes

```
$ /usr/lpp/mmfs/bin/mmlsquota --block-size G gpfs01:scratch
                                Block Limits
  ↵ File Limits
Filesystem Fileset      type          GB      quota      limit   in_doubt   grace |  ↵
  ↵ files   quota    limit in_doubt   grace Remarks
gpfs01     scratch    USR        1898      61440    66560       1    none |  ↵
  ↵ 557657 24000000 25000000      172      none
```

On this case I am using almost 1900 GB of data in half million files. I would like to reduce as much as possible before trying to move my data out of Scratch. We will focus on command line tricks that will help you search for Big files, search for small and numerous files. The idea is to attack first the biggest contributors to my usage. These examples will be as generic as possible. At the end of the day it's up to you to decide which data is no longer needed and can be safely deleted.

Consider delete a irreversible procedure, there is a way to recover files that have been deleted very recently but do not count on it. Being sure that you delete what you actually want to delete. The commands below are powerful and with that power comes a danger too.

3.2.2 Searching for empty files

An empty file is still a file. Even if apparently not using space its metadata still exists and any transfer of that file is also a file operation. This is something that in most cases is harmless. Be careful, in some cases empty files are used for some codes to indicate run conditions. Their sole existence carries a meaning.

On Spruce every user has an environmental variable pointing to their personal Scratch folder. You can see that variable with:

```
$ echo $SCRATCH
/scratch/<USERNAME>
```

On this examples we will use this variable to allow you to copy and paste this commands in total generality

```
find $SCRATCH -size 0
```

I got thousands of files. Some of them I do not want to remove, but there are quite a number of files like this:

```
./OrbitalGAF/58d36a1374d8b558e47f9623/58d36a1374d8b558e47f9623.e1165669
./OrbitalGAF/58d46c6d74d8b558e47f9cf7/58d46c6d74d8b558e47f9cf7.o1172814
./OrbitalGAF/58d46c6d74d8b558e47f9cf7/58d46c6d74d8b558e47f9cf7.e1172814
./OrbitalGAF/58d36a1374d8b558e47f9629/58d36a1374d8b558e47f9629.o1165671
./OrbitalGAF/58d36a1374d8b558e47f9629/58d36a1374d8b558e47f9629.e1165671
./OrbitalGAF/58d46c6d74d8b558e47f9cc7/58d46c6d74d8b558e47f9cc7.o1172802
./OrbitalGAF/58d46c6d74d8b558e47f9cc7/58d46c6d74d8b558e47f9cc7.e1172802
./OrbitalGAF/58d46c6d74d8b558e47f9cca/58d46c6d74d8b558e47f9cca.o1172803
./OrbitalGAF/58d46c6d74d8b558e47f9cca/58d46c6d74d8b558e47f9cca.e1172803
./OrbitalGAF/58d46c6d74d8b558e47f9cd3/58d46c6d74d8b558e47f9cd3.o1172806
./OrbitalGAF/58d46c6d74d8b558e47f9cd3/58d46c6d74d8b558e47f9cd3.e1172806
./OrbitalGAF/58d46c6d74d8b558e47f9cd6/58d46c6d74d8b558e47f9cd6.o1172807
./OrbitalGAF/58d46c6d74d8b558e47f9cd6/58d46c6d74d8b558e47f9cd6.e1172807
./OrbitalGAF/58d46c6d74d8b558e47f9cd9/58d46c6d74d8b558e47f9cd9.o1172808
./OrbitalGAF/58d46c6d74d8b558e47f9cd9/58d46c6d74d8b558e47f9cd9.e1172808
./OrbitalGAF/58d46c6d74d8b558e47f9cf4/58d46c6d74d8b558e47f9cf4.o1172813
./OrbitalGAF/58d46c6d74d8b558e47f9cf4/58d46c6d74d8b558e47f9cf4.e1172813
./OrbitalGAF/58d36a1374d8b558e47f963e/58d36a1374d8b558e47f963e.o1165678
./OrbitalGAF/58d36a1374d8b558e47f963e/58d36a1374d8b558e47f963e.e1165678
./OrbitalGAF/58d46c6d74d8b558e47f9cb5/58d46c6d74d8b558e47f9cb5.o1172796
./OrbitalGAF/58d46c6d74d8b558e47f9cb5/58d46c6d74d8b558e47f9cb5.e1172796
./OrbitalGAF/58d46c6d74d8b558e47f9cbe/58d46c6d74d8b558e47f9cbe.o1172799
./OrbitalGAF/58d46c6d74d8b558e47f9cbe/58d46c6d74d8b558e47f9cbe.e1172799
./OrbitalGAF/58d46c6d74d8b558e47f9cc1/58d46c6d74d8b558e47f9cc1.o1172800
./OrbitalGAF/58d46c6d74d8b558e47f9cc1/58d46c6d74d8b558e47f9cc1.e1172800
./OrbitalGAF/58d46c6d74d8b558e47f9cc4/58d46c6d74d8b558e47f9cc4.o1172801
./OrbitalGAF/58d46c6d74d8b558e47f9cc4/58d46c6d74d8b558e47f9cc4.e1172801
./OrbitalGAF/58d36a1374d8b558e47f9653/58d36a1374d8b558e47f9653.o1165683
```

All those are empty files created by Torque/Moab from simulations that I have carried out in the past. Those files in the format:

```
<JOB_NAME>.o<JOB_ID_NUMBER>
<JOB_NAME>.e<JOB_ID_NUMBER>
```

Those files contain the standard Output (o) and Standard Error (e) from the jobs that run on Spruce. Many of those files are empty because on my executions I usually send the output of the command to some other file and because the codes that I use do not write anything on the standard error. Deleting those empty files will not help me with my storage usage by it will reduce the number of files that I will have to move later on.

I can target those files with a more specific command:

```
find $SCRATCH -size 0 -regextype posix-egrep -regex ".*\.\[o,e\][0-9]{5,7}"
```

The command above search for all empty files on Scratch that contains. On my account I found thousands of cases that I can safely delete. In most cases those files are not important to you to keep. They are empty and not used at all other than telling you the JOB_ID that you probably do not need for old jobs. If you feel confortable deleting those files you can execute:

```
find $SCRATCH -size 0 -regextype posix-egrep -regex ".*\.[o,e][0-9]{5,7}" -exec rm {} \;
```

On my case it help me delete 16113 files

3.2.3 Searching for Big Files

Some problems and codes produce big files. In some cases those files are not needed to extract the actual answers for which you run those calculations and could be deleted safely. The first step is knowing where they are. This command search for files bigger than 2GB

```
find $SCRATCH -size +2G
```

You can increase or decrease the value if you are getting too many or very few. Once you identify the biggest files you can use the their names to target those files for deletion. Imagine that the files called “WAVECAR” can be deleted. You can use this command”

```
find $SCRATCH -size +2G -name "WAVECAR" -exec rm {} \;
```

3.3 Environment Modules

3.3.1 What are Modulefiles

Modulefiles are files written in the Tool Command Language (Tcl) and are used by the module command to implement different environments for particular applications. Whenever a user on a Unix/Linux based system runs applications, compiles code, etc. what the user can successfully do is determined by the current shell environment. The most common application of this concept is the Unix/Linux PATH shell environment. When a user types a command, that command is either referenced by a full path name: e.g. /usr/bin/some_dir/another_dir/command_utility or the path needs to be specified in the user’s PATH shell environment. Another example is when a particular user compiles application code, often several libraries need to be accessible and their are Unix shell environment variables that direct the compiler where to look (e.g. LD_LIBRARY_PATH). On a moderately sized or large cluster this becomes somewhat complicated to keep track of all the libraries and commands available on the system. An example would be compile-time libraries: often on a HPC system there are several versions of the same library to fulfill all user’s needs. If all libraries where available by default to all users there is a good chance that application code would access the incorrect versions and cause errors. Modulefiles can be loaded and unloaded to modify certain shell environment variables to make applications available to the user and not interfere with other applications.

On RC HPC managed systems we use modulefiles to control shell environment for any libraries, compilers, and software that are installed locally (not globally located by default). In addition, modulefiles contain useful information such as version numbers, URLs for webpages and manuals, and MAN page information. Further, modulefiles are a great way to see what software, libraries and compilers are available on our systems.

3.3.2 Listing available modulefiles

To list the currently available modulefiles on the system use the module command with the available subcommand:

```
$> module available

----- /usr/share/Modules/software -----
 $\hookleftarrow$ 
chemistry/autodock_vina genomics/clumpp      genomics/muscle      genomics/tabix
chemistry/gamess      genomics/cufflinks    genomics/phase       genomics/tablet
chemistry/gaussian/g03 genomics/distruct     genomics/phrap       genomics/tophat2
chemistry/gromacs      genomics/eigensoft   genomics/picard-tools  genomics/trf
data/cdf/3.5.0.2       genomics/emboss      genomics/plink       genomics/trinity
data/hdf5/1.8.12        genomics/faast36     genomics/qiime       genomics/vcftools
genomics/abyss          genomics/fastqc      genomics/repeatexplorer gnu/parallel
genomics/allpaths       genomics/fastx_toolkit  genomics/rsem        mae/elk/2.3.16
genomics/beagle         genomics/gatk        genomics/samtools    statistics/matlab
genomics/bedtools       genomics/hmmer       genomics/shapeit
genomics/blast          genomics/igv         genomics/soapdenovo
genomics/bowtie2        genomics/lastz      genomics/structure
```



```
----- /usr/share/Modules/development -----
 $\hookleftarrow$ 
compilers/gcc/4.8.2     libraries/DFS/myhadoop  libraries/hdf5/1.8.12  mpi/mpich2/1.2.1
compilers/gcc/4.9.0     libraries/bupc/2.2      libraries/mkl/4.1.1.036 mpi/mvapich2/1.9
compilers/intel/14.0    libraries/cdf/3.5.0.2  mpi/intel/4.1.1.036  mpi/openmpi/1.6.5
```

The output of the available subcommand informs the full path where the modulefiles are located (e.g. /usr/share/Modules/software) as well as a list of modulefiles in that particular location (e.g. gnu/parallel). Importantly, all modulefiles are sectioned by categories. For instance, all compiler modulefiles start with ‘compilers’/ (i.e. compilers/gcc/4.8.2). The module available command can be given words for matching, and will return modulefiles that start with the given word. For instance, if you want to list all MPI modulefiles:

```
$>module available mpi

----- /usr/share/Modules/development -----
 $\hookleftarrow$ 
mpi/intel/4.1.1.036  mpi/mpich2/1.2.1  mpi/mvapich2/1.9  mpi/openmpi/1.6.5
```

Further, you can list all gcc compilers:

```
$>module available compilers/gcc

----- /usr/share/Modules/development -----
 $\hookleftarrow$ 
compilers/gcc/4.8.2  compilers/gcc/4.9.0
```

3.3.3 Showing what a modulefile does

Viewing the contents of a modulefile can be done using the module command with the show subcommand:

```
$> module show compilers/intel/14.0

-----
/usr/share/Modules/development/compiler/intel/14.0:

module-whatis      Name: Intel Compiler
module-whatis      Version: 14.0
module-whatis      Category: compiler, runtime support
module-whatis      Description: Intel Compiler Family (C/C++/Fortran for x86_64)
module-whatis      URL: http://software.intel.com/en-us/articles/intel-compilers/
setenv      INTEL_LICENSE_FILE 28519@srih0001.hpc.wvu.edu
prepend-path    PATH /shared/software/intel/bin
prepend-path    MANPATH /shared/software/intel/man
prepend-path    LD_LIBRARY_PATH /shared/software/intel/lib/intel64
-----
```

the modulefile starts with the location of the modulefile (/usr/share/Modules/development/compiler/intel/14.0). The next few lines module-whatis gives information about version numbers, categories for tools, URLs for more information. If you only want to see these first few lines you can use the whatis subcommand:

```
$> module whatis compilers/intel/14.0

compilers/intel/14.0 : Name: Intel Compiler
compilers/intel/14.0 : Version: 14.0
compilers/intel/14.0 : Category: compiler, runtime support
compilers/intel/14.0 : Description: Intel Compiler Family (C/C++/Fortran for x86_64)
compilers/intel/14.0 : URL: http://software.intel.com/en-us/articles/intel-compilers/
```

Returning back to the actual modulefile, after the whatis lines you have specific Tcl commands that modify the environment. In this example we have two types of commands: 1) setenv which creates a shell environment variable (INTEL_LICENSE_FILE in this case). 2) prepend-path statement which modify a current shell environment variable. In this case the first prepend-path modifies the PATH environment variable by adding /shared/software/intel/bin to make that directory available to the user. More information about Tcl commands found in modulefiles can be found in the modulefile manpage (man modulefile).

3.3.4 Loading and Unloading modulefiles

Loading and unloading modulefiles can be done using the module command the subcommands load and unload. In this example, the Intel C compiler is named `icc`:

```
$> icc
-bash: icc: command not found

$> module load compilers/intel
$> icc
icc: command line error: no files specified; for help type "icc -help"

$> module unload compilers/intel
```

(continues on next page)

(continued from previous page)

```
$>icc
-bash: icc: command not found
```

As shown in the above command, initially the icc commands could not be found. After loading the module, icc could be found (although I gave it no input so it gave me an error), and as expected after unloading the module icc could not be found anymore. For more information about module subcommands see the module manpage (man module)

3.3.5 Using modulefiles through qsub

Modulefiles can be used through the scheduler just as on the command line. In your batch shell script (see [Running Jobs](#) for details) you can place the module load/unload or any other module subcommand directly before you executed commands.

3.4 Workload Manager (SLURM)

The login node (the shell you get upon logging into a cluster) should be used for text editing, file transfer, and job submissions. No Jobs should be run directly on the login node. Jobs that take up too much CPU time or RAM will be cancelled by the WVU RC staff to ensure proper functioning of the cluster. Jobs should be run using the cluster's queuing system, which uses the Moab Cluster Manager. Both batch jobs and interactive jobs should be submitted using the queuing system.

3.4.1 Batch queues

Both batch jobs and interactive jobs will be submitted to a batch queue, which specifies the length and resources available to your job. Current status of a cluster's queues can be found using the `qstat -q` command. There are several options that you can use to get an idea about the queues available on the system, the output here is for Spruce cluster

```
$ qstat -q

server: srih0001.hpc.wvu.edu

Queue      Memory  CPU Time Walltime Node  Run Que Lm  State
-----  -----  -----  -----  -----  ---  ---  --  -----
comm_mmem_week    --    -- 168:00:0  --   26   0  --  E R
standby          --    -- 04:00:00  --  182 122  --  E R
comm_256g_mem    --    -- 168:00:0  --    0   0  --  E R
comm_mmem_day    --    -- 24:00:00  --  566 592  --  E R
debug            --    -- 00:15:00  --    0   0  --  E R
comm_gpu          --    -- 168:00:0  --    0   0  --  E R
batch             --    -- 04:00:00  --    0   0  --  D S
comm_smp          --    -- 168:00:0  --    0   20  --  E R
comm_large_mem   --    -- 168:00:0  --    0   0  --  E R
...
-----  -----  -----  -----  -----  ---  ---  --  -----
                                         877    914
```

This command will give you an idea about the Walltime of the queues and the number of jobs and queue and running.

Each queue has a designated Walltime, which specifies how long a particular job can run. We recommend that users submit job scripts first to the debug queue which has the highest priority (meaning your job will be placed running

before all other jobs submitted to other queues). The debug queue only has a 5 minute Walltime, but is perfect for making sure your job script is valid and no errors will occur. Without first running a job in the debug queue, you run the risk of waiting a couple of hours before your job starts and then finding errors. Other queues including week, hour and long are more appropriate for longer running jobs and your job's priority runs using your user's priority which is explained in the [Policies](#) page. If you cannot run your job within the specified runtime limits, please let us know.

3.4.2 Useful Commands for Job Submission/Status

qsub	Used to submit both batch and interactive jobs (explained below)
canceljob	Terminates a job in the queue or a running job
checkjob	Displays detailed job state information

3.4.3 Submitting a Batch job to the Queue

To submit a batch job to the queue use the qsub command to submit a job submission shell script. Useable job submission shell scripts are available for easy modification at [Sample Job Scripts](#).

```
$> qsub <environment_options> <shellscriptname>
```

** are options qsub takes to specify job submission parameters including the specified queue to submit the job, how many CPUs/much RAM is needed, working directories etc (more on this below under Serial Jobs). ** is the name of a shell script (bash, tcsh, etc.) that contains the commands that will be executed at run time. Default standard output and standard error of the job will be placed in files named *jobname.ojobid* and *jobname.ejobid*, respectively. These files will be written to the directory in which the qsub command was executed in. Where jobname is specified using the -N environment option and jobid is given at run time by the system. A trivial example is

```
$> qsub -N example -q debug ./batch_job.sh  
42129.mountaineer
```

In this example output files are named example.o42129 and example.e42129 for standard output and standard error, respectively.

Batch Shell Scripts

Serial batch jobs (jobs not requiring any MPI libraries) are submitted to the system using shell scripts. An example (very simple) shell script could be

```
#!/bin/bash  
  
time command_to_be_executed
```

The above bash script, if submitted to the system using qsub would execute a single command. The unix time command before the command_to_be_executed is a very useful tool when using HPC scheduler systems. This command will report the real time it takes for a command to begin and complete execution. Times will be reported in the standard error file, and are useful for knowledge about how much walltime a command actually uses. The bash script shown above, when executed using the qsub command, the user would have to specify environment options at the command line. An easier way is to place environment options directly into you shell script using #PBS comments.

```
#!/bin/bash
```

(continues on next page)

(continued from previous page)

```
#PBS -N example
#PBS -q debug

time command_to_be_executed
```

By adding the -N and -q options directly into your shell script you no longer have to specify these options on the command line when issuing the qsub command.

Note: Do not place a job in the background using the ‘&’ symbol, you will confuse the scheduler and potentially loose your command output.

Qsub Environment Variables

Environment variables can be specified on the command line using the qsub command.

```
$> qsub -N batch_job -q week -l procs=3 ./batch_job.sh
```

The above example would submit a job named batch_job to the week queue with 3 CPUs used and would execute batch_job.sh located in the current directory. The output files would be batch_job.ojobid and batch_job.ejobid. Another way is to specify these options directly in the shell script using #PBS commands.

```
#!/bin/bash

#PBS -N batch_job
#PBS -q week
#PBS -l nodes=1:ppn=3

time command_to_be_executed
```

With the environment options contained in the shell script, you no longer have to specify them on the command line.

```
$> qsub ./batch_job.sh
```

The qsub command without options is identical to the previous command with options. Below is a list of commonly used qsub environment options, and these options are further explained in below sections.

-N	Job Name
-q	Queue specification
-l	Used to specify job resources (number of CPUs, nodes, length of Walltime)
-m	E-mail options
-M	E-mail address(es) for e-mail options
-e	Path for error stream
-o	Path for output stream
-t	request for array jobs

Note: More information about the PBS system can be found using the man pbs command at the terminal. Further an extensive list of qsub options including environment variables can be found using the man qsub command.

Resource Specification

The #PBS -l option is used to specify resources such as number of CPUs, nodes, and length of Walltime for the job specified. The three most common resources specified for the Mountaineer cluster are

nodes	Number of nodes needed
walltime	Maximum limit for walltime given in the format hh:mm:ss
ppn	Processors per node
procs	Number of processors requested
pvmem	Maximum amount of memory used by any single process in the job
vmem	Maximum amount of memory used by all concurrent processes in the job

Note: procs is used when you do not require each CPU to be on the same node.

For example, the PBS directive

```
#PBS -l nodes=1:ppn=6,walltime=06:00:00
```

Specifies that the job will need 6 processors located on a single node with a maximum run time of 6 hours. Notice there is no space between commas or equal signs. Alternatively, if nodes=1 (procs=6 instead) had not been specified then the scheduler would just grab the first 6 processors available regardless of what nodes they reside on (which will only work if your program supports distributed computing). In general, unless you are running jobs using MPI libraries (mpirun) or posix threads, you will most likely only specify a single processor for your job (procs=1). **Note:**Resources specifying per node request are given with the nodes directive and separated with a :, on the same line in your script.

Requesting Memory Specifications

Requesting memory specifications for jobs is done with the attributes vmem or pvmem through the PBS -l directive (resource specification). The man pages of pbs will specify two other memory related attributes: mem and pmem. However, these two attributes measure different job resources than virtual memory and therefore are not stable for use the way we commonly think of memory (use of RAM). In other words, do not use the attributes mem and pmem - they most likely do not do what you think they do. vmem and pvmem will put resource limits for the amount of RAM a job can access. This is important to ensure two large memory jobs do not end up on the same node; exceeding the node's memory limits and causing a node crash (which will kill all jobs on the node). If you do not specify memory limits - moab will assume a uniform distribution of memory across all jobs on the node. For example, a 16 processor/64Gb of RAM compute node will assume roughly 4Gb of RAM per processor. However, if a job using 62 Gb of RAM and only 8 cores is running on a compute node - without memory limits Moab will place 8 more processor jobs on that node when clearly there is not enough memory for any remaining jobs. This will crash the node. Therefore, we recommend that if you anticipate your jobs are going to use more than an average of 3Gb per processor that you specify memory limits for your job using pvmem or vmem. On Spruce community nodes and Mountaineer we enforce this by making the system default of pvmem=3gb. On these systems without specifying memory above 3Gb will cause your job to fail. This is important - because on community nodes if you specify a job with 5 cores and vmem=25Gb; the job still will fail if it exceeds 15Gb because pvmem=3gb is assigned to each job by default (i.e. vmem does not override pvmem settings). To make your PBS scripts portable across community nodes and private nodes, we recommend that you only use pvmem to specify memory limits of jobs. pvmem attribute specifies the maximum amount of virtual memory used by any single processes in the job. Therefore, if you want a job that uses 6 processors and needs 35 Gb of RAM you would specify the following resource directive line:

```
#PBS -l nodes=1:ppn=6,pvmem=6gb
```

pvmem=6gb with 6 processors specifies $6 \times 6 = 36\text{Gb}$ of total memory for the job.

Requesting Certain Node Types

There might be times where you want to be able to request a node with a particular feature or processor. The following will allow you to accomplish this task. Replace ‘feature_name’ with one of the features in the below table.

```
#PBS -l feature=feature_name
```

Note, you can also request a particular feature not by doing the following:

```
#PBS -l feature='!feature_name'
```

Available Features

Feature	Description
smb	Sandy Bridge Based Processor Nodes
ivy	Ivy Bridge Based Processor Nodes
haswell	Haswell Based Processor Nodes
broadwell	Broadwell Based Processor Nodes
avx	Processors with AVX Extension
avx2	Processors with AVX2 Extension
f16c	Processors with f16c Extension
adx	Processors with adx Extension
large	Nodes with 512 GB of memory

E-mail options

The #PBS -m and #PBS -M options are used to specify when and to whom the scheduler will send e-mails. The -m option consists of either the single character “n”, or one or more of the characters “a”, “b”, and “e”.

n	No mail will be sent
a	Mail is sent when the job is aborted by the batch system
b	Mail is sent when the job begins execution
e	Mail is sent when the job ends

Note: If the -m option is not specified, mail will be sent if the job is aborted.

The shellscript option #PBS -M specifies the e-mail addresses to send mail to. For example, the PBS directive

```
#PBS -m ae
#PBS -M user@mailserver.com
```

The scheduler will send an e-mail to user@mailserver.com if the job is aborted, or when the job is completed. To specify more than one e-mail address with the -M option, each address should be separated with a comma without any spaces.

To Receive no e-mails even on aborts

Even with the ‘n’ option of ‘-m’ directive, the system will still send an e-mail if the job is cancelled or aborts. To provide the ability for our users to circumvent this response, we have set-up an alias e-mail address that can be used to bounce these e-mails. To receive absolutely no e-mails from the system, no matter what happens before, during and after execution of your job, use the `noemail@hpc.wvu.edu` address with the ‘n’ option:

```
#PBS -m n  
#PBS -m noemail@hpc.wvu.edu
```

Output file specification

Default standard output and standard error of the job will be placed in files named `jobname.ojobid` and `jobname.ejobid`, respectfully. These files will be written to the directory in which the qsub command was executed in. Where jobname is specified using the -N environment option and jobid is given at run time by the system. The #PBS -e and #PBS -o options are used to specify what files should be written for the standard error and standard output stream, respectively.

-e	pathname for standard error stream output
-o	pathname for standard output stream output

An example, the PBS directive

```
#PBS -e /scratch/username/examplejob.error  
#PBS -o /scratch/username/examplejob.output
```

The scheduler will write the files /scratch/username/examplejob.error and /scratch/username/examplejob.output for the standard error and standard output streams, respectively.

Note: Use full pathnames for your home directory and scratch directory

Requesting Array jobs

By using the directive #PBS -t , you can request a job to be repeated by a single script a number of times. This is useful if you have data where you want a single parameter to range over a section of numbers. For instance, if I wanted a series of commands to be run, with a single variable in the command to be executed over a range of 10-20 I could use the following command directives in my shell script

```
#PBS -N demographic_${PBS_ARRAYID}  
#PBS -l nodes=1:ppn=2  
#PBS -t 10-20  
  
mkdir output_${PBS_ARRAYID}/  
cd output_${PBS_ARRAYID}/  
$SCRATCH/demographic_model.py -input_parameter ${PBS_ARRAYID} -procs 2 -output_file_<br/>demographic_output.txt
```

The above script would launch ten jobs. Each job would have the name demographic_; so the first job would be named demographic_10, the second job would be named demographic_11, and so fourth. Each job would be run a single node with 2 processors (specified as #PBS -l nodes=1:ppn=2). Further, each job would make a directory named ouput_(first job output_10, second job output_11, and so forth). Would cd into that directory and execute the python script demographic_model.py from my scratch directory. Notice that one of the input parameters would change each single job using the PBS set environment variable PBS_ARRAYID. Array request are very useful in scientific environments

when you need to modify a parameter and see the output for a range of values. Note: this a theoretical example since I never specified walltime or a queue to execute this job from.

The number range for array request does not have to be sequential. You can also list a comma separated list of numbers as

```
#PBS -t 10,15,20,25
```

Further, you can also specify that only a certain number of jobs are queued at one time in cases where you have a large number of jobs and need to share a queue with another user

```
#PBS -t 1-200%10
```

The above directive will only launch ten jobs to the queue at a time until all 200 job requests have been executed.

3.4.4 Interactive Jobs

Interactive jobs allow a user to be given an interactive terminal on a compute node. This allows a user to “interact” directly with a compute node instead running in a batch or scripted mode. Interactive jobs are very useful when debugging jobs as it allows a user to walk step-by-step through your submit script to find errors or problems. Interactive jobs are also useful when needing to use a graphical program on the cluster.

To run an interactive job use the following command followed by any necessary PBS variables/flags. If you don’t specify any flags, you will be given an interactive job in the default queue for the cluster.

```
qsub -I
```

Do note, interactive jobs are only allowed on certain queues. All condo owner queues are allowed to have interactive jobs as well as queues such as ‘standby’ and ‘debug’. If you find you need an interactive queue on a community resource for a particular task or project, please contact [Research Computing Help Desk](#) for assistance.

3.4.5 Graphical Interface Jobs

Sometimes it might be useful or required to run a graphical program on the cluster. Non-compute intensive processes for visualization purposes can be run on the login node. These processes include “could” gnuplot, R and Matlab assuming they have low overhead. However, if you know your program is consume a lot of resources, it is best to run an interactive job.

To execute a graphical application on a compute node, you need to first review [Using X Windows applications](#) to properly setup your X (i.e. display) environment. To launch a graphical job on a compute node, you will need to execute the following along with any necessary flags/pbs environment variables.

```
$> qsub -I -X
```

Once you are given an access to a interactive terminal you can run your the proper executable to launch your graphical (i.e. X Window) program. For example:

```
$> module load statistics/matlab
$> matlab &
```

3.4.6 Checking the Status of Jobs

The status of a job currently submitted to the queue can be checked using the checkjob command. checkjob displays detailed job state information and diagnostic output for a specified job. Detailed information is available for queued, blocked, active, and recently completed jobs. Users can use checkjob to view the status of their own jobs.

Examples:

```
$> checkjob -v <jobid>
```

where is the jobid given at submission time.

The output of checkjob looks like this

```
job 1653450 (RM job '1653450.srih0001.hpc.wvu.edu')

AName: IVY
State: Completed
Completion Code: 0 Time: Fri May 19 15:30:21
Creds: user:username group:groupname class:debug qos:member
WallTime: 00:00:16 of 00:01:00
SubmitTime: Fri May 19 15:29:58
(Time Queued Total: 00:00:07 Eligible: 00:00:07)

Deadline: 3:59:49 (Fri May 19 19:30:58)
TemplateSets: DEFAULT
Total Requested Tasks: 1

Req[0] TaskCount: 1 Partition: torque
Opsys: --- Arch: --- Features: ivy
GMetric[energy_used] Current: 0.00 Min: 0.00 Max: 0.00 Avg: 0.00 Total: 0.00
NodeAccess: SINGLEJOB
TasksPerNode: 1
Allocated Nodes:
[sgpc0001.hpc.wvu.edu:1]

SystemID: Moab
SystemJID: 1653450
Notification Events: JobEnd,JobFail
Task Distribution: sgpc0001.hpc.wvu.edu
UMask: 0000
OutputFile: srih0001.hpc.wvu.edu:/gpfs/home/username/IVY.o1653450
ErrorFile: srih0001.hpc.wvu.edu:/gpfs/home/username/IVY.e1653450
StartCount: 1
Execution Partition: torque
SrcRM: torque DstRM: torque DstRMJID: 1653450.srih0001.hpc.wvu.edu
Submit Args: runjob_ivy.pbs
Flags: RESTARTABLE
Attr: checkpoint
StartPriority: 1000
PE: 1.00
```

Sometimes your job is rejected and you still get a jobid in that case you can check the reasons with checkjob. For example, consider this submission script where we ask for too much memory for a serial job.

The submission script looks like

```
#!/bin/sh

#PBS -N TEST
#PBS -l nodes=1:ppn=1,vmem=200g
#PBS -l walltime=00:01:00
#PBS -m ae
#PBS -q groupname
#PBS -n

cd $PBS_O_WORKDIR

date
```

The job is accepted by torque but will see the job in queue for a long time. Now we execute checkjob to know the reasons for not being running

```
$> checkjob -v 1653589

job 1653589 (RM job '1653589.srih0001.hpc.wvu.edu')

AName: TEST
State: Idle
Creds: user:username group:groupname class:groupname qos:member
WallTime: 00:00:00 of 00:01:00
BecameEligible: Fri May 19 15:52:14
SubmitTime: Fri May 19 15:51:52
    (Time Queued Total: 00:01:06 Eligible: 00:00:53)

Deadline: 3:59:54 (Fri May 19 19:52:52)
TemplateSets: DEFAULT
Total Requested Tasks: 1

Req[0] TaskCount: 1 Partition: ALL
Memory >= 0 Disk >= 0 Swap >= 3072M
Dedicated Resources Per Task: PROCS: 1 SWAP: 200G
NodeAccess: SINGLEJOB
TasksPerNode: 1
Reserved Nodes: (3:09:16:24 -> 3:09:17:24 Duration: 00:01:00)
[sarc3001.hpc.wvu.edu:1]

SystemID: Moab
SystemJID: 1653589
Notification Events: JobEnd,JobFail

UMask: 0000
OutputFile: srih0001.hpc.wvu.edu:/gpfs/home/username/TEST.o1653589
ErrorFile: srih0001.hpc.wvu.edu:/gpfs/home/username/TEST.e1653589
Partition List: torque
SrcRM: torque DstRM: torque DstRMJID: 1653589.srih0001.hpc.wvu.edu
Submit Args: runjob_badmem.pbs
Flags: RESTARTABLE
```

(continues on next page)

(continued from previous page)

Attr:	checkpoint
StartPriority:	2000
PE:	37.34
Reservation	'1653589' (3:09:16:24 -> 3:09:17:24 Duration: 00:01:00)
Node Availability for Partition torque	-----
srig0001.hpc.wvu.edu	rejected: Swap
szec2001.hpc.wvu.edu	rejected: State (Busy)
szec2002.hpc.wvu.edu	rejected: State (Busy)
szec2003.hpc.wvu.edu	rejected: State (Busy)
...	
sbmc0017.hpc.wvu.edu	rejected: State (Busy)
sbmc0018.hpc.wvu.edu	rejected: State (Busy)
sbmg0001.hpc.wvu.edu	rejected: Swap
sric0001.hpc.wvu.edu	rejected: Swap
sric0002.hpc.wvu.edu	rejected: Swap
ssmc0006.hpc.wvu.edu	rejected: Swap
sgsc2001.hpc.wvu.edu	rejected: Class
sgsg2001.hpc.wvu.edu	rejected: Swap
sric0022.hpc.wvu.edu	rejected: Class
sric0025.hpc.wvu.edu	rejected: State (Busy)
sbmc0019.hpc.wvu.edu	rejected: State (Busy)
sbmc0020.hpc.wvu.edu	rejected: Swap
sbmc0021.hpc.wvu.edu	rejected: State (Busy)
sbmc0022.hpc.wvu.edu	rejected: State (Busy)
sric0024.hpc.wvu.edu	rejected: Swap
sllc0001.hpc.wvu.edu	rejected: Swap
...	
sspc3006.hpc.wvu.edu	rejected: Swap
sspc3007.hpc.wvu.edu	rejected: Swap
sspc3008.hpc.wvu.edu	rejected: Swap
sspc3009.hpc.wvu.edu	rejected: State (Running)
sspc3010.hpc.wvu.edu	rejected: Swap
NOTE: job req cannot run in partition torque (available procs do not meet requirements)	↳: 0 of 1 procs found
idle procs: 623 feasible procs: 0	
Node Rejection Summary:	[Class: 2][State: 110][Swap: 53]

The “Swap” reason is “memory” related. The “State” reason is CPU related. The Queue system search for 623 cores available and could not find a single machine with 200GB available to launch the job.

Another important tool to monitor jobs and its state is showq

You can get the eligible jobs and their priorities with

```
showq -i -u <username>
```

For example

```
$ showq -i -u username
eligible jobs-----
```

(continues on next page)

(continued from previous page)

JOBID ↳ CLASS	PRIORITY SYSTEMQUEUETIME	XFACTOR	Q	USERNAME	GROUP	PROCS	WCLIMIT	↳
1579829*	14108	1.7	me	username	groupname	16	14:00:00:00	↳
↳ groupname	Tue May 9 12:09:46	10599	1.6	me	username	groupname	4	14:00:00:00
1595467*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:11	10599	1.6	me	username	groupname	4	14:00:00:00
1595464*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:11	10599	1.6	me	username	groupname	4	14:00:00:00
1595468*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:11	10599	1.6	me	username	groupname	4	14:00:00:00
1595466*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:11	10599	1.6	me	username	groupname	4	14:00:00:00
1595463*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:10	10599	1.6	me	username	groupname	4	14:00:00:00
1595465*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:11	10599	1.6	me	username	groupname	4	14:00:00:00
1595462*	10599	1.6	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Thu May 11 22:39:10	6423	1.3	me	username	groupname	2	14:00:00:00
1618053*	6423	1.3	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Sun May 14 20:15:33	6363	1.3	me	username	groupname	4	14:00:00:00
1618385*	6363	1.3	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Sun May 14 21:14:58	6363	1.3	me	username	groupname	4	14:00:00:00
1618386*	6363	1.3	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Sun May 14 21:14:58	6363	1.3	me	username	groupname	4	14:00:00:00
1618387*	6363	1.3	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Sun May 14 21:14:59	6363	1.3	me	username	groupname	4	14:00:00:00
1618388*	6363	1.3	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Sun May 14 21:14:59	3967	1.2	me	username	groupname	4	14:00:00:00
1630355*	3967	1.2	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Tue May 16 13:11:17	3903	1.2	me	username	groupname	4	14:00:00:00
1630507*	3903	1.2	me	username	groupname	4	14:00:00:00	↳
↳ groupname	Tue May 16 14:15:09	3884	1.2	me	username	groupname	16	14:00:00:00
1630546*	3884	1.2	me	username	groupname	16	14:00:00:00	↳
↳ groupname	Tue May 16 14:34:33	1	1.4	co	username	groupname	16	7:00:00:00 comm_
1630494*	1	1.4	co	username	groupname	16	7:00:00:00 comm_	↳
↳ larg	Tue May 16 14:08:50	1	1.4	co	username	groupname	16	7:00:00:00 comm_
1630349*	1	1.4	co	username	groupname	16	7:00:00:00 comm_	↳
↳ larg	Tue May 16 13:10:08							
18 eligible jobs								
Total jobs: 18								

Those are jobs that accrue priority as time passes for them on queue. Some jobs could become blocked, meaning that they are not gaining priority but will eventually become eligible later in time.

\$ showq -b -u username						
blocked jobs-----						
JOBID	USERNAME	GROUP	STATE	PROCS	WCLIMIT	QUEUETIME

(continues on next page)

(continued from previous page)

1623738	username groupname	Idle	16	7:00:00:00	Mon May 15	13:49:50
1623747	username groupname	Idle	16	7:00:00:00	Mon May 15	13:51:21
1623757	username groupname	Idle	16	7:00:00:00	Mon May 15	13:52:57
1652487	username groupname	Idle	16	4:00:00	Fri May 19	12:24:44
1646112	username groupname	Idle	4	4:00:00	Thu May 18	15:20:54
1646096	username groupname	Idle	4	4:00:00	Thu May 18	15:17:55
1630495	username groupname	Idle	4	7:00:00:00	Tue May 16	14:10:13
1630501	username groupname	Idle	16	7:00:00:00	Tue May 16	14:11:17
1623766	username groupname	Idle	16	7:00:00:00	Mon May 15	13:55:24
1623746	username groupname	Idle	16	7:00:00:00	Mon May 15	13:50:50
1623749	username groupname	Idle	16	7:00:00:00	Mon May 15	13:51:48
1623751	username groupname	Idle	16	7:00:00:00	Mon May 15	13:52:25
1646143	username groupname	Idle	16	7:00:00:00	Thu May 18	15:26:36
1623759	username groupname	Idle	16	7:00:00:00	Mon May 15	13:53:51
1623758	username groupname	Idle	16	7:00:00:00	Mon May 15	13:53:29
1623760	username groupname	Idle	16	7:00:00:00	Mon May 15	13:54:53
1623740	username groupname	Idle	16	7:00:00:00	Mon May 15	13:50:23
1623731	username groupname	Idle	16	7:00:00:00	Mon May 15	13:49:08
1630569	username groupname	Idle	16	7:00:00:00	Tue May 16	14:48:03
1623739	username groupname	Idle	16	7:00:00:00	Mon May 15	13:49:53
1623732	username groupname	Idle	16	7:00:00:00	Mon May 15	13:49:10

21 blocked jobs

Total jobs: 21

Finally, you can see the jobs that are currently running with their remaining time until hit their wall time

\$ showq -r -u username									
active jobs-----									
JOBID	S	PAR	EFFIC	XFACTOR	Q	USERNAME	GROUP	MHOST	PROCS
↪ REMAINING					STARTTIME				
1599005	R	tor	24.99	1.0	co	username groupname	sric0011.hpc.wvu		
↪ 16 00:24:38					Fri May 12 17:01:10				
1599006	R	tor	24.99	1.0	co	username groupname	sric0020.hpc.wvu		
↪ 16 00:51:08					Fri May 12 17:27:40				
1599007	R	tor	24.98	1.0	co	username groupname	sric0021.hpc.wvu		
↪ 16 1:03:41					Fri May 12 17:40:13				
1599008	R	tor	24.99	1.0	co	username groupname	sric0023.hpc.wvu		
↪ 16 1:04:45					Fri May 12 17:41:17				
1599009	R	tor	24.99	1.1	co	username groupname	sric0032.hpc.wvu		
↪ 16 4:42:25					Fri May 12 21:18:57				
1599010	R	tor	24.99	1.1	co	username groupname	sric0026.hpc.wvu		
↪ 16 4:42:25					Fri May 12 21:18:57				
1599011	R	tor	24.99	1.1	co	username groupname	sric0017.hpc.wvu		
↪ 16 10:10:42					Sat May 13 02:47:14				
1546851	R	tor	99.73	2.6	co	username groupname	sric0025.hpc.wvu		
↪ 16 2:13:45:30					Mon May 15 06:22:02				
1570354	R	tor	87.78	1.0	me	username groupname	sarc3001.hpc.wvu		
↪ 16 3:08:32:50					Tue May 9 01:09:22				

(continues on next page)

(continued from previous page)

1595446	R tor	98.27	1.0 me	username groupname sarc2001.hpc.wvu	[]
↳ 4 6:06:02:54	Thu May 11	22:39:26			
1595448	R tor	99.98	1.0 me	username groupname sarc2001.hpc.wvu	[]
↳ 4 6:07:29:35	Fri May 12	00:06:07			
1595449	R tor	99.99	1.0 me	username groupname sarc0001.hpc.wvu	[]
↳ 4 6:08:21:37	Fri May 12	00:58:09			
1595453	R tor	99.99	1.0 me	username groupname sarc0002.hpc.wvu	[]
↳ 4 6:08:49:41	Fri May 12	01:26:13			
1618813	R tor	24.77	1.7 co	username groupname srjc0037.hpc.wvu	[]
↳ 16 6:20:47:59	Fri May 19	13:24:31			
1618812	R tor	24.77	1.7 co	username groupname srjc0051.hpc.wvu	[]
↳ 16 6:20:47:59	Fri May 19	13:24:31			
1618814	R tor	24.78	1.7 co	username groupname srjc0036.hpc.wvu	[]
↳ 16 6:20:47:59	Fri May 19	13:24:31			
1618815	R tor	24.84	1.7 co	username groupname srjc0030.hpc.wvu	[]
↳ 16 6:20:54:14	Fri May 19	13:30:46			
1595460	R tor	99.97	1.1 me	username groupname sarc0006.hpc.wvu	[]
↳ 4 8:06:16:50	Sat May 13	22:53:22			
1595461	R tor	99.97	1.2 me	username groupname sarc0009.hpc.wvu	[]
↳ 4 8:13:36:38	Sun May 14	06:13:10			
19 active jobs	232 of 3112 processors in use by local jobs (7.46%)				
	155 of 165 nodes active (93.94%)				
Total jobs:	19				

3.4.7 Canceling/Removing a Job

Jobs can be cancelled or removed using the canceljob command. Users can only remove jobs they submitted to the scheduler.

```
$> canceljob <jobid>
```

is the jobid given at submission time.

Now canceljob is deprecated and Moab offers an alternative to cancel jobs. For example, if you want to cancel jobs that start with 1693 you can use this command to cancel those jobs. As a user you can only cancel jobs that you own so do not worry about canceling jobs from other users by doing this.

```
$> mjobctl -c "x:1693.*"
```

3.4.8 Adding Prologue and Epilogue scripts to a Job

It is possible to declare scripts that run before and after the execution of the main submission script. The main advantage of those is to keep a record of the conditions under which a given job is running. Here we present a simple example of how to declare an prologue and epilogue.

Add these lines to your submission script:

```
#PBS -l prologue=/absolute/path/to/prologue.sh  
#PBS -l epilogue=/absolute/path/to/epilogue.sh
```

The best way of working with those scripts is adding them to your home folder and use them on all your submission scripts. They should collect information that you can use later for debugging or profiling purposes.

Example of Prologue

prologue.sh

```
#!/bin/sh  
  
echo ""  
echo "Prologue Args:"  
echo "Job ID: $1"  
echo "User ID: $2"  
echo "Group ID: $3"  
echo ""  
  
env | sort  
hostname  
date  
  
exit 0
```

Example of Epilogue

epilogue.sh

```
#!/bin/sh  
  
echo ""  
echo "Epilogue Args:"  
echo "Job ID: $1"  
echo "User ID: $2"  
echo "Group ID: $3"  
echo "Job Name: $4"  
echo "Session ID: $5"  
echo "Resource List: $6"  
echo "Resources Used: $7"  
echo "Queue Name: $8"  
echo "Account String: $9"  
echo ""
```

(continues on next page)

(continued from previous page)

```
env | sort
hostname
date

exit 0
```

Both prologue and epilogue must be made executable, use “

```
chmod +x prologue.sh epilogue.sh
```

to change their permissions.

3.4.9 Samples of Job Submission scripts

Below are bash scripts that can be modified and submitted to the qsub command for job submission. For details about the different parts of the scripts please visit the [Running Jobs](#) page. These scripts can be copied and pasted in the terminal using any number of text editors (i.e. vi, emacs, etc...)

3.4.10 Script for running a non-array batch queue

The below script has PBS directives to set-up commonly used variables such as job name, resources needed, e-mail address upon job completion and abnormal termination and specify a queue to run on

```
#!/bin/sh

#This is an example script for executing generic jobs with
# the use of the command 'qsub <name of this script>'

#These commands set up the Grid Environment for your job. Words surrounding by a bracket
#(<,>) should be changed
#Any of the PBS directives can be commented out by placing another pound sign in front
#example
##PBS -N name
#The above line will be skipped by qsub because of the two consecutive # signs

# Specify job name
#PBS -N <name>

# Specify the resources need for the job
# Walltime is specified as hh:mm:ss (hours:minutes:seconds)
#PBS -l nodes=<number_of_nodes>:ppn=<number_of_processors_per_node>,walltime=<time_needed_
#by_job>

# Specify when Moab should send e-mails 'ae' below user will
# receive e-mail for any errors with the job and/or upon completion
# If you don't want e-mails just comment out these next two PBS lines
#PBS -m ae

# Specify the e-mail address to receive above mentioned e-mails
```

(continues on next page)

(continued from previous page)

```
#PBS -M <email_address>

# Specify the queue to execute task in. Current options can be found by executing the
# command qstat -q at the terminal
#PBS -q <queue_name>

# Enter your command below with arguments just as if you were going to execute on the
# command line
# It is generally good practice to issue a 'cd' command into the directory that contains
# the files
# you want to use or use full path names
```

3.4.11 Script for running an array batch queue

Script is the same as above, but adds PBS -t to execute array request job submissions.

```
#!/bin/sh

#This is an example script for executing genetic jobs with
# the use of the command 'qsub <name of this script>'

#These commands set up the Grid Environment for your job. Words surrounding by a bracket
#(<,>) should be changed
#Any of the PBS directives can be commented out by placing another pound sign in front
#example
##PBS -N name
#The above line will be skipped by qsub because of the two consecutive # signs

# Specify job name, use ${PBS_ARRAYID} to ensure names and output/error files have
# different names
#PBS -N <name_${PBS_ARRAYID}>

# Specify the range for the PBS_ARRAYID environment variable
# <num_range> can be a continuous range like 1-200 or 5-20
# or <num_range> can be a comma separated list of numbers like 5,15,20,55
# You can also specify the maximum number of jobs queued at one time with the percent
# sign
# so a <num_range> specified as 5-45% would launch forty jobs with a range from 5-45,
# but only queue 8 at a time until
# all jobs are completed.
# Further, you can mix and match continuous range and list like 1-10,15,25-40%10
#PBS -t <num_range>

# Specify the resources need for the job
# Walltime is specified as hh:mm:ss (hours:minutes:seconds)
#PBS -l nodes=<number_of_nodes>:ppn=<number_of_processors_per_node>,walltime=<time_needed_
#by_job>

# Specify when Moab should send e-mails 'ae' below user will
```

(continues on next page)

(continued from previous page)

```
# receive e-mail for any errors with the job and/or upon completion
# If you don't want e-mails just comment out these next two PBS lines
#PBS -m ae

# Specify the e-mail address to receive above mentioned e-mails
#PBS -M <email_address>

# Specify the queue to execute task in. Current options can be found by executing the
# command qstat -q at the terminal
#PBS -q <queue_name>

# Enter your command below with arguments just as if you were going to execute on the
# command line
# It is generally good practice to issue a 'cd' command into the directory that contains
# the files
# you want to use or use full path names
# Any parameter or filename that needs to use the current job number of the array number
# range use ${PBS_ARRAYID}
```

3.5 File Transfer (Globus and SFTP)

3.5.1 Overview

For a quick start guide please visit: [Globus: How to Get Started](#).

If you need further assistance beyond this documentation, please contact helpdesk@hpc.wvu.edu.

3.5.2 Transfer Data

Use Globus Online web browser to transfer data files between your PC/workstation and the WVU HPC systems. There are two dedicated servers, data transfer nodes (DTNs), that are connected to WVU's Science DMZ (REX) that allow data to be transferred with low latency and high bandwidth across WVU campus as well as to other locations around the globe. This space is available for all researchers at WVU who need to temporarily store and transfer data.

Logging in and transferring files

Visit the URL [Globus Online](#) to login.

Select West Virginia University from the organizational list and login using your WVU Login username and password.

Transferring data to your local workstations/PC

To learn how to transfer data to your local workstation, you will need to install Globus Connect Personal.

3.5.3 Share Data

Use Globus Online to share your data files with researchers located at other institutions around the world without needing administrative assistance.

Note If you are not using an endpoint that starts with wvu\#, before you can start sharing data using Globus, you must contact Research Computing and request that a shared endpoint be created for you.

3.5.4 Archive Data

Globus Online provides an easy mechanism for researchers to archive their data sets for free to their MIX Google Drive account via a [Google Drive Connector](#). All WVU faculty and students have unlimited storage in Google Drive through their MIX account. Staff can request a MIX account by contacting helpdesk@hpc.wvu.edu. Google Drive gives researchers a safe environment to archive their data sets, which replicates between multiple data centers to prevent data loss. Note: You can also use the Google Drive web interface as well as the [Google Drive Sync Client](#) to transfer files to your Google Drive MIX account; however, the sync client and web interface is best used for small files (Example .docx, .pdf, etc.). Globus Google Drive Connector is optimized for transferring a large number of files as well as very large files, which cannot be done, via the Google Drive Web Interface or Sync Client. You will see the best performance with Google Drive Connector with transferring several large files at once. If you have many small files, it might be best to zip/tar these files first before transferring to Google Drive.

Google Drive is not approved for use of sensitive or protected data sets (i.e. HIPPA, FERPA, ITAR, subject to Export Control laws, etc.). If you need assistance archiving these types of data sets, please contact helpdesk@hpc.wvu.edu.

To archive data to a Google Drive you will need to follow the instructions in the Globus Online Google Drive Connector Instructions, which will guide you on how to make a Shared Endpoint that you can attach to your MIX Google Drive Account to transfer data to/from.

Globus Online Google Drive Connector Instructions

- **Note:** In step one, when searching all endpoints you will need to search for ‘wvu#googledrive’.

To transfer files to your MIX Google Drive Account, review the “Transfer Data” Section above. You will want to select the name of the Shared Endpoint you just created in step 3 as one of the two endpoints. The other endpoint could be a WVU endpoint, such as wvu#hpcdtn or a personal endpoint you created that connects to your workstation/PC.

3.5.5 Publish Data

Use Globus Online to publish your datasets, possibly as a requirement from a granting agency. Published datasets are organized by communities and their member collections and are searchable by other Globus users.

Note If you are not using an endpoint that starts with wvu\#, before you can start sharing data using Globus, you must contact Research Computing and request that a shared endpoint be created for you.

For more information on how to publish your data sets using Globus Online, visit [Globus, Data Publication User Guide](#).

3.6 Web Interface (Open On-Demand)

The term Interactive Scientific Computing consists on using a computer in a similar way as you use a handheld calculator. On a handheld calculator, you type some input and you expect that input to be processed right away to produce results. The result does not need to return immediately, but you expect that calculations are done take an amount of time that allow a human to wait before a new command is submitted.

Interactive Scientific Computing is different from the way we use to work in High Performance Computing (HPC). On a HPC supercomputer, you use the computer with a queue system on what we called non-interactive computing. Supercomputers are large computing devices usually build as clusters of individual computers. Supercomputers in many cases are used by tens or hundreds of users at the same time. For the typical use of Supercomputers, the computations are programmed in advance, the user submit jobs expecting that those jobs start being executed sometime in the future and produce the result that will be analyzed later on.

Despite of the usual usage of a HPC supercomputing, you could have strong motivations for use a computer far more powerful than your own desktop or laptop. Your research have scaled to a point where your desktop computer or laptop is not longer capable of managing the task, you need specialized software packages and you do not want to spent time compiling or installing software and you would like to rely on software that is already present on the HPC cluster.

There are several ways to a HPC cluster for Interactive Computing. You can execute interactive computing directly from the terminal or using a web interface such as Open-On Demand and one interactive environments such as Jupyter or RStudio. We will describe both alternatives.

3.6.1 Interactive Computing from the Terminal

First, connect to the cluster using the instructions described on [here](#):`ref:qs-connect::`. Once on the head node you can request an interactive session with:

```
$> qsub -I
```

This is the simplest command for requesting an interactive session. Under this command, the queue selected will be standby with a walltime of 4 hours. The job will allocate one node and one core on the node for the execution.

You can request more cores for your interactive job, for example:

```
$> qsub -I -l nodes=1:ppn=4
```

On Spruce Knob you can request 16 cores for an entire node. There are a few nodes with 20 and 24 cores but requesting with those numbers will reduce the chances of getting a node for your execution inmediately. On Thorny Flat most nodes have 40 cores. A way of requesting an entire node could be:

```
$> qsub -I -n
```

Requesting more than 1 core does not necessarily means that your code will actually using those extra cores. It all depends on your code being able to work in parallel directly or indirectly. Directly means that the code use some sort of paralelization such as multithreading such as pthreads or OpenMP, or multiprocessing such as MPI, or any other parallelization in languages such as R or Python that offer libraries for explicit parallelism. Indirectly parallelization means that underlying libraries such as FFTW or OpenBLAS could have been compiled with multithreading support meaning that codes that uses those libraries could take indirectly advantage of parallelism.

You can also request GPUs for an interactive job, in that case you have to explicitly request a queue such as `comm_gpu_inter` that offers compute nodes with GPU cards and explicitly request the number of GPUs that you want to use. For example if you want to use one GPU card for your execution use:

```
$> qsub -I -q comm_gpu_inter -l nodes=1:ppn=8:gpus=1
```

After you submit your request for interactive jobs, you will have to wait a few minutes before you are assigned a compute node. After getting access to the compute node, you can load modules and execute commands as you were on a machine. There are several text-based environments for interactive computing.

One is R and you can access it with:

```
$> module load lang/r/4.2.0_gcc112
$> R

R version 4.2.0 (2022-04-22) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> sessionInfo()
R version 4.2.0 (2022-04-22)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Red Hat Enterprise Linux Server 7.9 (Maipo)

Matrix products: default
BLAS/LAPACK: /gpfs20/shared/software/libs/openblas/0.3.19_gcc112/lib/libopenblas_
  ↪skylakep-r0.3.19.so

locale:
[1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8       LC_COLLATE=en_US.UTF-8
[5] LC_MONETARY=en_US.UTF-8   LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8     LC_NAME=C
[9] LC_ADDRESS=C              LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats      graphics    grDevices utils      datasets  methods   base

loaded via a namespace (and not attached):
[1] compiler_4.2.0
>
```

Another text-based interactive environment is IPython, to use it execute for example:

```
$> module load lang/python/cpython_3.10.1_gcc112
```

(continues on next page)

(continued from previous page)

```

Loading gcc version 11.2.0 : lang/gcc/11.2.0
Loading python version cpython_3.10.1_gcc112 : lang/python/cpython_3.10.1_gcc112

$> ipython
Python 3.10.1 (main, Dec 28 2021, 19:48:41) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.30.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import platform

In [2]: platform.machine()
Out[2]: 'x86_64'

In [3]: platform.version()
Out[3]: '#1 SMP Thu Mar 25 21:21:56 UTC 2021'

In [4]: platform.platform()
Out[4]: 'Linux-3.10.0-1160.24.1.el7.x86_64-x86_64-with-glibc2.17'

In [5]: platform.uname()
Out[5]: uname_result(system='Linux', node='trcis001.hpc.wvu.edu', release='3.10.0-1160.
         ↪24.1.el7.x86_64', version='#1 SMP Thu Mar 25 21:21:56 UTC 2021', machine='x86_64')

In [6]: platform.system()
Out[6]: 'Linux'

In [7]: platform.processor()
Out[7]: 'x86_64'

```

Matlab is another software that offers interactive environment. Usually MATLAB is accessed on a graphical interface but MATLAB can also work from a text-based interface:

```

$> module load lang/gcc/9.3.0 matlab/2021a
Loading gcc version 9.3.0 : lang/gcc/9.3.0

$> matlab -nodesktop
MATLAB is selecting SOFTWARE OPENGL rendering.

                                     < M A T L A B (R) >

                                     ↪ Copyright 1984-2021 The MathWorks, Inc.

                                     ↪ R2021a (9.10.0.1602886) 64-bit (glnxa64)

                                     ↪ February 17, 2021

To get started, type doc.
For product information, visit www.mathworks.com.

>>

```

3.6.2 Interactive Computing from a web interface

For taking advantage of WVU's High Performance Computing cluster for interactive scientific computing another alternative is from a web browser. On this lesson you will not have to learn Linux commands, you just need to execute one for the purpose of downloading all the materials for the tutorials but beyond that your interaction will take place on a friendly web interface. You do not have to manually submitting jobs or editing submission scripts, these are tasks very important for HPC but they will be delegated for other lessons.

We will be using a tool, a web-based client portal, that hides all that complexity and allows you to start using powerful computers for your research from a web interface, with minimal effort and fast learning curve.

Several technologies are involved here and it is important to understand how those different pieces are interconnected.

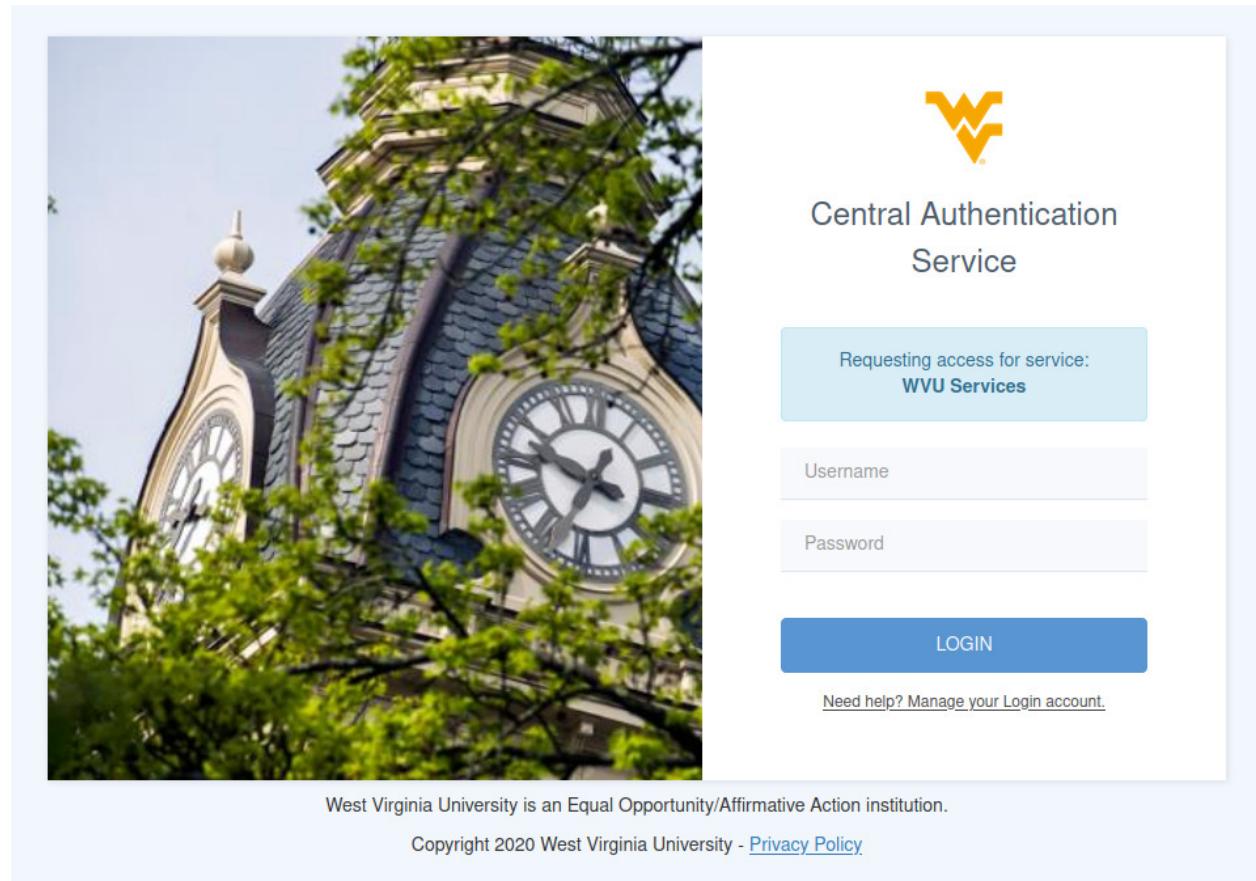
Open OnDemand is a web-based client, based on the Ohio Supercomputer Center's proven "OSC On Demand" platform, that enables HPC centers to install and deploy advanced web and graphical interfaces for their users. HPC resources are accessible from a web browser without the user having to install any special software or plugin.

The path for this tutorial is as follows. First we will demonstrate how to access the open on demand portal. Next we will create Jupyter and RStudio sessions and opening a terminal and a file manager.

3.6.3 Accessing the Dashboard

First, go to [Thorny Flat On Demand Dashboard](#)

The first page you will see is asking for your credentials



The image shows a composite view. On the left is a large, slightly blurred photograph of a building's facade, featuring a prominent clock tower with white faces and black hands. On the right is a screenshot of a web-based login interface. At the top right is the West Virginia University (WVU) logo, which consists of a yellow 'W' and 'V' intertwined. Below the logo, the text "Central Authentication Service" is displayed. A light blue callout box contains the text "Requesting access for service: WVU Services". Below this are two input fields: "Username" and "Password", both currently empty. At the bottom of the form is a large blue "LOGIN" button. Underneath the button, a small link reads "Need help? Manage your Login account." At the very bottom of the image, there is a footer with the text "West Virginia University is an Equal Opportunity/Affirmative Action institution.", "Copyright 2020 West Virginia University - [Privacy Policy](#)", and a small "94" icon.

After entering your credentials and using your DUO authentication you will land on the Open On Demand Dashboard:



RESEARCH COMPUTING

OnDemand provides an integrated, single access point for all of your HPC resources.

From this dashboard you can launch interactive jobs, open terminals and access a file manager, we will see each of those operations in the next sections.

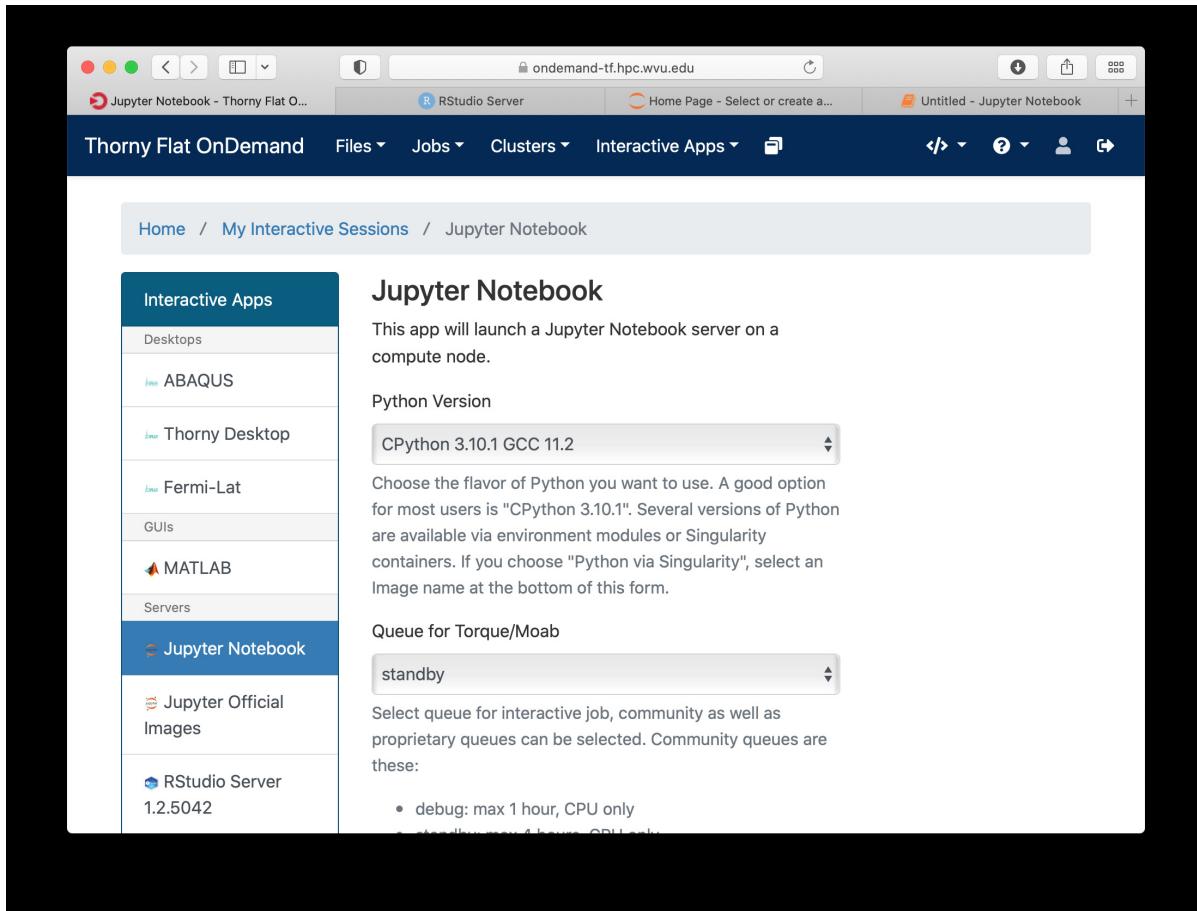
3.6.4 Interactive applications

From the dashboard go to *Interactive Apps*. There are several options there, we will show 2 apps that are currently ready for being used. Jupyter Notebooks and RStudio.

Jupyter

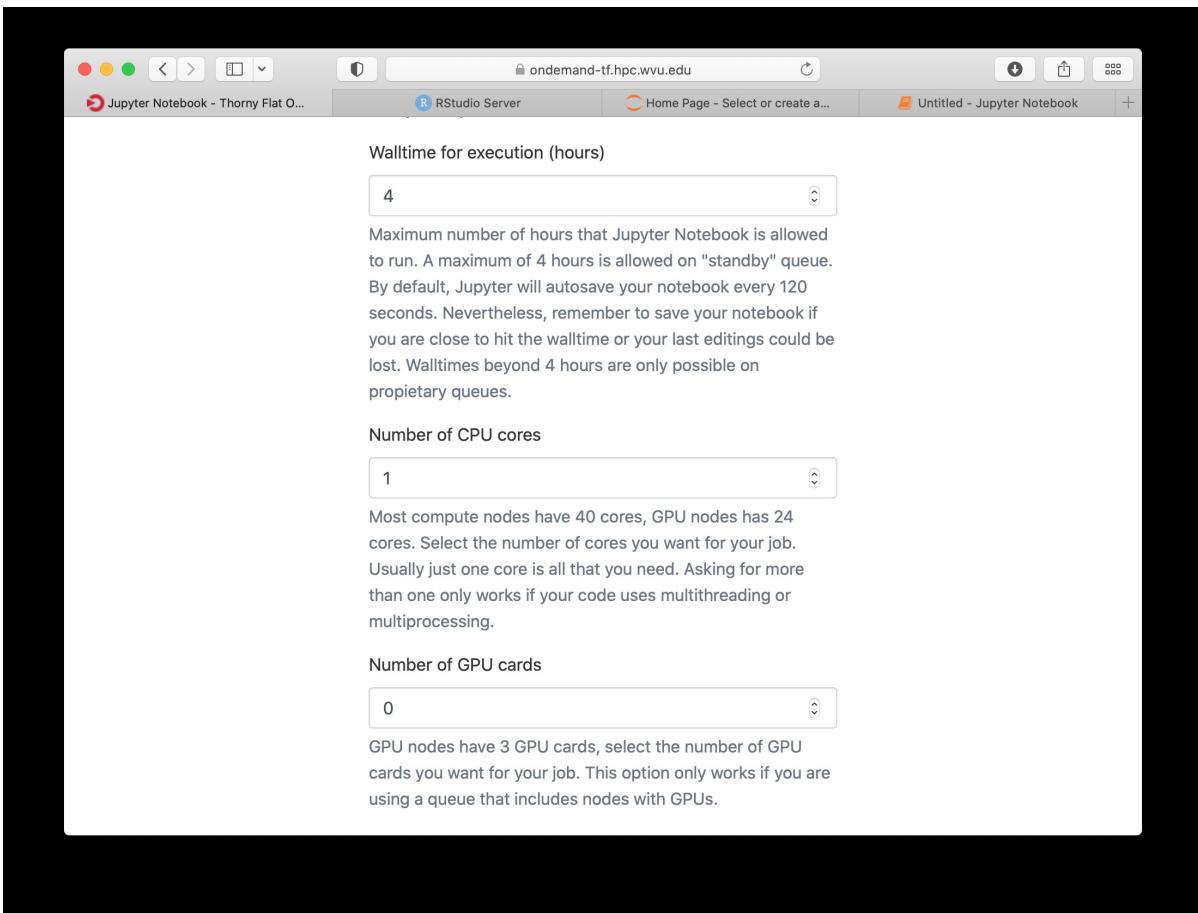
For Jupyter click on *Interactive Apps > Jupyter Notebook*. A form is shown with all the options available to create the Jupyter session.

A good starting point is to select *CPython 3.7.4* as the Python version, select *standby* as the queue and *4 hours* as the wall time. There are options for alternative Python versions, queues and walltimes. A short description of each options is shown on the form.

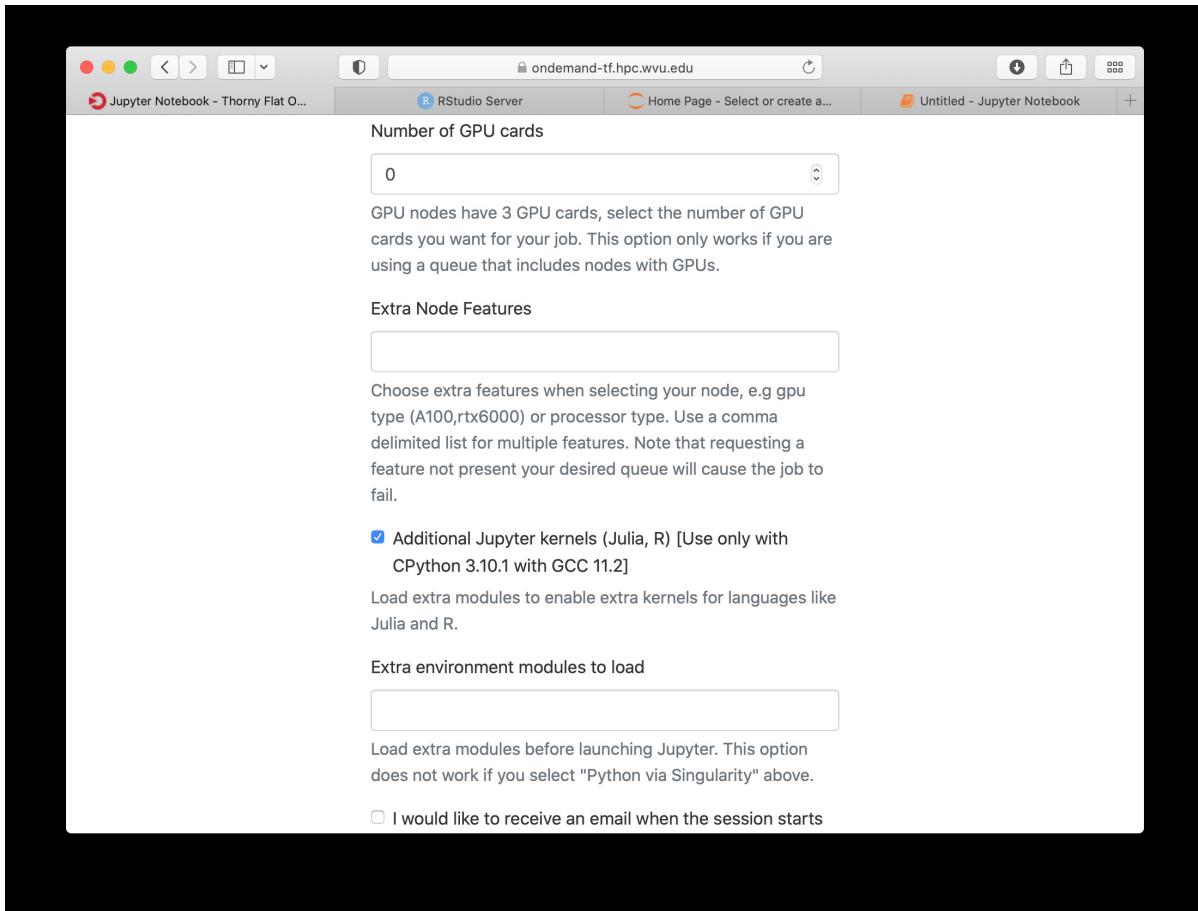


The next fields on the form ask for the number of cores, GPU cards, extra modules and the singularity image in case you have selected that as your Python version.

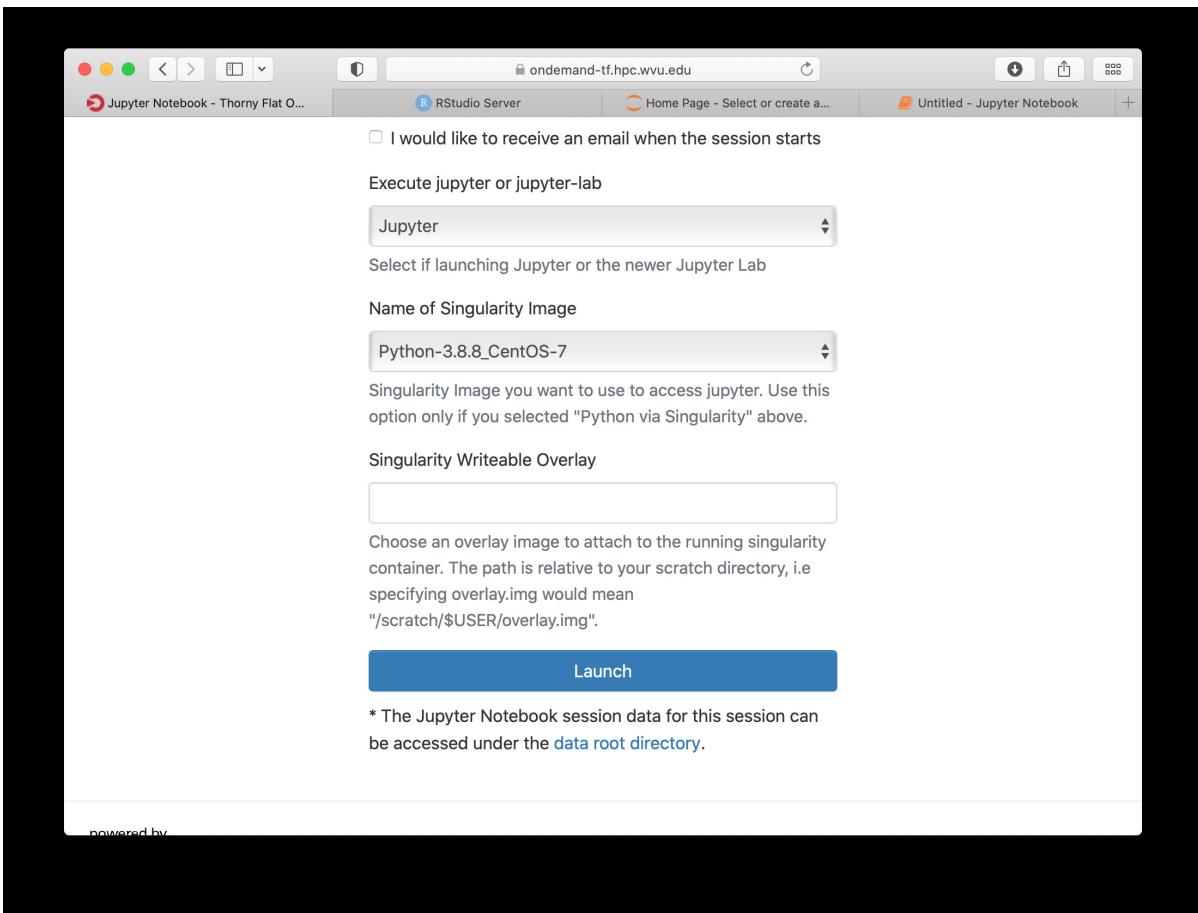
Notice that taking advantage of multiple cores depends on your code being able to use those cores. In the case of Python that usually means that your code is using *multiprocessing* module or you are using *numpy* with multi-threading capabilities. The usage of multiple cores is not something that happens automatically so if you are not sure asking for one core is enough. A similar situation happens with GPUs noticing that only the queue *comm_gpu_inter* give you access to GPUs for community nodes.



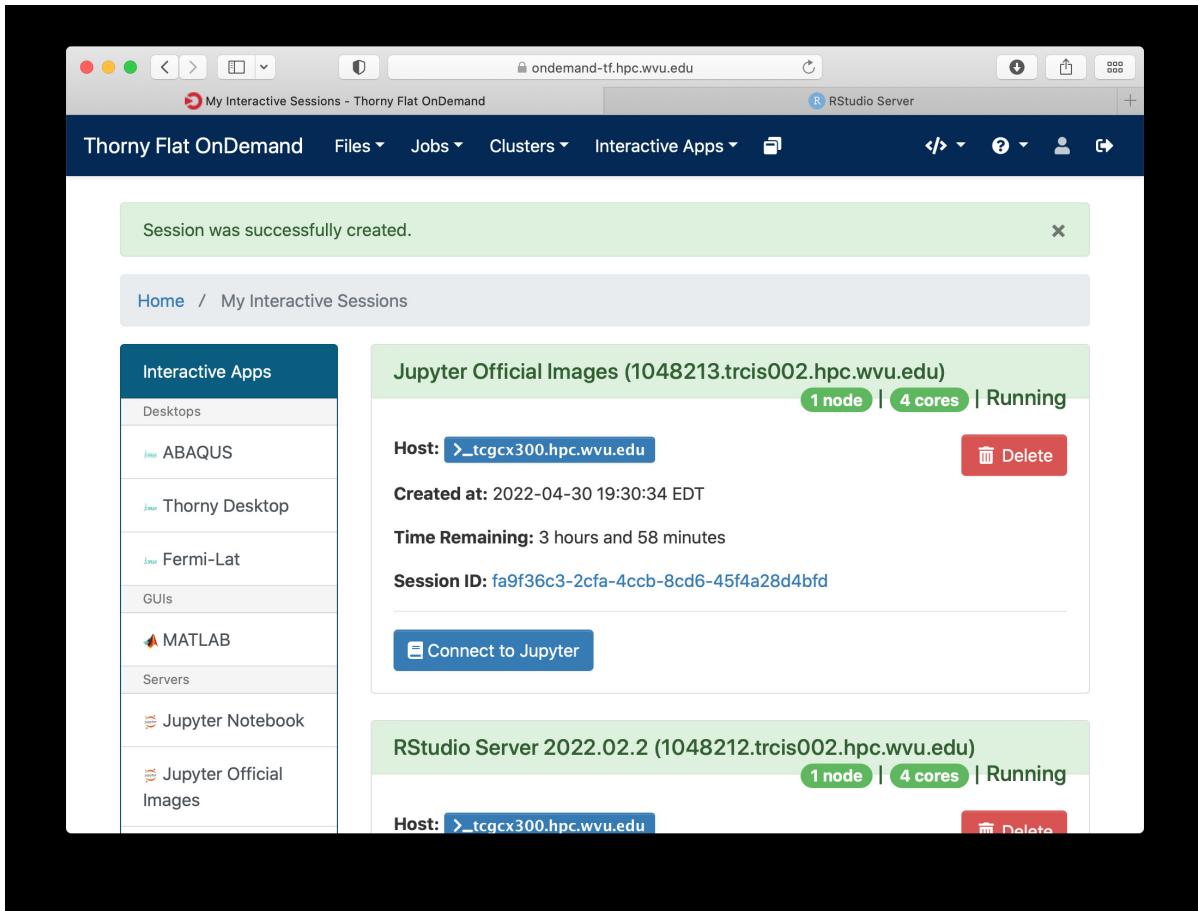
Once you have customize the parameters for your Jupyter session, click on *Launch*. Open On demand will launch a new interactive session and when the interactive session is launched you will get a button to connect to the Jupyter notebook.

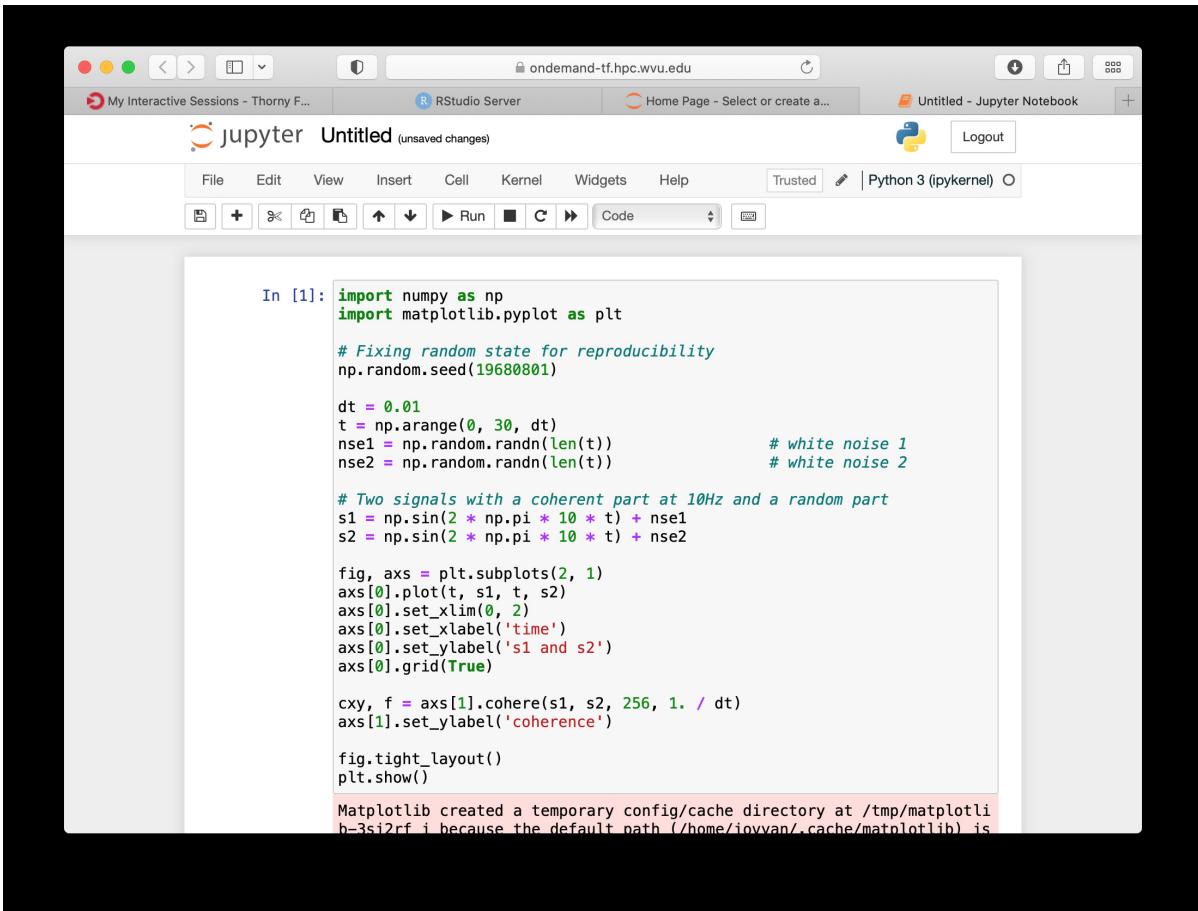


The Jupyter session is launched on a compute node. The Jupyter interface shows as file manager where you can select a notebook to launch, upload one from your local computer or create a new Notebook, go to *New > Python 3* to create a new Jupyter notebook with Python 3 as kernel.



The new notebook give you entries for typing Python instructions that are executed when you type *SHIFT-ENTER*





The screenshot shows a Jupyter Notebook interface running on a web browser. The title bar indicates the session is on 'ondemand-tf.hpc.wvu.edu'. The notebook tab is titled 'Untitled - Jupyter Notebook'. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3 (ipykernel). The toolbar below the menu has icons for file operations like Open, Save, and Run, along with a Code button. The code cell contains the following Python script:

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t)) # white noise 1
nse2 = np.random.randn(len(t)) # white noise 2

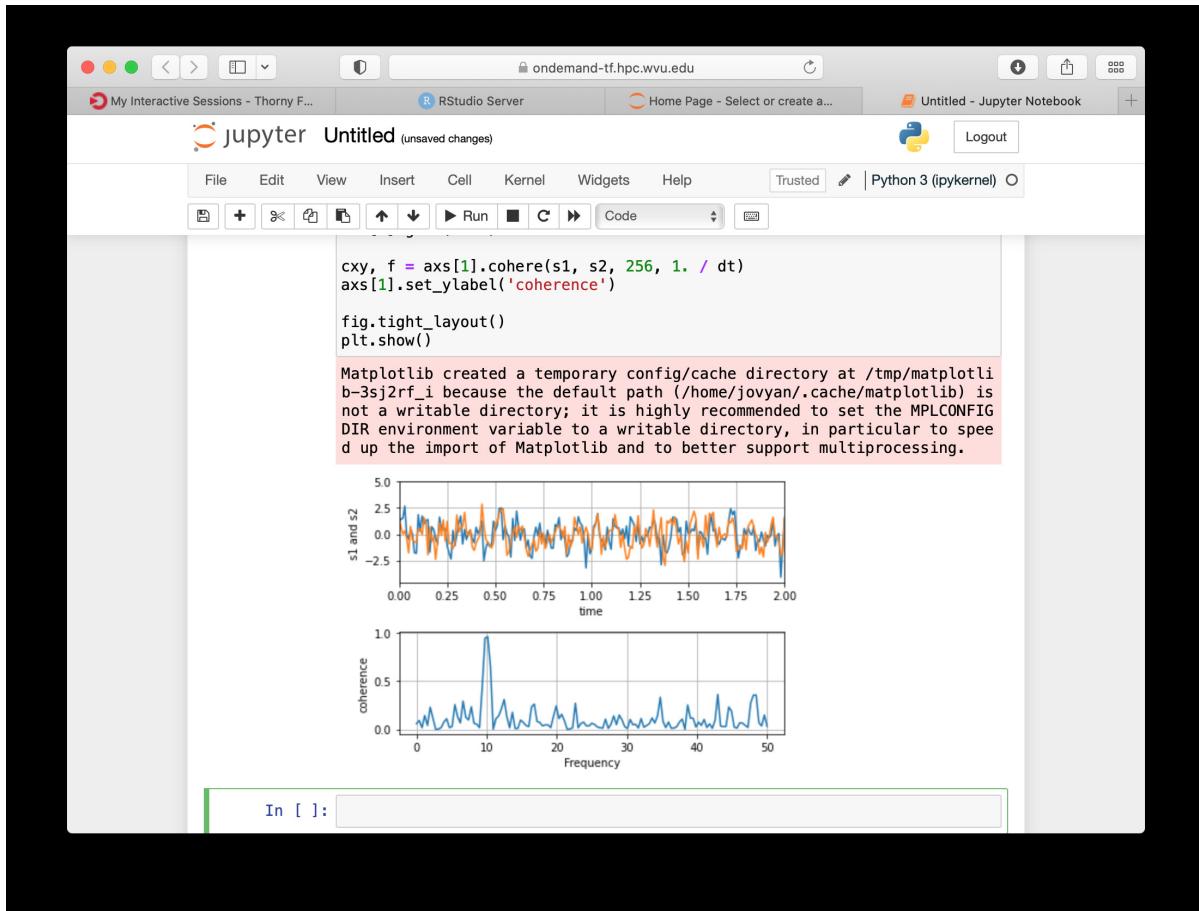
# Two signals with a coherent part at 10Hz and a random part
s1 = np.sin(2 * np.pi * 10 * t) + nse1
s2 = np.sin(2 * np.pi * 10 * t) + nse2

fig, axs = plt.subplots(2, 1)
axs[0].plot(t, s1, t, s2)
axs[0].set_xlim(0, 2)
axs[0].set_xlabel('time')
axs[0].set_ylabel('s1 and s2')
axs[0].grid(True)

cxy, f = axs[1].cohere(s1, s2, 256, 1. / dt)
axs[1].set_ylabel('coherence')

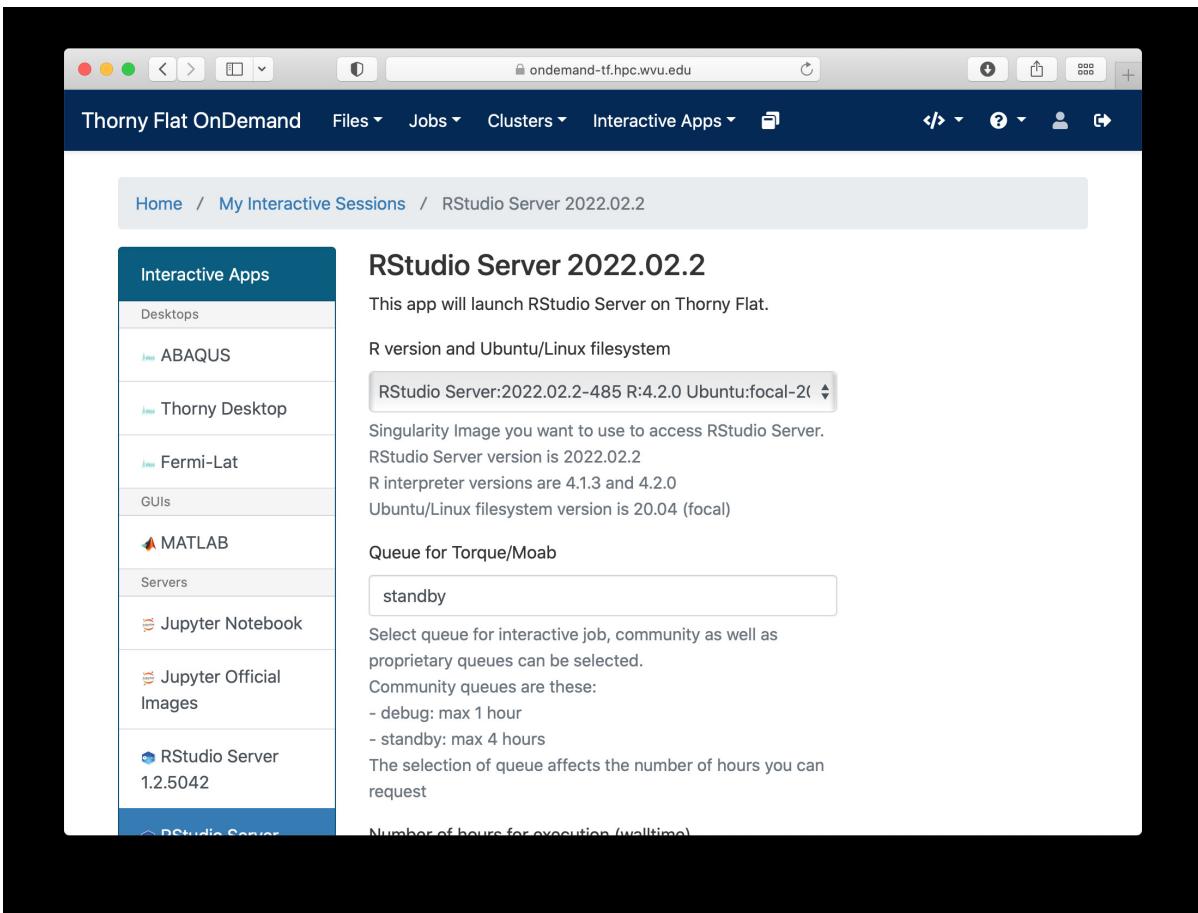
fig.tight_layout()
plt.show()
```

A message at the bottom of the code cell area states: 'Matplotlib created a temporary config/cache directory at /tmp/matplotlib/b-3si2cf i because the default path (/home/joyvan/.cache/matplotlib) is'.

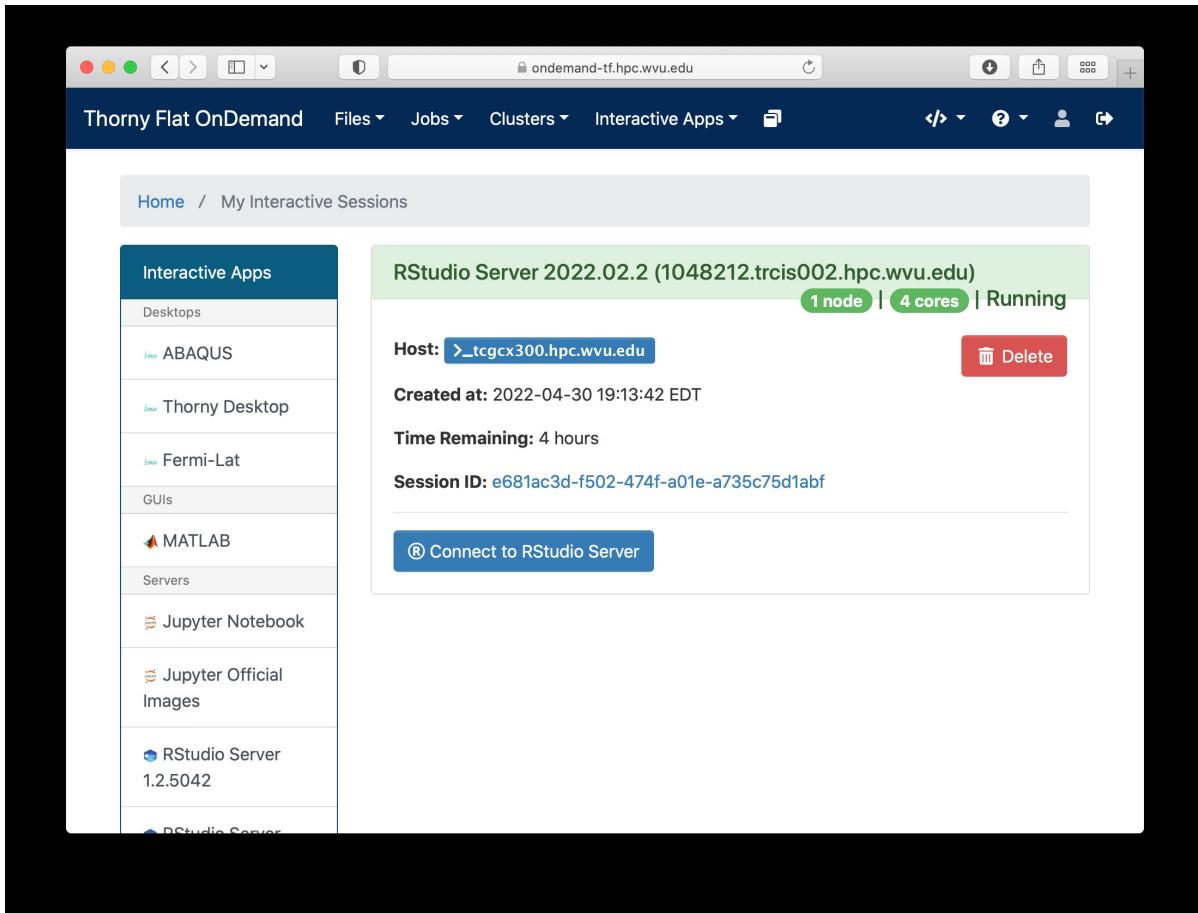


RStudio

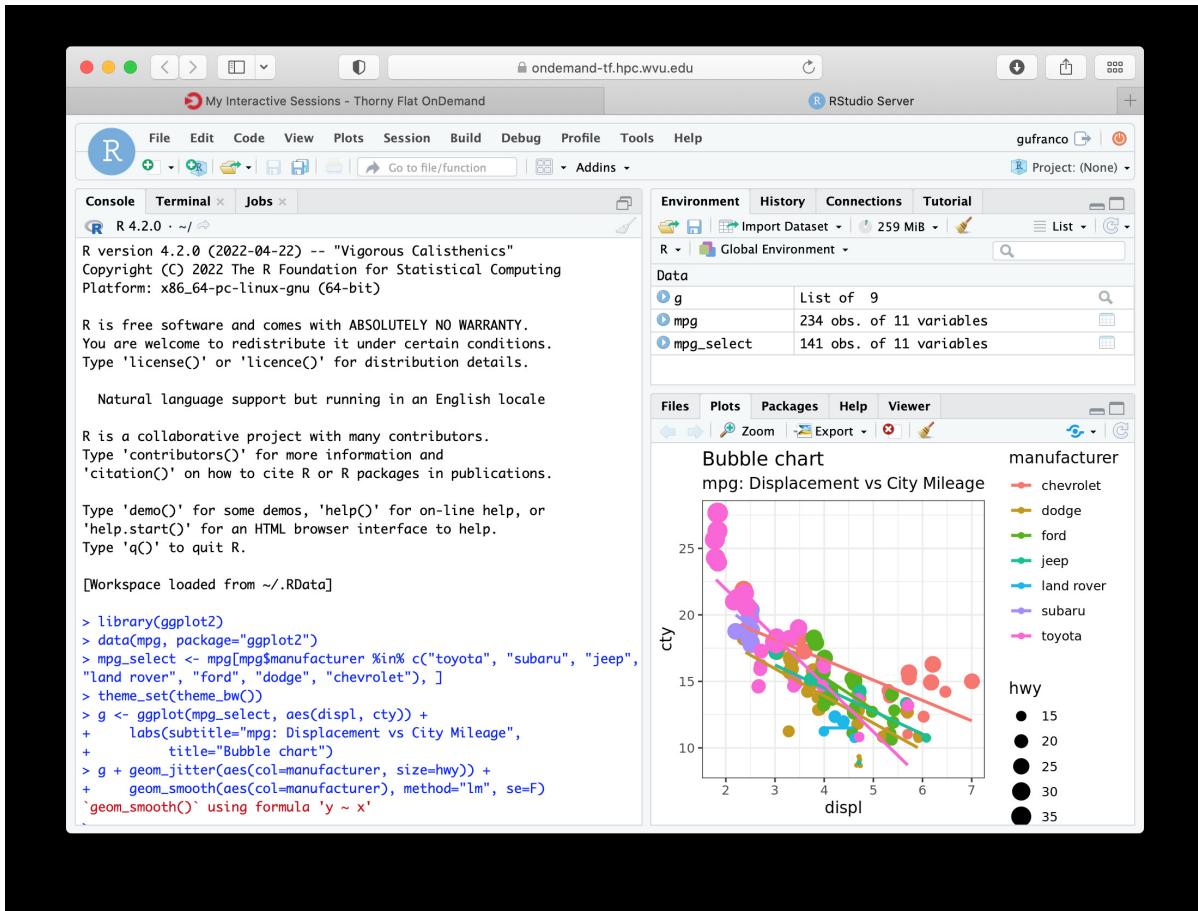
Another popular interactive environment is RStudio, select *Interactive Apps > RStudio*. The options in the form are very few. Select a queue such as *standby*, 4 hours of wall time and 1 core.



When you click on Launch, Open on Demand will create a new interactive session on Thony Flat and when the job starts execution, a button appears to open the session on a new tab.

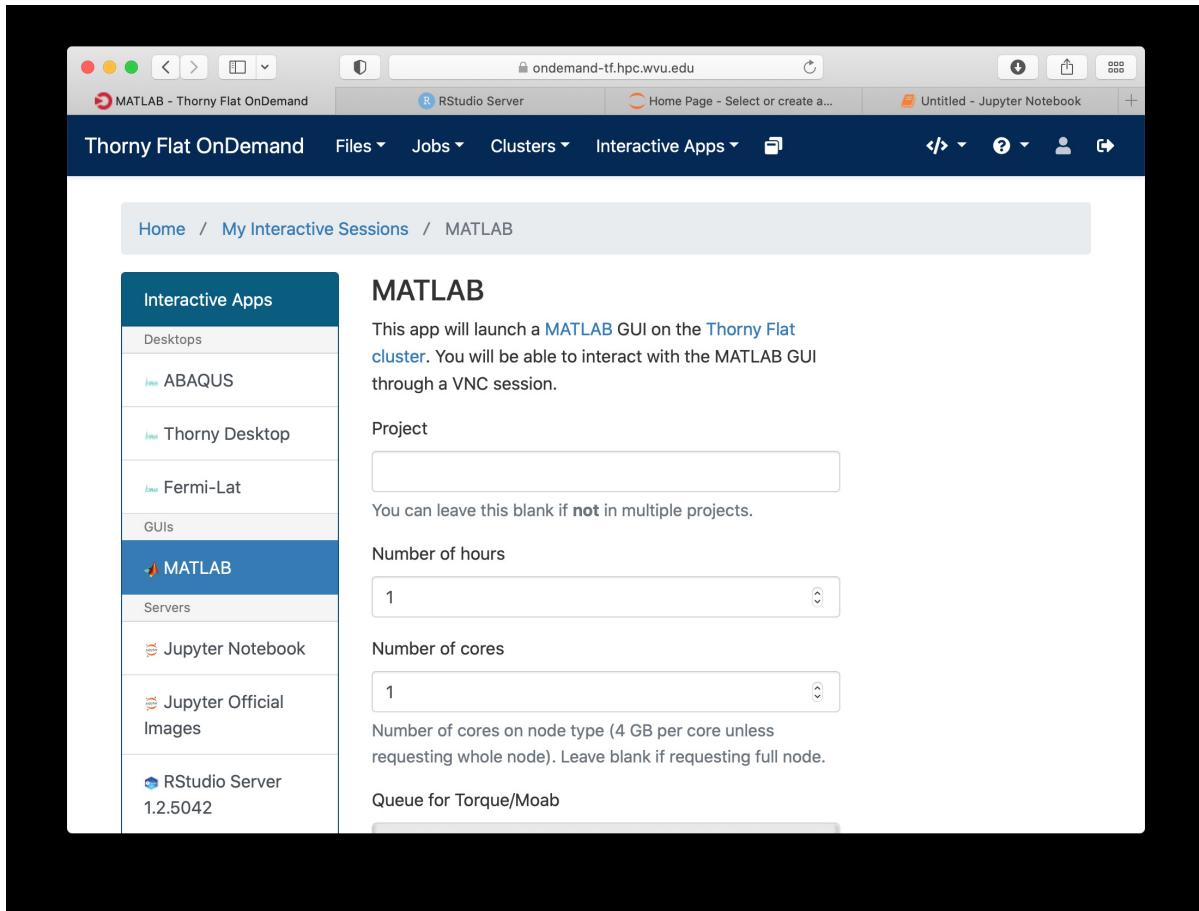


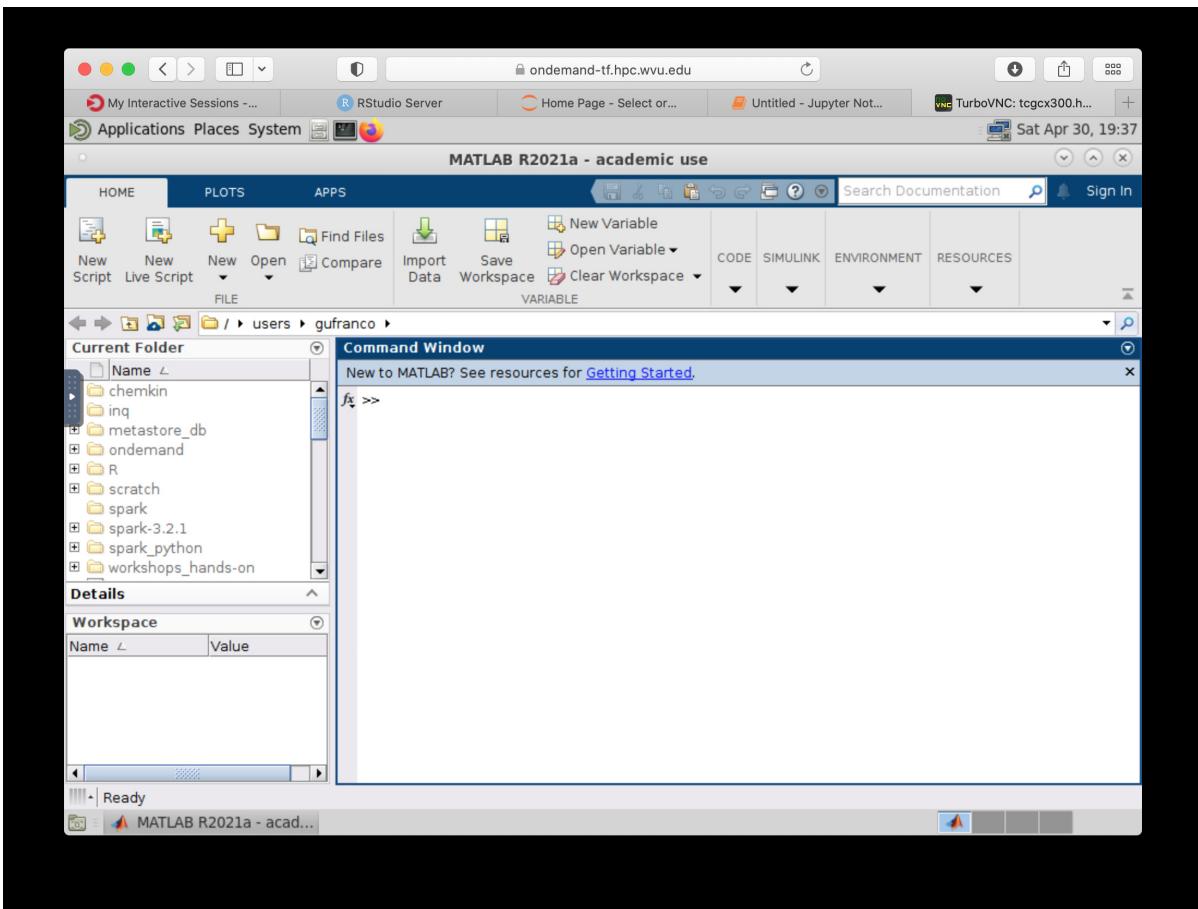
The interface for RStudio shows the commands on the left and variables and plots on the right.

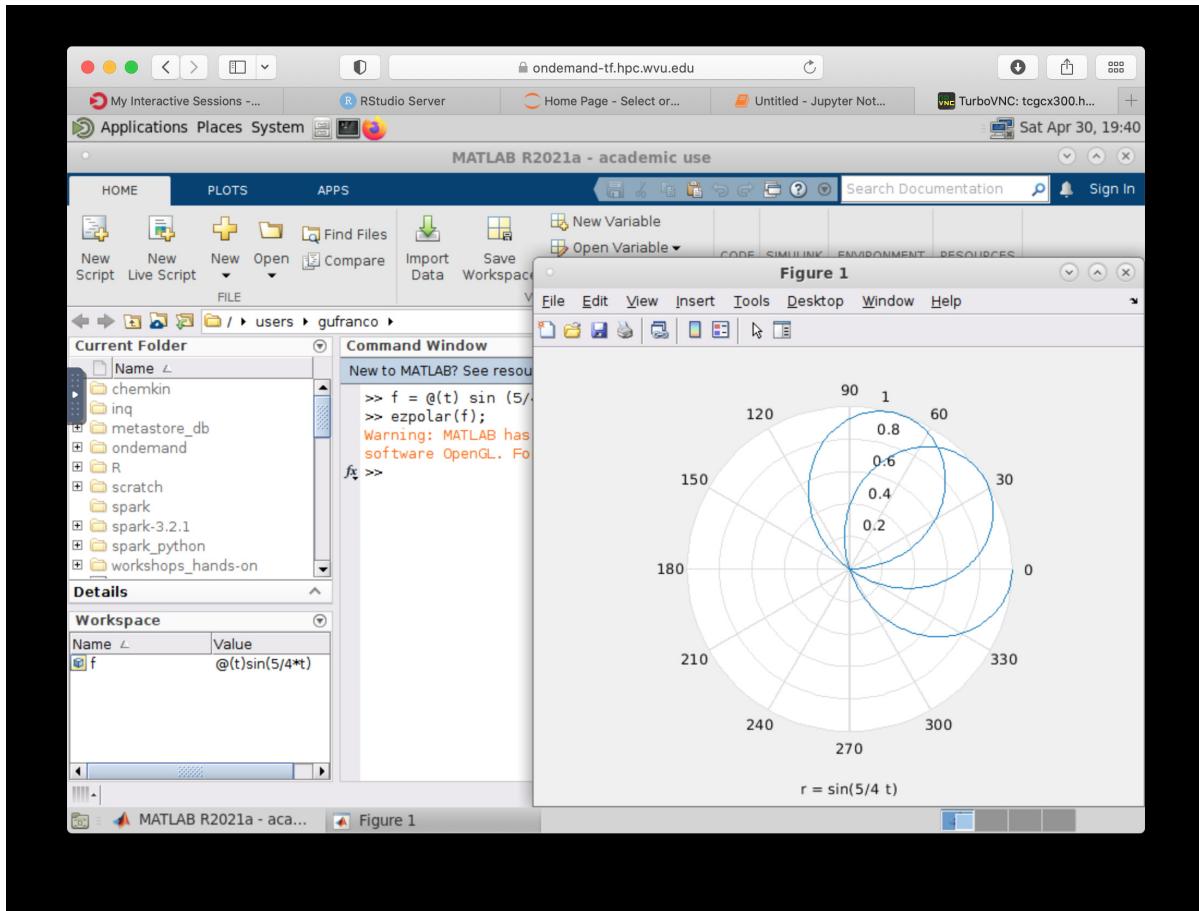


3.6.5 MATLAB

MATLAB is a numeric computing environment and programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. MATLAB can run using Open On-Demand via an app that executes a virtual desktop on a browser tab.



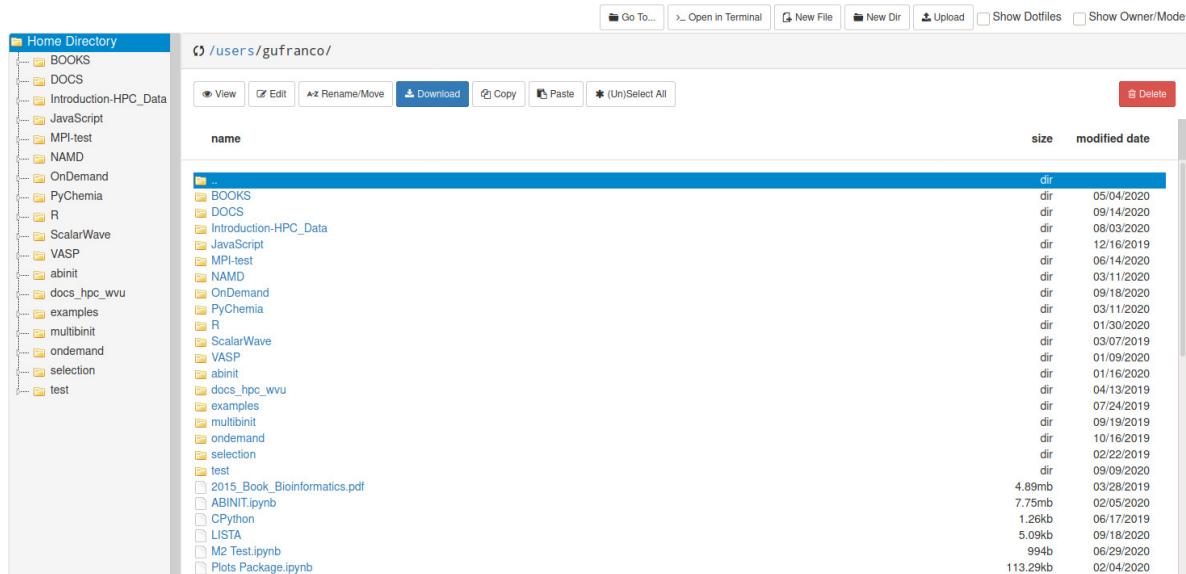




3.6.6 File Manager

The Open On demand dashboard also offers a simple but useful File Manager that give you options to view, edit, download and rename files. It is a simple way to see plots and download individual files to your local computer.

To access the File Manager on the Dashboard, go to *Files > Home Directory*. The File Manager is shown as



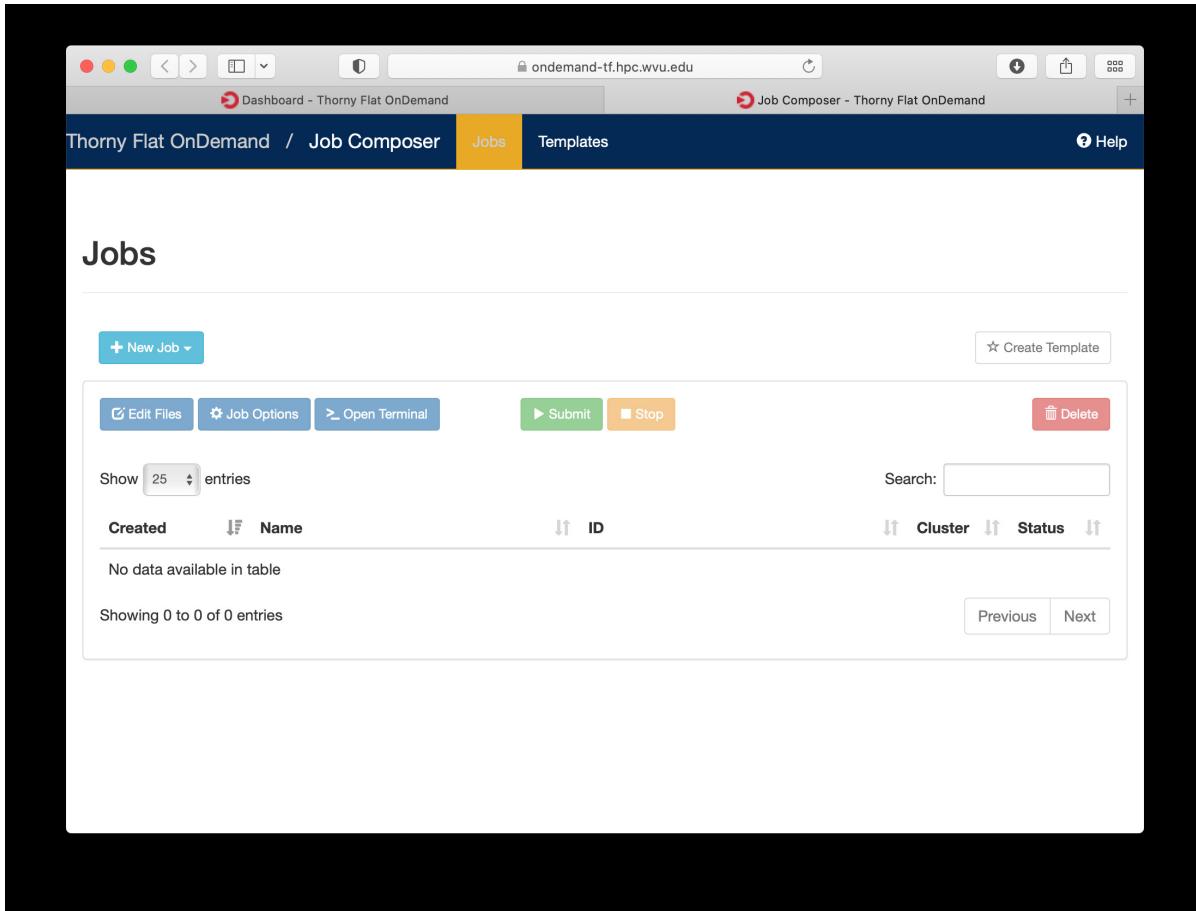
3.6.7 Job manager

Another useful tool integrated with the Dashboard is the Job Manager, you can see the jobs currently submitted on the cluster. Go to *Jobs > Active Jobs* to access the Job Manager

ID	Name	User	Account	Time Used	Queue	Status	Cluster
1048212.trcis002.hpc.wvu.edu	sys/dashboard/ sys/RStudio_2022	gufranco		00:27:11	standby	Running	Thorny Flat
1048213.trcis002.hpc.wvu.edu	sys/dashboard/ sys/Jupyter_Official	gufranco		00:10:19	standby	Running	Thorny Flat
1048214.trcis002.hpc.wvu.edu	sys/dashboard/ sys/bc_matlab	gufranco		00:05:17	standby	Running	Thorny Flat

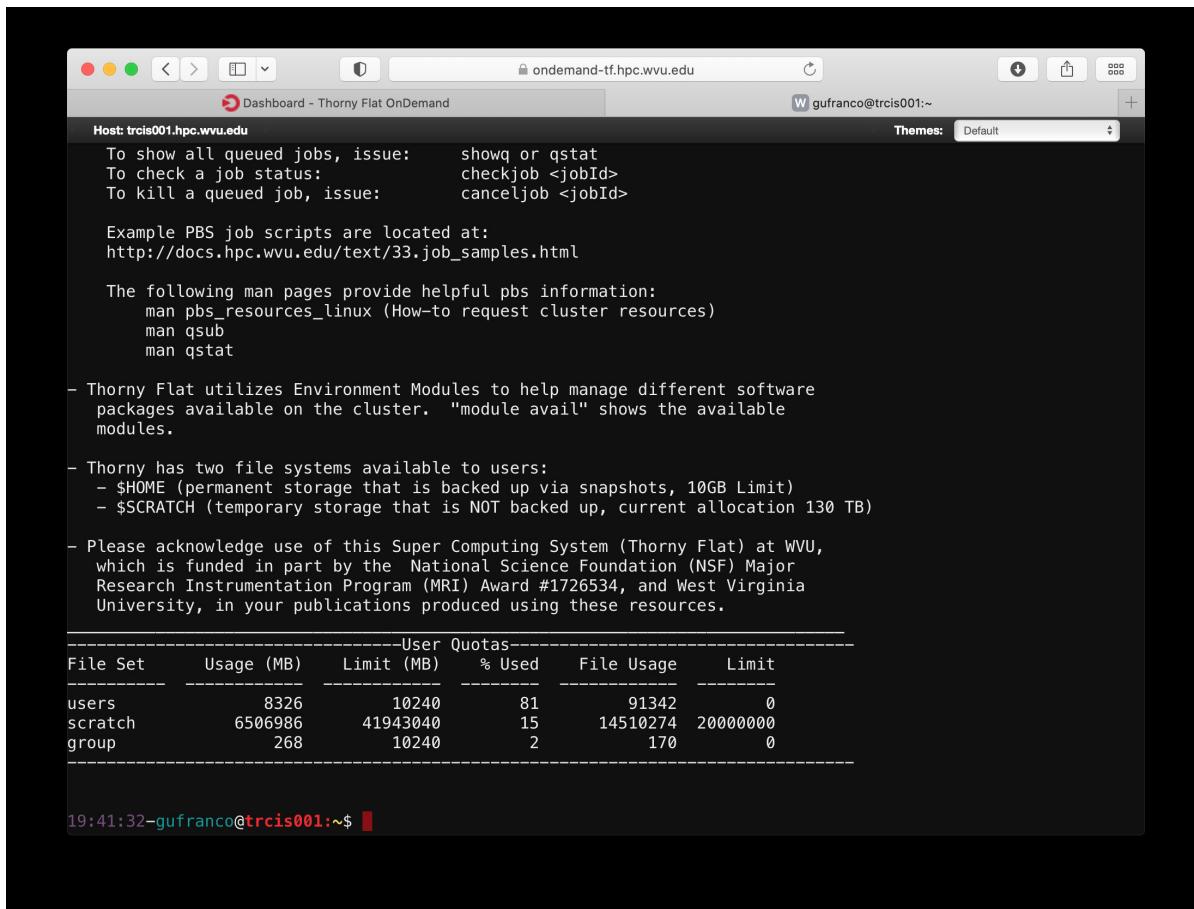
3.6.8 Job composer

Another useful tool integrated with the Dashboard is the Job Composer, you can create jobs and submit them to the cluster via a web interface. Go to *Jobs > Jobs Composer* to access the tool.



3.6.9 Terminal

Finally a Terminal session can be opened from the dashboard, the terminal runs on the head node exactly as a normal connection to the cluster via SSH. It has, however a short limit of time to be alive if no command is typed on it. The terminal is intended for quick commands only.



ADVANCED USAGE

The contents on this section are intended for users who demand more control on the system, the ability to install or execute codes not provided by central installation.

The different topics on this section are mostly independent of each other. The few instances where two topics are connected will be treated as self consistently as possible even with the drawback of some redundancy.

We start with *Singularity Containers* a virtualization software platform that can package an application and its dependencies in a virtual container that can run on any Linux server. Singularity brings many of the features of Docker for HPC usage. Docker a well known platform virtualization software but due to design it requires to run with superuser privileges that cannot be offered on a shared resource like an HPC cluster.

Conda *Conda*, is an open source package management system and environment management system. Originally created for Python programs, it can package and distribute software for any language. We will show how to easily create, save, load and switch between environments. We also include information about installing software from specialized channels.

General Purpose Graphics Processing Units (GPGPUs) is the usage of GPUs for numerical calculation, also called GPU accelerators, GPU-based high performance computers are starting to play a significant role in HPC. On our cluster we have nodes with dedicated GPU cards for computing. In *GPU Computing* we discuss how to use GPUs in several situations.

Environment Modules system is a tool to help users manage their Unix or Linux shell environment, by allowing groups of related environment-variable settings to be made or removed dynamically. On *Environment Modules* we show how to create modules and create a private repository for them.

On section *Jupyter Notebooks* we discuss about Jupyter notebooks. A Jupyter Notebook (formerly IPython Notebook) is a web-based interactive computational environment computational notebooks. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the “.ipynb” extension. Originally created for the Python language nowadays it supports a number of languages including Python, R, Julia and Haskell.

On section *HDF5: Hierarchical Data Format* one example of using HDF5 for storing numerical data in binary format.

4.1 Singularity Containers

Singularity is an open source container solution developed specifically for HPC environments. With Singularity, HPC users can safely bring their own execution environments to the cluster. Unlike other container solutions, Singularity does not require root level permissions to run containers, which allows users to freely control what software stack they wish to use. Provisioning of a container image can be done locally on the user’s machine or on Singularity Hub. The resulting image can then be securely executed on any machine with Singularity installed. Reproduction of results means that a user can now share a single Singularity image file that will ensure a consistent execution environment wherever it is run.

Singularity is a virtualization solution. There are several kinds of Virtualization solutions. From one side we have System Virtual Machines, they provide a substitute for a real machine. They provide functionality needed to execute entire operating systems. In some cases the virtualization even emulate different architectures and allow execution of software applications and operating systems written for another CPU or architecture. From the other side we have OS-level virtualization where the kernel allows the existence of multiple isolated user-space instances. Singularity belongs to the latest.

Containers are similar to Virtual Machines, however, the differences are enough to consider them different technologies and those differences are very important for HPC. Virtual Machines takes up a lot of system resources. Each Virtual Machine (VM) runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run. This quickly adds up to a lot of precious RAM and CPU cycles, valuable resources for HPC.

In contrast, all that a container requires is enough of an operating system, supporting programs and libraries, and system resources to run a specific program. From the user perspective, a container is in most cases a single file that contains the file system, ie a rather complete Unix filesystem tree with all libraries, executables, and data that are needed for a given workflow or scientific computation.

There are several container solutions and one prominent example is Docker. Docker is intended to run *trusted containers* by *trusted users*. This model do not fit well in HPC where computer clusters are accessed by a number of users whose privileges cannot be escalated to superuser level. For that reason Singularity utilizes a very different security paradigm. This is a required feature for implementation within any multi-user environment with *untrusted users* running *untrusted containers*.

For more information about Singularity and complete documentation see: <https://singularity.lbl.gov/quickstart>

4.1.1 Activating Singularity

There are two main components in Singularity containers. The runtime executable and the singularity image.

The *Container Runtime* or *Singularity runtime* is designed to leverage the above mentioned container formats. The runtime program is called singularity and is capable not only of running singularity images but also to create them.

To activate singularity on your current terminal, on both, Spruce and Thorny execute:

```
module load singularity/2.5.2
```

That will modify the *\$PATH* variable allowing you to execute the main command for singularity:

```
singularity
```

4.1.2 Singularity Images

The second component are *Singularity images*. On Spruce and Thorny we have a number of centrally managed images:

```
/shared/software/containers
```

or using the variable:

```
$SNG_PATH
```

The list of Singularity images on Spruce and Thorny is the same, the table below shows the current images available.

Image Name	Description	Based on	GUI	GPU
centos-final.simg	Example of image using libgraph 1.0.2, run examples “circle”, “julia” and “sample”	centos:latest	Yes	No
dakota-6.10-release-public-rhel7.simg	Dakota 6.10: Sandia software for optimization and Uncertainty Quantification	centos:latest	No	No
docs_hpc_wvu.simg	Contains Sphinx, pandoc and scripts to create documentation for https://docs.hpc.wvu.edu	ubuntu:bionic	No	No
dolmades.simg	Windows Apps under Linux using Singularity (using wine)	c1t4r/dolmades	Yes?	No
Grass-7.4.0.simg	GRASS (Geographic Resources Analysis Support System), open source GIS	ubuntu:trusty	Yes	No
Grass-7.4.simg	GRASS (Geographic Resources Analysis Support System), open source GIS	ubuntu:trusty	Yes	No
Grass-7.6.1.simg	GRASS (Geographic Resources Analysis Support System), open source GIS	ubuntu:bionic	Yes	No
Jupyter-5.2.2_Python-3.6.8.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks	ubuntu:bionic	Yes	No
jupyter_conda.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks	continuumio/miniconda3	Yes	No
jupyter.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks	ubuntu:bionic	Yes	No
jupyter-xenial.simg	Python 3.5.2 with a number of Scientific Packages and Jupyter Notebooks	ubuntu:xenial	Yes	No
Keras-2.1.4_TensorFlow-1.5.0.simg	Neural Networks and Deep Learning with Keras 2.1.4 and TensorFlow 1.5.0	gw000/keras-full:2.1.4	Yes	Yes
loos.simg	Lightweight Object-Oriented Structure library (LOOS)	ubuntu:trusty	No	No
mini-conda3_firefox.simg	Jupyter from miniconda with Firefox	continuumio/miniconda3	Yes	No
miniconda3.simg	Jupyter from miniconda without firefox	continuumio/miniconda3	Yes	No
ParaView-5.6.0.simg	ParaView 5.6: open-source, multi-platform data analysis and visualization	ubuntu:bionic	Yes	No
RStudio-desktop-1.2.1335_R-3.4.4.simg	RStudio Desktop 1.2 with R 3.4.4	jekriske/r-base	Yes	No
RStudio-server-1.2.1335_R-3.4.4.simg	RStudio Server 1.2 with R 3.4.4	nickjer/singularity-r	Yes	No
singularity-rstudio.simg	RStudio Server 1.2	nickjer/singularity-r	Yes	No
Stacks-2.1.simg	Stacks: Pipeline for building loci from short-read sequences like illumina	ubuntu:trusty-20170817	No	No
Stacks-2.4.simg	Stacks: Pipeline for building loci from short-read sequences like illumina	ubuntu:trusty	No	No
Tensorflow-1.13.1-gpu-py3-jupyter.simg	TensorFlow with support for GPUs	tensorflow/tensorflow:1.13.1-gpu-py3-jupyter	Yes	No
Tensorflow-1.13.1-py3-jupyter.simg	TensorFlow without support for GPUs	tensorflow/tensorflow:1.13.1-py3-jupyter	Yes	No
Tensor-Flow_gpu_py3.simg	TensorFlow with support for GPUs	tensorflow/tensorflow:latest-gpu-py3	No	Yes
Visit-2.13.2.simg	Visit: Interactive, scalable, visualization, animation and analysis tool	ubuntu:trusty	No	No
Visit-3.0.simg	Visit: Interactive, scalable, visualization, animation and analysis tool	centos:latest	No	No
wkhtmltox-0.12.simg	wkhtmltopdf command line tools to render HTML into PDF and other image formats	ubuntu:trusty	No	No
4.1. Singularity Containers				115
wkhtmltox.simg	wkhtmltopdf command line tools to render HTML into PDF and other image formats	ubuntu:trusty	No	No

4.1.3 Interactive Job with X11 forwarding

Several images above are intended for interactive computing. In those cases you ensure that you connect to the cluster with X11 forwarding, before asking for an interactive job. From Linux or MacOS you can connect via SSH with X11 forwarding using:

```
ssh -X <username>@spruce.hpc.wvu.edu
```

If you are using MacOS you need a X Window System on your Mac. You can install XQuartz to get it <https://www.xquartz.org/>

If you are using Windows you will need a X11 Server, for example using MobaXterm <https://mobaxterm.mobatek.net/>

Once you have login into the cluster, create an interactive job with the following command line, in this case we are using *standby* as queue but any other queue is valid.:

```
qsub -X -I -q standby
```

Once you get inside a compute node, load the module:

```
module load singularity/2.5.2
```

After loading the module the command singularity is available for usage, and you can get a shell inside the image with:

```
singularity shell ${SNG_PATH}/<Image Name>
```

4.1.4 Non-interactive execution with Submission scripts

In this case you do not need to export X11, just login into Spruce or Thorny:

```
ssh <username>@spruce.hpc.wvu.edu
```

Once you have login into the cluster, create a submission script, (name the file *runjob.pbs* for example), in this case we are using *standby* as queue but any other queue is valid.

```
#!/bin/sh

#PBS -N JOB
#PBS -l nodes=1:ppn=1
#PBS -l walltime=04:00:00
#PBS -m ae
#PBS -q standby

module load singularity/2.5.2

singularity exec ${SNG_PATH}/<Image Name> <command_or_script_to_run>
```

Submit your job with:

```
qsub runjob.pbs
```

4.1.5 GPU Support and Singularity

To get access to GPUs from inside the container use the argument `--nv` either for the `shell` or `exec` subcommands. Lets demonstrate this with an interactive example using Tensorflow on spruce

Assuming that you are now log into Spruce execute:

```
$> qsub -I -q comm_gpu
```

After a few seconds you get into a compute node:

```
salg0001:~$>
```

Next step is to activate Singularity:

```
$> module load singularity/2.5.2
```

Lets use for example `Keras-2.1.4_TensorFlow-1.5.0.simg` one of the images centrally managed and located at `$SNG_PATH`:

```
$> singularity shell --nv $SNG_PATH/Keras-2.1.4_TensorFlow-1.5.0.simg
```

Lets check that from inside the image the GPUs are visible:

```
$> nvidia-smi
Wed Sep 25 18:06:29 2019
+-----+
| NVIDIA-SMI 396.26           Driver Version: 396.26 |
+-----+
| GPU  Name     Persistence-M| Bus-Id     Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====
|  0  Tesla K20m        Off  | 00000000:08:00.0 Off |          0 |
| N/A   39C   P0    76W / 225W |      96MiB /  4743MiB |     44%     Default |
+-----+
|  1  Tesla K20m        Off  | 00000000:24:00.0 Off |          0 |
| N/A   40C   P0    49W / 225W |      0MiB /  4743MiB |     0%     Default |
+-----+
|  2  Tesla K20m        Off  | 00000000:27:00.0 Off |          0 |
| N/A   34C   P0    52W / 225W |      0MiB /  4743MiB |    91%     Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name          Usage     |
| =====+=====+=====+=====
|
```

Now we can use IPython and Tensorflow:

```
$> ipython3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.
```

(continues on next page)

(continued from previous page)

```
In [1]: import tensorflow as tf

In [2]: tf.test.is_built_with_cuda()
Out[2]: True

In [3]: tf.test.is_gpu_available()
2019-09-25 18:18:37.476402: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your
→CPU supports instructions that this TensorFlow binary was not compiled to use: SSE4.1_
→SSE4.2 AVX
2019-09-25 18:18:40.271869: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] ↵
→Found device 0 with properties:
name: Tesla K20m major: 3 minor: 5 memoryClockRate(GHz): 0.7055
pciBusID: 0000:08:00.0
totalMemory: 4.63GiB freeMemory: 4.48GiB
2019-09-25 18:18:40.390182: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] ↵
→Found device 1 with properties:
name: Tesla K20m major: 3 minor: 5 memoryClockRate(GHz): 0.7055
pciBusID: 0000:24:00.0
totalMemory: 4.63GiB freeMemory: 4.56GiB
2019-09-25 18:18:40.508266: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] ↵
→Found device 2 with properties:
name: Tesla K20m major: 3 minor: 5 memoryClockRate(GHz): 0.7055
pciBusID: 0000:27:00.0
totalMemory: 4.63GiB freeMemory: 4.56GiB
2019-09-25 18:18:40.508594: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1120] ↵
→Device peer to peer matrix
2019-09-25 18:18:40.508681: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1126] ↵
→DMA: 0 1 2
2019-09-25 18:18:40.508697: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1136] 0: ↵
→ Y N N
2019-09-25 18:18:40.508705: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1136] 1: ↵
→ N Y Y
2019-09-25 18:18:40.508713: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1136] 2: ↵
→ N Y Y
2019-09-25 18:18:40.508730: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↵
→Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: Tesla K20m, pci bus_
→id: 0000:08:00.0, compute capability: 3.5)
2019-09-25 18:18:40.508742: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↵
→Creating TensorFlow device (/device:GPU:1) -> (device: 1, name: Tesla K20m, pci bus_
→id: 0000:24:00.0, compute capability: 3.5)
2019-09-25 18:18:40.508753: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↵
→Creating TensorFlow device (/device:GPU:2) -> (device: 2, name: Tesla K20m, pci bus_
→id: 0000:27:00.0, compute capability: 3.5)
Out[3]: True
```

Those two checks ensure that TensorFlow was indeed compiled with GPU support and the TensorFlow is able to see the 3 GPUs installed on the machine.

Now we can run a very simple calculation using the GPUs:

```
In [4]: with tf.device('/gpu:0'):
...:     a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
```

(continues on next page)

(continued from previous page)

```

...:     b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
...:     c = tf.matmul(a, b)
...:
...: with tf.Session() as sess:
...:     print (sess.run(c))
...:

2019-09-25 18:22:40.750833: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: Tesla K20m, pci bus ↴
↳ id: 0000:08:00.0, compute capability: 3.5)
2019-09-25 18:22:40.750901: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:1) -> (device: 1, name: Tesla K20m, pci bus ↴
↳ id: 0000:24:00.0, compute capability: 3.5)
2019-09-25 18:22:40.750914: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:2) -> (device: 2, name: Tesla K20m, pci bus ↴
↳ id: 0000:27:00.0, compute capability: 3.5)
[[ 22.  28.]
 [ 49.  64.]]

```

Notice that the calculation was performed on /gpu:0, as the machine has 3 GPUs you can also compute on /gpu:1 and /gpu:2 Another way of checking the available devices is with:

```

In [5]: with tf.Session() as sess:
...:     devices = sess.list_devices()
...:

2019-09-25 18:27:51.067844: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:0) -> (device: 0, name: Tesla K20m, pci bus ↴
↳ id: 0000:08:00.0, compute capability: 3.5)
2019-09-25 18:27:51.067891: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:1) -> (device: 1, name: Tesla K20m, pci bus ↴
↳ id: 0000:24:00.0, compute capability: 3.5)
2019-09-25 18:27:51.067904: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] ↴
↳ Creating TensorFlow device (/device:GPU:2) -> (device: 2, name: Tesla K20m, pci bus ↴
↳ id: 0000:27:00.0, compute capability: 3.5)

```

4.2 Conda

Conda is an open source package management system and environment management system. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments. It was created for Python programs, but it can package and distribute software for any language.

Conda as a package manager helps you find and install packages. If you need a package that requires a different version of Python, you do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, you can set up a totally separate environment to run that different version of Python, while continuing to run your usual version of Python in your normal environment.

There are two installers for conda, Anaconda and Miniconda.

4.2.1 Anaconda vs Minicoda

Anaconda is a downloadable, free, open source, high-performance and optimized Python and R distribution. Anaconda includes conda, conda-build, Python, and 100+ automatically installed, open source scientific packages and their dependencies that have been tested to work well together, including SciPy, NumPy and many others. Ananconda is more suited to be installed on a desktop environment as you get after installation a fairly complete environment for scientific computing.

From the other side Miniconda is free minimal installer for conda. Miniconda is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on and a small number of other useful packages, including pip, zlib and a few others. Miniconda is more suited for HPC environments where a minimal installation is all that is needed and users can create their own environments as needed.

4.2.2 Activating Conda on Spruce and Thorny Flat

On Spruce you can activate conda with:

```
source /shared/software/miniconda3/etc/profile.d/conda.sh
```

On Thorny Flat the command to activate conda is:

```
source /shared/software/conda/conda_init.sh
```

After activation your are positioned on the base environment.

4.2.3 Conda Environments

Conda allows you to create separate environments containing files, packages and their dependencies that will not interact with other environments.

When you begin using conda, you already have a default environment named base. You cannot install packages on the base environment. Yous should create new environments for installing packages. Try to keep separate environments for different packages or group of packages. That reduces the chances of incompatibility between them

4.2.4 Knowing which environments are available

By the time of writing this tutorial Spruce offers three environments centrally installed:

```
$> conda info --envs

# conda environments:
#
base                  /scratch/gufranco/conda-intel
genomics-core          /shared/software/miniconda3
jupyter                /shared/software/miniconda3/envs/jupyter
python27               /shared/software/miniconda3/envs/python27
python35               /shared/software/miniconda3/envs/python35
python36               /shared/software/miniconda3/envs/python36
python37               /shared/software/miniconda3/envs/python37
qiime2-2018.8          /shared/software/miniconda3/envs/qiime2-2018.8
snowflakes             /shared/software/miniconda3/envs/snowflakes
tpd0001               /shared/software/miniconda3/envs/tpd0001
```

4.2.5 Activating an existing environment

Suppose that you want to use the environment called “tpd0001”, to achieve that execute:

```
conda activate tpd0001
```

4.2.6 Deactivating the current environment

The current environment can be deactivated with:

```
conda deactivate
```

If you are in the base environment, the deactivation will not have any effect. You are always at least on the base environment.

4.2.7 Create a new environment

We will name the environment snowflakes and install the package BioPython. At the Anaconda Prompt or in your terminal window, type the following:

```
conda create --name snowflakes
```

or if you want also to install a package you can execute:

```
conda create --name snowflakes biopython
```

Conda collects metadata about the package and its dependencies and produces an installation plan:

```
## Package Plan ##

environment location: /shared/software/conda/envs/snowflakes

added / updated specs:
- biopython
```

The following packages will be downloaded:

package		build	
_libgcc_mutex-0.1		main	3 KB
biopython-1.74		py37h7b6447c_0	2.0 MB
blas-1.0		mkl	6 KB
ca-certificates-2019.8.28		0	132 KB
certifi-2019.9.11		py37_0	154 KB
intel-openmp-2019.4		243	729 KB
libedit-3.1.20181209		hc058e9b_0	163 KB
libffi-3.2.1		hd88cf55_4	40 KB
libgcc-ng-9.1.0		hdf63c60_0	5.1 MB
libgfortran-ng-7.3.0		hdf63c60_0	1006 KB
libstdcxx-ng-9.1.0		hdf63c60_0	3.1 MB
mkl-2019.4		243	131.2 MB

(continues on next page)

(continued from previous page)

mkl-service-2.3.0	py37he904b0f_0	218 KB
mkl_fft-1.0.14	py37ha843d7b_0	155 KB
mkl_random-1.1.0	py37hd6b4f25_0	321 KB
ncurses-6.1	he6710b0_1	777 KB
numpy-1.17.2	py37haad9e8e_0	4 KB
numpy-base-1.17.2	py37hde5b4d6_0	4.2 MB
openssl-1.1.1d	h7b6447c_1	3.7 MB
pip-19.2.3	py37_0	1.9 MB
python-3.7.4	h265db76_1	32.1 MB
readline-7.0	h7b6447c_5	324 KB
setuptools-41.2.0	py37_0	630 KB
six-1.12.0	py37_0	23 KB
sqlite-3.29.0	h7b6447c_0	1.1 MB
tk-8.6.8	hbc83047_0	2.8 MB
wheel-0.33.6	py37_0	40 KB
xz-5.2.4	h14c3975_4	283 KB
zlib-1.2.11	h7b6447c_3	103 KB
<hr/>		
Total:		192.2 MB

Conda asks if you want to proceed with the plan:

```
Proceed ([y]/n)? y
Type "y" and press Enter to proceed.
```

After that, conda, download and installs the packages creating a new environment for you. The final message shows how to activate and deactivate the environment:

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate snowflakes
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Each environment is isolated from other conda environments and that allow you to keep several environments with different packages on them or different versions of the same packages. As the message shows you activate the environment with:

```
conda activate snowflakes
```

Notice that when you activate a new environment, the prompt changes adding a prefix in parenthesis to indicate on which conda environment are you using at that moment. To check the environments available execute:

```
conda env list
```

or:

```
conda info --envs
```

4.2.8 Conda and Python

When you create a new environment, conda installs the same Python version used to install conda on Spruce and Thorny (3.7). If you want to use a different version of Python, for example Python 2.7, simply create a new environment and specify the version of Python that you want:

```
conda create --name python27 python=2.7
```

You activate the environment:

```
conda activate python27
```

And verify the python version:

```
$ python --version
Python 2.7.16 :: Anaconda, Inc.
```

Conda has packages for versions of python for 2.7, 3.5, 3.6 and 3.7

4.2.9 Managing packages and channels

New packages can be installed to existing conda environments. First search for packages with:

```
conda search mkl
```

Packages are stored in repositories called **channels**. By default, conda search on the pkgs/main channel only. However, there are many other packages on several other channels.

The most prominent channels to search for packages are **intel**, **conda-forge** and **bioconda**. To search for packages there execute:

```
conda search -c intel mkl
conda search -c conda-forge nitime
conda search -c bioconda blast
```

Packages can be installed on the current environment with:

```
conda install -c conda-forge nitime
```

In this case conda will pick the most recent version of the package compatible with the packages already present on the current environment. You can also be very selective on version and build that you want for the package. First get the list of versions and builds for the package that you want:

```
$ conda search -c intel mkl
Loading channels: done
# Name          Version       Build Channel
mkl           2017.0.3     intel_6  intel
mkl           2017.0.4     h4c4d0af_0  pkgs/main
```

(continues on next page)

(continued from previous page)

mk1	2018.0.0	hb491cac_4	pkgs/main
mk1	2018.0.0	intel_4	intel
mk1	2018.0.1	h19d6760_4	pkgs/main
mk1	2018.0.1	intel_4	intel
mk1	2018.0.2	1	pkgs/main
mk1	2018.0.2	intel_1	intel
mk1	2018.0.3	1	pkgs/main
mk1	2018.0.3	intel_1	intel
mk1	2019.0	117	pkgs/main
mk1	2019.0	118	pkgs/main
mk1	2019.0	intel_117	intel
mk1	2019.1	144	pkgs/main
mk1	2019.1	intel_144	intel
mk1	2019.2	intel_187	intel
mk1	2019.3	199	pkgs/main
mk1	2019.3	intel_199	intel
mk1	2019.4	243	pkgs/main
mk1	2019.4	intel_243	intel
mk1	2019.5	intel_281	intel

Now, install the package declaring the version and build:

```
$ conda install -c intel mkl=2019.4=intel_243
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /users/gufranco/.conda/envs/test

added / updated specs:
- mkl==2019.4=intel_243
```

The following packages will be downloaded:

package	build		
intel-openmp-2019.5	intel_281	888 KB	intel
mkl-2019.4	intel_243	204.1 MB	intel
tbb-2019.8	intel_281	874 KB	intel
		Total:	205.8 MB

The following NEW packages will be INSTALLED:

```
intel-openmp      intel/linux-64::intel-openmp-2019.5-intel_281
mkl              intel/linux-64::mkl-2019.4-intel_243
tbb              intel/linux-64::tbb-2019.8-intel_281
```

Proceed ([y]/n)?

(continues on next page)

(continued from previous page)

```
Downloading and Extracting Packages
tbb-2019.8           | 874 KB    | ###### #####################################################
→ ##### ##################################################### | 100%
mkl-2019.4           | 204.1 MB   | ###### #####################################################
→ ##### ##################################################### | 100%
intel-openmp-2019.5  | 888 KB    | ###### #####################################################
→ ##### ##################################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

4.2.10 Creating a new environment from a YML file

You can create your own environment, one easy way of doing that is via a YML file that describes the channels and packages that you want on your environment. The YML file will look like this, for a simple case when you want one env for bowtie2 (bowtie2.yml)

```
name: spruce-bowtie2
channels:
- bioconda
- conda-forge
- defaults
dependencies:
- bowtie2
```

Another example is this YML file for installing a curated set of basic genomics codes that requires just a few dependencies. (`biocore.yml`)

```
name: biocode
channels:
- bioconda
- conda-forge
- defaults
dependencies:
- bamtools
- bcftools
- bedtools
- hmmer
- muscle
- raxml
- samtools
- sga
- soapdenovo-trans
- soapdenovo2
- sra-tools
- vcftools
- velvet
```

To create an environment from those YML files you can select one location on your scratch folder

```
conda env create -p $SCRATCH/bowtie2 -f bowtie2.yml
```

or for the biocore.yml

```
conda env create -p $SCRATCH/biocore -f biocore.yml
```

By default, new environments are created inside your \$HOME folder on \$HOME/.conda

4.2.11 Listing the packages inside one environment

Bowtie2 has a number of dependencies (19 dependencies for 1 package) Notice that only bowtie2 comes from bioconda channel. All other packages are part of conda-forge, a lower level channel.

```
$ conda activate $SCRATCH/bowtie2
$ conda list
# packages in environment at /scratch/gufranco/bowtie2:
#
# Name            Version          Build  Channel
bowtie2           2.3.4.2         py36h2d50403_0  bioconda
bzip2             1.0.6           h470a237_2    conda-forge
ca-certificates   2018.8.24      ha4d7672_0    conda-forge
certifi            2018.8.24      py36_1        conda-forge
libffi             3.2.1           hfc679d8_5   conda-forge
libgcc-ng          7.2.0           hdf63c60_3   conda-forge
libstdcxx-ng       7.2.0           hdf63c60_3   conda-forge
ncurses            6.1             hfc679d8_1   conda-forge
openssl            1.0.2p          h470a237_0   conda-forge
perl               5.26.2          h470a237_0   conda-forge
pip                18.0            py36_1        conda-forge
python              3.6.6          h5001a0f_0   conda-forge
readline            7.0             haf1bffa_1  conda-forge
setuptools          40.2.0         py36_0        conda-forge
sqlite              3.24.0          h2f33b56_1   conda-forge
tk                 8.6.8            0            conda-forge
wheel               0.31.1          py36_1        conda-forge
xz                 5.2.4           h470a237_1   conda-forge
zlib               1.2.11          h470a237_3   conda-forge
```

4.2.12 Using a conda environment in a submission script

To execute software in a non-interactive job you need to source the main script, activate the environment that contains the software you need, execute the the scientific code and deactivate the environment. This is a simple example showing that for bowtie2

```
#!/bin/bash

#PBS -N MY_JOB
#PBS -q standby
#PBS -j oe
#PBS -l nodes=1:ppn=2
```

(continues on next page)

(continued from previous page)

```
source /shared/software/miniconda3/etc/profile.d/conda.sh
conda activate $SCRATCH/bowtie2

bowtie2 .....

conda deactivate
```

4.2.13 Deleting a environment

To remove an environment you can just execute this command.

```
conda remove --all -p $SCRATCH/bowtie2
```

4.2.14 More documentation

Conda Documentation

[<https://conda.io/docs/user-guide/tasks/manage-environments.html#Managing environments>]

Using Bioconda — Bioconda documentation

Available packages — Bioconda documentation

4.2.15 Downloading Miniconda

Miniconda can be downloaded from:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
sh Miniconda3-latest-Linux-x86_64.sh
```

4.2.16 No installing anything

Just load the module:

```
module purge
module load genomics/qiime
```

This module will load python 2.7.3 and qiime on top of that

4.3 GPU Computing

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Their highly parallel structure makes them more efficient than general-purpose central processing units (CPUs) for algorithms that process large blocks of data in parallel.

As GPUs become more powerful they start being used to perform numerical calculations in what is called a general purpose graphics processing unit (GPGPU). Due to their design GPUs are generally suited to high-throughput type computations that exhibit data-parallelism. One typical case is in Single Operations Multiple Data (SIMD) operations.

Today GPU computing is an integral part of an HPC environment. Three of the 10 most powerful supercomputers in the world today (2019) take advantage of GPU acceleration.

Both of our clusters, Spruce Knob and Thorny Flat have some nodes with GPUs for jobs that can take advantage of them.

4.3.1 Hardware configurations

The GPUs on Spruce Knob are from the Kepler GPU microarchitecture (K20 GPU accelerator). On Thorny Flat the GPUs are all from the Pascal GPU microarchitecture (NVIDIA QUADRO P6000)

Spruce Knob

Spruce has 2 community nodes with GPUs. One has 1x NVIDIA K20m and the second one has 3x NVIDIA K20m.

NVIDIA K20	Specifications
Launch	November 12, 2012
Chips	1x GK110
NVIDIA CUDA Cores	2496
Base Clock	706
Max Boost clock	758
Bus type	GDDR5
Bus width	320-bit
GPU Memory Size	5 GB
Clock (MT/s)	5200
Memory Bandwidth	208 (GB/s)
Single Precision (MAD or FMA)	3524
Double Precision (FMA)	1175
Cuda compute ability	3.5
Thermal Design Power (TDP)	225 W
Form Factor	Full Height, Dual Slot

Thorny Flat

Thorny has 6 community nodes with 3x NVIDIA P6000 per node.

NVIDIA QUADRO P6000	Specifications
GPU Memory Size	24 GB GDDR5X
Memory Interface	384-bit
Memory Bandwidth	432 GB/s
NVIDIA CUDA Cores	3840
System Interface	PCI Express 3.0 x16
Max Power Consumption	250 W
Thermal Solution	Active
Form Factor	Dual Slot, Full Height
Compute APIs	CUDA, OpenCL

4.3.2 Interactive computing with GPUs

On Spruce we have one queue for GPU computing. In the case of interactive jobs execute:

```
qsub -I -q comm_gpu -l nodes=1:ppn=1:gpus=1
```

On Thorny Flat we have separate queues for interactive and non-interactive jobs. In the case of interactive jobs, execute:

```
qsub -I -q comm_gpu_inter -l nodes=1:ppn=1:gpus=3
```

4.3.3 Checking the presence of GPUs

The command *nvidia-smi* can be used to monitor the presence and status of the GPUs on the current compute node. For example on Spruce the command will return something like:

```
$> nvidia-smi
Tue Sep 24 15:48:45 2019
+-----+
| NVIDIA-SMI 396.26           Driver Version: 396.26 |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====
|  0  Tesla K20m        Off  | 00000000:08:00.0 Off |                  0 |
| N/A   40C     P0    73W / 225W |    96MiB /  4743MiB |     44%     Default |
+-----+
|  1  Tesla K20m        Off  | 00000000:24:00.0 Off |                  0 |
| N/A   40C     P0    49W / 225W |    0MiB /  4743MiB |      0%     Default |
+-----+
|  2  Tesla K20m        Off  | 00000000:27:00.0 Off |                  0 |
| N/A   34C     P0    52W / 225W |    0MiB /  4743MiB |     91%     Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name          Usage     |
| =====+=====+=====+=====
|
```

On Thorny Flat, the same command returns something like:

```
$> nvidia-smi
Tue Sep 24 15:56:16 2019
+-----+
| NVIDIA-SMI 410.48           Driver Version: 410.48 |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====
|  0  Quadro P6000        Off  | 00000000:37:00.0 Off |                  Off |
| 26%   53C     P0    176W / 250W |    523MiB / 24449MiB |     94%     E. Process |
+-----+
```

(continues on next page)

(continued from previous page)

1 Quadro P6000	Off	00000000:AF:00.0 Off		Off	
26% 53C P0	152W / 250W	527MiB / 24449MiB	94%	E. Process	
<hr/>					
2 Quadro P6000	Off	00000000:D8:00.0 Off		Off	
26% 50C P0	162W / 250W	523MiB / 24449MiB	96%	E. Process	
<hr/>					
<hr/>					
Processes:			GPU Memory		
GPU PID Type	Process name		Usage		
<hr/>					
<hr/>					

4.4 Environment Modules

The Environment Modules system is a tool to help users manage their Unix or Linux shell environment, by allowing groups of related environment-variable settings to be made or removed dynamically.

The Environment Modules are stratified in several tiers and under several groups. Here we will show some of those key groups in particular for Thorney Flat.

4.4.1 Programming Languages

----- /shared/modulefiles/tier0 -----		
<hr/>		
lang/gcc/11.2.0		lang/nvidia/nvhpc-byo-compiler/21.3
↳ intelpython_2.7.14		lang/python/
lang/gcc/8.2.0		lang/nvidia/nvhpc-nompi/20.7
↳ intelpython_2.7.16		lang/python/
lang/gcc/9.3.0		lang/nvidia/nvhpc-nompi/21.11
↳ intelpython3_2019.5		lang/python/
lang/go/1.15.6		lang/nvidia/nvhpc-nompi/21.3
↳ intelpython3_2020.2		lang/python/
lang/intel/2018		lang/pgi/19.10
↳ intelpython_3.6.3		lang/python/
lang/intel/2018_u4		lang/pgi/19.4
↳ intelpython_3.6.9		lang/python/
lang/intel/2019		lang/python/cpython_3.10.1_gcc112
↳ intelpython_3.9.7		lang/python/
lang/intel/2019_u5		lang/python/cpython_3.7.4_gcc82
↳ pypy2.7-v7.3.2-linux64		lang/python/
lang/java/jdk-17.0.1		lang/python/cpython_3.8.10_gcc82
↳ pypy3.6-v7.1.1-thorney		lang/python/
lang/julia/1.6.5_gcc112		lang/python/cpython_3.8.10_gcc93
↳ pypy3.6-v7.3.2-linux64		lang/python/
lang/julia/1.7.1_gcc112		lang/python/cpython_3.8.12_gcc112
↳ pypy3.7-v7.3.2-linux64		lang/python/
lang/nvidia/nvhpc/20.7		lang/python/cpython_3.9.5_gcc82
↳ gcc112		lang/r/3.6.3_

(continues on next page)

(continued from previous page)

lang/nvidia/nvhpc/21.11 ↳ gcc112	lang/python/cpython_3.9.5_gcc93	lang/r/4.0.5_
lang/nvidia/nvhpc/21.3 ↳ gcc112	lang/python/cpython_3.9.7_gcc93	lang/r/4.1.2_
lang/nvidia/nvhpc-byo-compiler/20.7 ↳ gcc112	lang/python/cpython_3.9.9_gcc112	lang/r/4.1.3_
lang/nvidia/nvhpc-byo-compiler/21.11 ↳ refblas/3.10_gcc112	lang/python/intelpython2_2019.5	

4.4.2 Libraries

----- /shared/modulefiles/tier0 -----		
libs/armadillo/9.9_gcc93 ↳ netcdf/fortran-4.5.3_gcc82	libs/hdf5/1.12.0_gcc82	libs/
libs/arpack-ng/3.8.0_intel21 ↳ netcdf/fortran-4.5.3_gcc82_ompi405	libs/hdf5/1.12.0_gcc82_ompi4.0.5	libs/
libs/arpack-ng/3.8.0_intel21_impi21 ↳ netcdf/fortran-4.5.3_gcc93	libs/hdf5/1.12.0_gcc93	libs/
libs/atompaw/4.1.0.5_gcc48 ↳ netcdf/fortran-4.5.3_gcc93_mpic341	libs/hdf5/1.12.0_gcc93_mpic341	libs/
libs/atompaw/4.1.0.5_intel18 ↳ netcdf/fortran-4.5.3_gcc93_mvap235	libs/hdf5/1.12.0_gcc93_mvap235	libs/
libs/boost/1.41_gcc112 ↳ netcdf/fortran-4.5.3_intel19	libs/hdf5/1.12.0_gcc93_ompi31	libs/
libs/boost/1.78_gcc112_mpic343 ↳ netcdf/fortran-4.5.3_intel21	libs/hdf5/1.12.0_gcc93_ompi4.0.5	libs/
libs/boost/1.78_gcc112_mvap236 ↳ netcdf/fortran-4.5.3_intel21_impi21	libs/hdf5/1.12.0_intel19	libs/
libs/boost/1.78_gcc112_ompi412 ↳ netlib/3.8.0_gcc82	libs/hdf5/1.12.0_v12_gcc93	libs/
libs/boost/1.78_gcc48 ↳ netlib/3.8.0_intel18	libs/hdf5/1.12.1_gcc112	libs/
libs/boost/1.78_gcc93 ↳ openblas/0.3.13_gcc48	libs/hdf5/1.12.1_gcc112_mpic343	libs/
libs/cfitsio/4.0.0_gcc112 ↳ openblas/0.3.13_gcc82	libs/hdf5/1.12.1_gcc112_mvap236	libs/
libs/eigen/3.3.7 ↳ openblas/0.3.13_gcc93	libs/hdf5/1.12.1_gcc112_ompi412	libs/
libs/fftw/3.3.10_gcc112 ↳ openblas/0.3.13_intel18	libs/hdf5/1.12.1_intel21	libs/
libs/fftw/3.3.10_gcc112_mpic343 ↳ openblas/0.3.13_intel19	libs/hdf5/1.12.1_intel21_impi21	libs/
libs/fftw/3.3.10_gcc112_mvap236 ↳ openblas/0.3.17_intel21	libs/libpsml/1.1.10_intel21	libs/
libs/fftw/3.3.10_gcc112_ompi412 ↳ openblas/0.3.19_gcc112	libs/libpsml/1.1.7_gcc82	libs/
libs/fftw/3.3.10_intel21 ↳ pgplot/5.2.2_gcc112	libs/libpsml/1.1.7_gcc93	libs/
libs/fftw/3.3.10_intel21_impi21 ↳ refblas/3.10_gcc112	libs/libpsml/1.1.8_gcc112	libs/

(continues on next page)

(continued from previous page)

libs/fftw/3.3.8_intel18 ↳ refblas/3.10_gcc48	libs/libxc/3.0.1_gcc48	libs/
libs/fftw/3.3.8_intel18_thd ↳ refblas/3.8_gcc82	libs/libxc/3.0.1_gcc82	libs/
libs/fftw/3.3.9_gcc48 ↳ scalapack/2.1.0_gcc112_mpic343	libs/libxc/3.0.1_intel18	libs/
libs/fftw/3.3.9_gcc82 ↳ scalapack/2.1.0_gcc112_mvap236	libs/libxc/4.3.4_gcc82	libs/
libs/fftw/3.3.9_gcc93 ↳ scalapack/2.1.0_gcc112_ompi412	libs/libxc/4.3.4_gcc93	libs/
libs/fftw/3.3.9_gcc93_mpic341 ↳ scalapack/2.1.0_gcc82_mvap235	libs/libxc/4.3.4_intel18	libs/
libs/fftw/3.3.9_gcc93_mvap235 ↳ scalapack/2.1.0_gcc93_mpic341	libs/libxc/4.3.4_intel19	libs/
libs/fftw/3.3.9_intel18 ↳ scalapack/2.1.0_gcc93_mvap235	libs/libxc/5.1.7_gcc112	libs/
libs/fftw/3.3.9_intel19 ↳ sparsehash/2.0.3	libs/libxc/5.1.7_intel21	libs/
libs/fftw/3.3.9_intel21 ↳ suitesparse/5.4.0_gcc82	libs/magma/2.5.1_gcc48	libs/
libs/fftw/3.3.9_intel21_impi21 ↳ superlu/5.2.1_gcc93	libs/netcdf/4.7.4_gcc48	libs/
libs/gmp/6.2.0 ↳ swig/4.0.1_gcc82	libs/netcdf/4.7.4_gcc82	libs/
libs/gridxc/0.9.6_intel21 ↳ wannier90/3.1.0_gcc112	libs/netcdf/4.7.4_gcc93	libs/
libs/hdf5/1.10.5_gcc48_ompi31 ↳ wcslib/7.7_gcc112	libs/netcdf/4.7.4_gcc93_mpic341	libs/
libs/hdf5/1.10.5_gcc82_ompi31 ↳ xmlf90/1.5.4_gcc112	libs/netcdf/4.7.4_gcc93_mvap235	libs/
libs/hdf5/1.10.5_intel18 ↳ xmlf90/1.5.4_gcc48	libs/netcdf/4.7.4_intel19	libs/
libs/hdf5/1.10.5_intel18_impi18 ↳ xmlf90/1.5.4_gcc82	libs/netcdf/4.8.0_gcc82_ompi405	libs/
libs/hdf5/1.10.5_intel19 ↳ xmlf90/1.5.4_gcc93	libs/netcdf/4.8.1_gcc112	libs/
libs/hdf5/1.10.5_intel19_impi19 ↳ xmlf90/1.5.5_intel21	libs/netcdf/4.8.1_intel21	libs/
libs/hdf5/1.10.7_gcc48 ↳ yaml/0.2.2_gcc82	libs/netcdf/4.8.1_intel21_impi21	libs/
libs/hdf5/1.10.7_gcc82 ↳ zeromq/4.3.1_gcc82	libs/netcdf/fortran-4.5.3_gcc112	libs/
libs/hdf5/1.10.7_gcc93	libs/netcdf/fortran-4.5.3_gcc48	

4.4.3 Parallel Libraries

----- /shared/modulefiles/tier0 -----		
parallel/cuda/10.0 →6_gcc48	parallel/hwloc/2.5.0_gcc93	parallel/openmpi/3.1.
parallel/cuda/10.1 →6_gcc82	parallel/hwloc/2.6.0_gcc112	parallel/openmpi/3.1.
parallel/cuda/10.2 →6_gcc93	parallel/impi/2017	parallel/openmpi/3.1.
parallel/cuda/11.0 →6_intel21	parallel/mpich/3.4.1_gcc93	parallel/openmpi/3.1.
parallel/cuda/11.1 →5_gcc82	parallel/mpich/3.4.1_intel19	parallel/openmpi/4.0.
parallel/cuda/11.2 →5_gcc93	parallel/mpich/3.4.3_gcc112	parallel/openmpi/4.0.
parallel/cuda/11.3 →7_intel21	parallel/mvapich2/2.3.5_gcc82	parallel/openmpi/4.0.
parallel/cuda/11.4 →1_gcc48	parallel/mvapich2/2.3.5_gcc93	parallel/openmpi/4.1.
parallel/cuda/11.5 →1_gcc82	parallel/mvapich2/2.3.6_gcc112	parallel/openmpi/4.1.
parallel/cuda/11.6 →1_gcc93	parallel/openmpi/2.1.2_gcc48	parallel/openmpi/4.1.
parallel/hwloc/1.10.1_gcc48 →2_gcc112	parallel/openmpi/2.1.6_gcc48	parallel/openmpi/4.1.
parallel/hwloc/1.10.1_gcc82 →2_intel21	parallel/openmpi/2.1.6_gcc82	parallel/openmpi/4.1.
parallel/hwloc/1.10.1_intel18 →gcc48	parallel/openmpi/2.1.6_intel18	parallel/ucx/1.10.1_
parallel/hwloc/1.11.13_gcc82 →gcc93	parallel/openmpi/3.1.4_gcc48	parallel/ucx/1.10.1_
parallel/hwloc/2.0.3_gcc82 →gcc112	parallel/openmpi/3.1.4_gcc82	parallel/ucx/1.11.2_
parallel/hwloc/2.0.3_intel18 →gcc82	parallel/openmpi/3.1.4_gcc93	parallel/ucx/1.9.0_
parallel/hwloc/2.2.0_gcc82 →gcc93	parallel/openmpi/3.1.4_intel18	parallel/ucx/1.9.0_
parallel/hwloc/2.2.0_gcc93	parallel/openmpi/3.1.4_intel18_tm	
parallel/hwloc/2.5.0_gcc48	parallel/openmpi/3.1.6_gcc112	

4.4.4 Atomistic Packages

----- /shared/modulefiles/tier2 -----		
atomistic/abinit/8.10.2_intel18	atomistic/namd/2.13_CPU	
atomistic/abinit/8.10.3_gcc82	atomistic/namd/2.13_CUDA	
atomistic/abinit/8.10.3_gcc82_mpiio	atomistic/namd/NAMD_2.14_mpi	
atomistic/abinit/8.10.3_intel18	atomistic/namd/NAMD_2.14-mpi-smp	
atomistic/abinit/9.4.2_intel19_mpic341	atomistic/namd/NAMD_2.14_ofi	
atomistic/abinit/9.4.2_intel21	atomistic/namd/NAMD_2.14-ofi-smp	

(continues on next page)

(continued from previous page)

atomistic/abinit/9.4.2_intel21_impi21	atomistic/namd/NAMD_Git-2021-12-14-mpi
atomistic/abinit/9.6.2_gcc93_ompi41	atomistic/namd/NAMD_Git-2021-12-14-mpi-smp
atomistic/abinit/9.6.2_intel21_impi21	atomistic/namd/NAMD_Git-2021-12-14-ofi
atomistic/amber/18_cuda	atomistic/namd/NAMD_Git-2021-12-14-ofi-smp
atomistic/amber/18_mpi	atomistic/octopus/10.4_gcc93_mplic341
atomistic/amber/18_openmp	atomistic/octopus/11.0_intel21
atomistic/amber/20_gcc93_cuda113	atomistic/octopus/11.3_intel21_impi21
atomistic/amber/20_gcc93_mplic341	atomistic/octopus/11.3_intel21_impi21_
└─prereq	
atomistic/amber/20_gcc93_mplic341_cuda113	atomistic/orca/5.0.1_shared
atomistic/amber/20_gcc93_openmp	atomistic/orca/5.0.2_shared
atomistic/amber/20_gcc93_serial	atomistic/orca/5.0.2_static
atomistic/amber/20_intel21_impi21	atomistic/orca/5.0.3_shared
atomistic/amber/20_intel21_openmp	atomistic/orca/5.0.3_static
atomistic/amber/20_intel21_serial	atomistic/plumed/2.5.3_gcc82
atomistic/castep/19.11-mpi_intel18	atomistic/plumed/2.7.1_gcc93_mplic341
atomistic/elk/5.2.14_intel18	atomistic/siesta/4.0.2_intel18
atomistic/elk/7.2.42_intel21	atomistic/siesta/4.0.2_intel19
atomistic/elk/8.3.15_intel21_impi21	atomistic/siesta/4.1.5_intel21
atomistic/espresso/6.4_intel18_seq	atomistic/siesta/4.1.5_intel21_impi21
atomistic/espresso/6.4_intel18_thd	atomistic/siesta/4.1.5_psml_intel21
atomistic/espresso/6.8_intel21	atomistic/vasp/5.4.4_intel18_seq
atomistic/espresso/6.8_intel21_impi21	atomistic/vasp/5.4.4_intel18_thd
atomistic/gaussian/g16	atomistic/vasp/5.4.4_intel19_seq
atomistic/gaussian/g16_rev1	atomistic/vasp/5.4.4_intel19_thd
atomistic/gromacs/2016.6_gcc48_cuda10	atomistic/vasp/6.2.1_intel19
atomistic/gromacs/2021.3_double_intel21	atomistic/vasp/6.2.1_intel21
atomistic/gromacs/2021.3_intel21	atomistic/vasp/6.2.1_intel21_impi21
atomistic/gromacs/2021.4_gcc93_ompi411	atomistic/vasp/6.2.1_intel21_impi21_prereq
atomistic/gromacs/2021.4_gcc93_ompi411_cuda11	atomistic/wannier90/1.2_gcc93
atomistic/gromacs/5.1.5_cuda10	atomistic/wannier90/2.1.0_gcc82
atomistic/gromacs/5.1.5_double_intel18	atomistic/wannier90/2.1.0_gcc93
atomistic/gromacs/5.1.5_intel18	atomistic/wannier90/3.1.0_gcc82
atomistic/gromacs/5.1.5_intel19	atomistic/wannier90/3.1.0_gcc93
atomistic/lammps/2020.10.29_gcc93_mplic341	atomistic/wannier90/3.1.0_intel19
atomistic/mesmer/6.0_gcc93	atomistic/wannier90/3.1.0_intel21
atomistic/mesmer/6.0_intel19	

4.4.5 Modules for Intel Compilers, Math and Parallel Libraries

----- /shared/modulefiles/intel -----		
advisor/2021.2.0	dnnl/latest	intel_ipp_ia32/latest
advisor/2021.4.0	dnnl-cpu-gomp/2021.2.0	intel_ipp_intel64/2021.2.0
advisor/latest	dnnl-cpu-gomp/2021.4.0	intel_ipp_intel64/2021.4.0
ccl/2021.2.0	dnnl-cpu-gomp/latest	intel_ipp_intel64/latest
ccl/2021.4.0	dnnl-cpu-iomp/2021.2.0	itac/2021.4.0
ccl/latest	dnnl-cpu-iomp/2021.4.0	itac/latest
clk/2021.4.0	dnnl-cpu-iomp/latest	mkl/2021.2.0
clk/latest	dnnl-cpu-tbb/2021.2.0	mkl/2021.4.0
compiler/2021.2.0	dnnl-cpu-tbb/2021.4.0	mkl/latest

(continues on next page)

(continued from previous page)

compiler/ 2021.4.0	dnnl-cpu-tbb/latest	mkl32/ 2021.2.0
compiler/latest	dpct/ 2021.2.0	mkl32/ 2021.4.0
compiler32/ 2021.2.0	dpct/ 2021.4.0	mkl32/latest
compiler32/ 2021.4.0	dpct/latest	mpi/ 2021.2.0
compiler32/latest	dpl/ 2021.2.0	mpi/ 2021.4.0
compiler-rt/ 2021.2.0	dpl/ 2021.5.0	mpi/latest
compiler-rt/ 2021.4.0	dpl/latest	oclpga/ 2021.2.0
compiler-rt/latest	icc/ 2021.4.0	oclpga/ 2021.4.0
compiler-rt32/ 2021.2.0	icc/latest	oclpga/latest
compiler-rt32/ 2021.4.0	icc32/ 2021.4.0	tbb/ 2021.2.0
compiler-rt32/latest	icc32/latest	tbb/ 2021.4.0
dal/ 2021.2.0	init_opencl/ 2021.2.0	tbb/latest
dal/ 2021.4.0	init_opencl/ 2021.4.0	tbb32/ 2021.2.0
dal/latest	init_opencl/latest	tbb32/ 2021.4.0
debugger/ 10.1.1	inspector/ 2021.4.0	tbb32/latest
debugger/ 10.2.4	inspector/latest	vpl/ 2021.2.2
debugger/latest	intel_ipppc_ia32/ 2021.4.0	vpl/ 2021.6.0
dev-utilities/ 2021.2.0	intel_ipppc_ia32/latest	vpl/latest
dev-utilities/ 2021.4.0	intel_ipppc_intel64/ 2021.4.0	vtune/ 2021.2.0
dev-utilities/latest	intel_ipppc_intel64/latest	vtune/ 2021.7.1
dnnl/ 2021.2.0	intel_ipp_ia32/ 2021.2.0	vtune/latest
dnnl/ 2021.4.0	intel_ipp_ia32/ 2021.4.0	

4.5 Jupyter Notebooks

A Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The “notebook” term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the “.ipynb” extension.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through “Download As” in the web interface, via the nbconvert library or “jupyter nbconvert” command line interface in a shell.

4.5.1 Open On-Demand

Enter on the webpage: <https://ondemand-tf.hpc.wvu.edu>

In the menu “Interactive Apps” select “Jupyter Notebook”. There are several versions of Python that can be selected. On the same form you can select to use the traditional “Jupyter Notebooks” or the new “JupyterLab” interface.

4.5.2 Install the Julia kernel

Load the module for julia:

```
$> module load lang/gcc/11.2.0 lang/julia
```

Enter in julia:

```
$> julia
      _ _(_)_   | Documentation: https://docs.julialang.org
      (--)| (--)   | Type "?" for help, "]?" for Pkg help.
      - -  -| |- __ -| | Version 1.7.1 (2021-12-22)
      || || || || /_` | |
      || ||_| | || | C| | |
      _/ | \_`_-|_-|_\_`_-| |
      |__/
julia>
```

Execute the following commands to install the package IJulia, the Jupyter kernel for the julia language:

```
julia> using Pkg
julia> Pkg.add("IJulia")
```

After installing the IJulia package, the next time you enter in Open-On Demand and create a new interactive job. Select the checkbox under “Additional Jupyter kernels (Julia, R)” to ensure that the modules for julia are loaded.

4.5.3 Alternative languages on Jupyter Notebooks

Python is the natural language on Jupyter Notebooks. There are alternative languages that can be used. Two languages are R and Julia. Select “Additional Jupyter kernels (Julia, R)” in the formulary to create a new interactive job.

4.6 HDF5: Hierarchical Data Format

Hierarchical Data Format version (HDF5) is a file format designed to store and organize large amounts of data, in particular numerical data.

Originally developed at the National Center for Supercomputing Applications. During the 90s, NASA investigated 15 different file formats for use in the Earth Observing System (EOS) project. After a two-year review process, HDF was selected as the standard data and information system. The format and standard implementations are now supported by The HDF Group, a non-profit corporation. HDF is supported by many commercial and non-commercial software platforms and programming languages. HDF5, differs significantly from its predecessor, the HDF4 formal both in design and API. Despite of HDF4 being still supported we will concentrate on this section to HDF5 as it should be the natural choice for any new development.

In terms of programming languages, HDF5 is officially supported in the three major languages for scientific computing: Fortran, C and C++. The same is true for high level languages such as Python, R and Julia. Python supports HDF5 via *h5py* and via PyTables. The popular package for data analysis, pandas, can import from and export to HDF5 via *PyTables*. There are two packages for R that offers support, the *rhd5* and *hdf5r* packages. In Julia, the package HDF5 is available.

The plan of this simple tutorial is to create a HDF5 file from a Numpy array, using Python and *h5py* and read the file and extract a particular row using a small code in C.

The first part is very simple, with *h5py*, the code below will create a numpy array 100x7 of random numbers. The array will be stored in a HDF5 file:

```
#!/usr/bin/env python3

import h5py
import numpy as np
data=np.random.rand(100,7)
with h5py.File("testfile.h5", "w") as f:
    dset = f.create_dataset("dset", data=data)
f.close()
```

After executing this code in a Python interpreter or writing a script from it, you end up with a file called `testfile.h5` which is a file in HDF5 file format that contains an array of numbers with 100 rows and 7 columns. Now we will demonstrate how to write a C code that is able to read the file.

4.7 XWindow

4.7.1 Introduction to X

To view a graphical user interface (GUI) from one of HPC systems you will need to have a X Windows System (often shortened to just ‘X’) installed on your PC. UNIX based operating systems (Linux and Mac OS X) come with an X Window terminal called X11. For Windows based machines, you will need to install one.

Once you have a functioning X Server running on your PC, you will need to tunnel your graphics program from the HPC system to your PC.

Note: You may execute lite (i.e. not resource intensive) GUI programs on the login nodes of the HPC systems. However, these should be programs cannot be resource intensive. If you need to run programs such as MATLAB and you plan on doing computation through the MATLAB GUI, you will need to submit your job to the scheduler.

4.7.2 Launching X (GUI Programs)

Linux/OS X Instructions

With Linux or OS X you can simply run the command below from your terminal window to enable X11 forward through SSH.

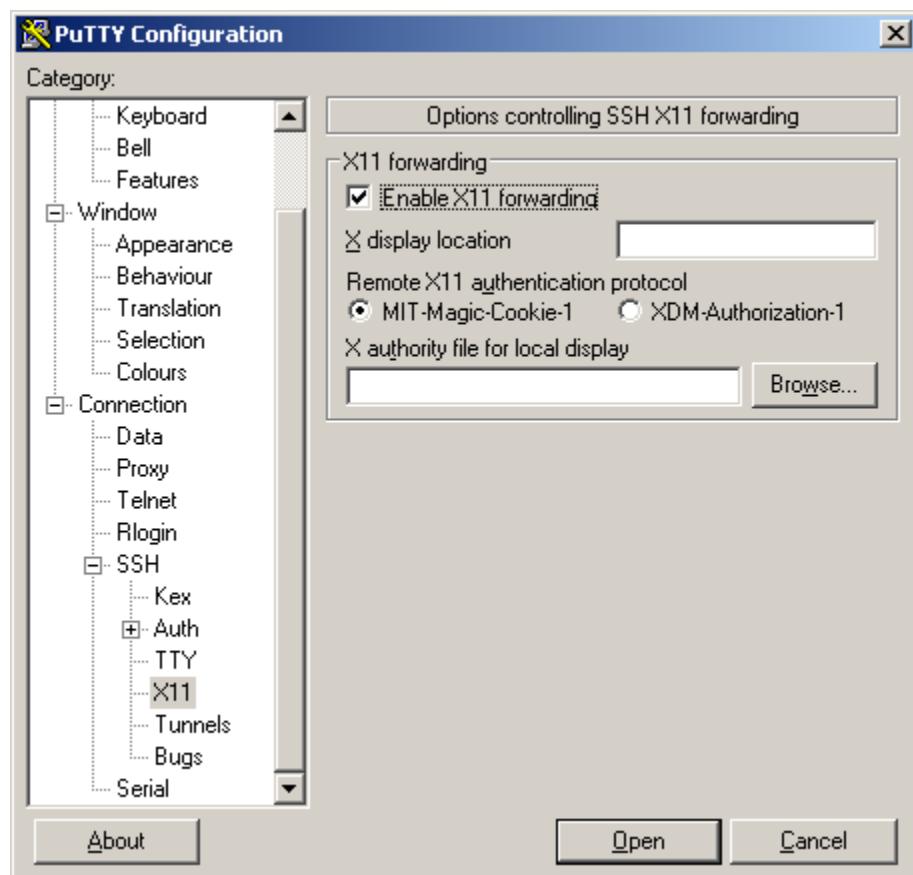
```
$> ssh -XC username@hostname
```

To verify your X setup, you can run the command “*xclock*” from the terminal. If you have successfully tunneled your X traffic, you should see a small clock show up on your screen which you may now close. If you are not seeing a clock, please open a [HelpDesk Ticket](#) for further assistance.

Windows Instructions

Windows does not have a built in X11 Server so one will need to be installed. Below are instructions for [Xming](#) which is compatible with Putty but you maybe also which to look into [MobaXterm](#).

1. Download Xming from here <https://sourceforge.net/projects/xming/files/Xming/6.9.0.31/Xming-6-9-0-31-setup.exe/download>.
2. Install and follow the on-screen instructions. If you do not already have Putty installed you will want to select “Full Installation” when you get to the “Select Components” screen.
3. At the end of the installation you can keep “Launch Xming” selected or launch Xming from your start menu.
4. When Xming is running, you will have a icon in your Windows systray that looks like an “X”.
5. Follow the [Putty Instructions](#) to setup connectivity but before you click “Open” be sure to set “Enable X11 forwarding” under the Category Option section. This will be under Connection, SSH, and X11. Example:



6. You should now be connected to an HPC system with X Forward enabled.
7. To verify your X setup, you can run the command “xclock” from the terminal. If you have successfully tunneled your X traffic, you should see a small clock show up on your screen which you may now close. If you are not seeing a clock, please open a HelpDesk Ticket for further assistance.

4.7.3 Running X (GUI Programs) on Computing Nodes

GUI intensive programs must be ran on a compute node. To accomplish this, one will need to follow the applicable above steps to first establish a X session to an HPC system. Once this has been accomplished you will need to run the following command which will launch an interactive job and give you terminal access on a compute node. Note: The following command is just an example and you will still need to set the appropriate pbs flags that meets the task's needs.

```
$> qsub -q standby -X -I
```

4.8 Compile Source Code

SCIENTIFIC PROGRAMMING

Scientific Programming is done in a variety of programming languages. From open languages to proprietary ones. From low level languages, closer to the machine to high level languages that are closer to human language. A variety of language serve an equal diverse research areas and levels of expertise.

From all those languages we have chosen a limited set of languages and describe for them which are the ways under which those languages can be accessed on our clusters. The purpose of this chapter is not to teach any of the languages that will be presented, learning a programming language is not possible on a single chapter even less to explain the details of a number of different languages some of them very different from each other.

Programming languages can be classified for how close they are to the machine when expressing operations. With low level languages you have to declare the type of each variable and operations such as conditionals, loops and functions (routines or methods) are the building blocks for those languages. The three low level languages more used in scientific computing are Fortran, C and C++ [Fortran](#), [C](#) and [C++](#). First section in this chapter we will devote to these three languages. The different compilers available for them and how to compile basic source code written in those languages.

The next section is dedicated to Python [Python Language](#). Python is a high level language that is often used in scientific computing due to its expressiveness and the large set of scientific libraries that have been written for it. We will show the various Python implementations available on our clusters and the use of IPython and Jupyter as effective tools for interactive scientific computing.

The second high level language is R [R Language](#). R is a commonly used language for scientific computing involving statistical operations. Many packages are available for it thanks to the Comprehensive R Archive Network (CRAN), a repository of thousands of R packages. We also cover RStudio, a user friendly interface for interactive computing in R as well as Jupyter that also offers a kernel for interacting with the language.

The next section is dedicated to Julia [Julia Language](#). Julia is a modern language with a syntax that is clean and modern but with the ability to type variables and offering some modern features that make this language comparable to low level languages in terms of performance. We will demonstrate some of the nice features of the language and the use of Jupyter with a Julia kernel for interactive scientific computing.

After our 2 triads of languages. Fortran, C and C++ for the low level and Python, R and Julia for the high level. We will include a couple of sections describing two other languages used in scientific computing.

The section dedicated to MATLAB [MATLAB](#). MATLAB is a commercial product offering a high level language with an impressive variety of subpackages for most areas in science and engineering. WVU has a license for this package on our clusters.

The next section is about Perl [Perl Language](#). Perl is a popular programming language still in use specially in bioinformatics due to its particular ability to deal with patterns. There is a large collection of packages for Perl. The Comprehensive Perl Archive Network (CPAN) is a repository of software modules and accompanying documentation of codes written in the Perl programming language. We will shows how to install perl packages from CPAN as user.

Parallel Programming is at the core of High Performance Computing. On an HPC cluster, compute nodes have tens of cores, large amounts of RAM and sometimes include accelerators such as GPUs. Those nodes are interconnected using expensive networking. The main reason for all this is to take advantage of parallelism at various level to speed

up scientific calculations that are too complex for a normal desktop machine. There are several strategies for parallel computing and the final 3 sections explore those models.

The first of those three chapters is dedicated to OpenMP *Parallel Programming: OpenMP*. Computers today are multicore, in striking difference with desktop computers in the 20th century that only have a single core on a single CPU chip. Being multicore means that the CPU offers several computing cores, complete processing units as capable of running processes as a single CPU. The cores on the machine are able to see all the memory (RAM) of the computer. Multicore machines allow for a special kind of parallelism called “shared memory multiprocessing” and OpenMP is a prominent example of this kind of parallelism.

The next section is about Message Passing Interface (MPI) *Parallel Programming: MPI*. A more general parallelism is achieved when several machines each one of them with their own memory work in concert on a given problem. That model of parallelism is called “Distributed Parallel Computing”. The machines at some point need to communicate information to the other partners and that is done by passing messages between the different compute nodes. MPI is the dominant interface for this kind of parallel programming. Several implementations of MPI exist and we will describe how to compile and run calculations that uses MPI.

The last two sections are dedicated to GPU computing *Parallel Programming: OpenACC* and *Parallel Programming: CUDA*. Modern HPC cluster now rely more and more on accelerators. Accelerators are electronic devices specially dedicated for certain kinds of tasks for which they excel compared to a normal CPU. General Purpose Graphics Processing Units (GPGPUs) is the usage of GPUs for numerical calculation, also called accelerators, GPU-based high performance computers are starting to play a significant role in HPC.

5.1 Fortran, C and C++

Scientific computing traces its roots to the very origin of computers. The first electro-mechanical computers originally were created to compute large numerical calculations much faster than humans can do. With the evolution of computers came new ways of expressing algorithms in ways closer to human languages but still very close to the internal operation of electronic machines. With the new electronic computers in the 1950’s, programmers used assembly language (or assembler language), low-level programming language with a close correspondence between the instructions in the language and the architecture’s machine code instructions. Programming in assembly is very time consuming and prone to error. New programming languages evolved to facilitate the translation of algorithms into code, avoiding almost entirely the need for writing in assembler. Today, three dominant low level languages for scientific computing are Fortran, C and C++.

Fortran is the oldest of those three, and many algorithms that are fundamental for most problems in science were written in it. Fortran 77 was for many years the standard de facto for scientific computing and have evolve over the years with Fortran 90, 95, 2003 and 2008.

C is a general purpose language and used in most of the basic software written in Unix environments. The language itself is not targeted to write scientific programs so it less expressive for vectors and matrices compared to Fortran.

C++ is the third language in the list. C++ is in many ways an extension of C with high element constructs such as classes. C++ is widely used in desktop applications, games and also scientific applications.

Fortran, C and C++ shared a number of attributes. These languages need a compiler that translates the source code into machine code. Parallelization interfaces such as MPI are written with explicit support for those three languages and compiler suites such as GCC, Intel and NVIDIA HPC include those languages by default.

5.1.1 Accessing C, C++ and Fortran Compilers

In an HPC cluster it is customary to include compilers for Fortran, C and C++ by default. On Linux machines the usual compilers are from the GNU Compiler Collection (GCC). Apart from GCC, there are other vendors of compilers for these three languages. Each vendor brings its own advantages and particular options to optimize the resulting binaries, support for CPU extensions, support for language extensions and support for various standards of the languages that are standardized and reviewed over time. The Intel Compiler Suite and the NVIDIA HPC SDK are two examples of alternative compilers that are available in many HPC clusters. Each compiler for the three languages has its own name. The table below shows the names of the compilers for each of the three languages from each vendor.

Table 1: Name of compiler commands from different vendors

Language	GNU Compiler Collection (GCC)	Intel Compiler Suite	NVIDIA HPC SDK
Fortran	gfortran	ifort	nvfortran
C	gcc	icc	nvc
C++	g++	icpc	nvc++

The compiler installed by default with the Operating System is usually too old. That is due to the tendency of having more stable software packages on production systems and that is the case on HPC clusters too. The versions of those installed compilers are old for the purpose of many scientific packages and do not support recent CPU extensions and language extensions such as OpenMP or OpenACC. On Thorny Flat the default version of the compilers is 4.8, one way of identifying the version of gcc compilers is running the command:

```
$> gfortran -v
Using built-in specs.
COLLECT_GCC=gfortran
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/
--info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-
--shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-
--cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-
--build-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,
--fortran,ada,go,lto --enable-plugin --enable-initfini-array --disable-libgcj --with-
--isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/isl-install --
--with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-
--install --enable-gnu-indirect-function --with-tune=generic --with-arch_32=x86-64 --
--build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
```

It is usually better to use environment modules to access newer versions of GCC or compilers from other vendors such as Intel or NVIDIA. The table below shows the modules that can be used on Thorny Flat to load the modules for compilers from different vendors and from several versions

Vendor	Modules	Advantages
GCC	lang/gcc/8.2.0 lang/gcc/9.3.0 lang/gcc/11.2.0	Standard <i>de-facto</i> on Linux/UNIX Most codes compile correctly with GCC Relative good support for OpenMP and AVX extensions
Intel	lang/intel/2018 lang/intel/2019 compiler/2021.2.0 compiler/2021.4.0	Compilers that optimize particularly well on Intel Hardware. Good support for OpenMP but not OpenACC
NVIDIA	lang/nvidia/nvhpc/20.7 lang/nvidia/nvhpc/21.3 lang/nvidia/nvhpc/21.11	Good support for OpenACC in addition to OpenMP. Informative parallelization information

5.1.2 Compiling codes

```
#include <stdio.h>
#include <stdlib.h>

void sieve(int *, int);

int main(int argc, char *argv[])
{
    /* Declaring variables */
    int *array, n;

    /* Reading command line arguments */
    if (argc != 2) /* argc should be 2 for correct execution */
    {
        /* We print argv[0] assuming it is the program name */
        printf("usage: %s max_number\n", argv[0]);
    }
    else
    {
        n=atoi(argv[1]);
        array =(int *)malloc(sizeof(int));
        sieve(array,n);
    }
}
```

(continues on next page)

(continued from previous page)

```

return 0;
}

void sieve(int *a, int n)
{
    int i=0, j=0;

    for(i=2; i<=n; i++) {
        a[i] = 1;
    }

    for(i=2; i<=n; i++) {
        printf("\ni:%d", i);
        if(a[i] == 1) {
            for(j=i; (i*j)<=n; j++) {
                printf ("\nj:%d", j);
                printf("\\nBefore a[%d*%d]: %d", i, j, a[i*j]);
                a[(i*j)] = 0;
                printf("\\nAfter a[%d*%d]: %d", i, j, a[i*j]);
            }
        }
    }

    printf("\\nPrimes numbers from 1 to %d are : ", n);
    for(i=2; i<=n; i++) {
        if(a[i] == 1)
            printf("%d, ", i);
    }
    printf("\\n\\n");
}

```

```

module str2int_mod
contains

    elemental subroutine str2int(str,int,stat)
        implicit none
        ! Arguments
        character(len=*),intent(in) :: str
        integer,intent(out)          :: int
        integer,intent(out)          :: stat

        read(str,*,iostat=stat) int
    end subroutine str2int

end module

program sieve

    use str2int_mod
    implicit none

    integer :: i, stat, i_max=0

```

(continues on next page)

(continued from previous page)

```

logical, dimension(:), allocatable :: is_prime
character(len=32) :: arg

i = 0
do
    call get_command_argument(i, arg)
    if (len_trim(arg) == 0) exit

    i = i+1
    if ( i == 2 ) then
        call str2int(trim(arg), i_max, stat)
        write(*,*) "Sieve for prime numbers up to", i_max
    end if

end do

if (i_max .lt. 1) then
    write (*,*) "Enter the maximum number to search for primes"
    call exit(1)
end if

allocate(is_prime(i_max))

is_prime = .true.
is_prime (1) = .false.
do i = 2, int (sqrt (real (i_max)))
    if (is_prime (i)) is_prime (i * i : i_max : i) = .false.
end do
do i = 1, i_max
    if (is_prime (i)) write (*, '(i0, 1x)', advance = 'no') i
end do
write (*, *)

end program sieve

```

5.2 Python Language

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Due to Python's extensive third-party libraries such as NumPy, SciPy and many others, make Python a programming language for scientific computing. Python servers as an important tool for several areas in science and engineering and packages for plotting offer publication-ready plotting capabilities.

Among external libraries *NumPy* provides the foundation for numerical operation, in particular operations related to vectors and matrices. *SciPy* extends NumPy to include special functions, numerical integration and other scientific general purpose operations. *Matplotlib* is the basic library for plotting in 2D and some functions for 3D plots.

Several specialized packages have been written such as Biopython and Astropy providing domain-specific functionality in Astronomy and Biology. Python is commonly used in artificial intelligence projects with the help of libraries like

TensorFlow, Keras and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

Both clusters, Spruce and Thorny offer a variety of Python implementations and versions, including the reference Python implementation written in C and called CPython, the MKL optimized Intel Python Distribution and PyPy, a fast, compliant alternative implementation of the Python thanks to its Just-in-Time compiler.

5.2.1 Accessing the Python interpreter

The Python interpreter is the basic command that allows you to interact with Python interactively and is the same command used for scripting. There are two versions of Python. Python 2.x is now deprecated but still used in some cases. Python 3.x is the current version and has been around 2008 more than a decade now. You should always prefer Python 3.x over Python 2.x for scientific applications. From January 1, 2020, the 2.x branch of the Python programming language is no longer supported by the Python Software Foundation. Many packages have been abandoning compatibility with Python 2.x and as such there is no much reason to continue using it for scientific applications.

There are four ways of accessing the different Python installations on our clusters: Using a Python version installed by default from the OS repositories, using an *environment module*, creating a *conda* environment or using a *singularity container* that provides Python inside. We will explore those options in the contexts of our two clusters, Spruce Knob and Thorny Flat.

Spruce Knob

The Operating System on Spruce comes with Python 2.6.6 preinstalled:

```
$ python --version
Python 2.6.6
```

No scientific packages are installed along with this version of Python making it unsuitable for most scientific purposes.

On Spruce there is a variety of environment modules for Python. These modules include a good variety of scientific computing packages preinstalled for the corresponding version of Python. The modules available on Spruce are:

```
lang/python/cpython_2.7.15_gcc82
lang/python/cpython_3.6.9_gcc82
lang/python/cpython_3.7.4_gcc82
lang/python/intelpython_2.7.16
lang/python/intelpython_3.6.3
lang/python/intelpython_3.6.9
lang/python/pypy2.7-7.1.1-portable
lang/python/pypy3.6-7.1.1-portable
```

The modules with suffix _gcc82 were compiled with GCC 8.2.0 so the module lang/gcc/8.2.0 must be preloaded to activate the corresponding Python module. For example to load lang/python/cpython_3.7.4_gcc82 you should execute the following command to activate Python:

```
module load lang/gcc/8.2.0 lang/python/cpython_3.7.4_gcc82
```

There are basically three implementations of Python to choose. The versions with prefix `cpython_` are compilations of the reference implementation written in C and Python from <https://www.python.org/downloads> also called *CPython*.

The modules with prefix `intelpython_` are the Intel Distribution for Python. They Intel-optimized versions of the reference CPython using MKL and some other optimized libraries for use in Intel processors. In particular Intel Python includes MKL optimized versions of Numpy, Scipy and Scikit-Learn.

The modules with prefix `pypy` offers an alternative implementation of the Python programming language. PyPy often runs faster than CPython, because PyPy is a just-in-time compiler, while CPython uses a more traditional approach. Most Python code runs well on PyPy, except for code that depends on CPython extensions, which either does not work or incurs some overhead when run in PyPy.

Thorny Flat

The Operating System on Thorny Flat comes with Python 2.7.5 preinstalled:

```
$ python --version
Python 2.7.5
```

No scientific packages are installed along with this version of Python making it unsuitable for most scientific purposes.

On Thorny there are several environment modules for Python. These modules include a good variety of scientific computing packages preinstalled for the corresponding version of Python. The modules available on Thorny Flat are:

```
lang/python/cpython_3.10.5_gcc112
lang/python/cpython_3.10.5_gcc93
lang/python/cpython_3.8.13_gcc112
lang/python/cpython_3.8.13_gcc93
lang/python/cpython_3.9.13_gcc112
lang/python/cpython_3.9.13_gcc93
lang/python/intelpython2_2019.5
lang/python/intelpython_2.7.16
lang/python/intelpython_3.9
lang/python/pypy2.7-v7.3.9-linux64
lang/python/pypy3.9-v7.3.9-linux64
```

5.2.2 Packages installed with CPython modules

In the particular case of CPython modules a number of scientific packages were included. The following table shows the list and version of the packages included on the CPython modules.

Package	Version
appdirs	1.4.4
argon2-cffi	20.1.0
asv	0.4.2
async-generator	1.10
atomicwrites	1.4.0
attrs	20.3.0
backcall	0.2.0
bleach	3.3.0
cached-property	1.5.2
cffi	1.14.5
cloudpickle	1.6.0
cycler	0.10.0
Cython	0.29.22
dask	2021.3.0
decorator	4.4.2
defusedxml	0.7.1

continues on next page

Table 2 – continued from previous page

Package	Version
distlib	0.3.1
entrypoints	0.3
filelock	3.0.12
h5py	3.1.0
imageio	2.9.0
importlib-metadata	3.7.3
importlib-resources	5.1.2
iniconfig	1.1.1
ipykernel	5.5.0
ipyparallel	6.3.0
ipython	7.16.1
ipython-genutils	0.2.0
ipywidgets	7.6.3
jedi	0.18.0
Jinja2	2.11.3
joblib	1.0.1
joblib	1.0.1
jsonschema	3.2.0
jupyter	1.0.0
jupyter-client	6.1.12
jupyter-console	6.4.0
jupyter-core	4.7.1
jupyterlab-pygments	0.1.2
jupyterlab-widgets	1.0.0
kiwisolver	1.3.1
MarkupSafe	1.1.1
matplotlib	3.3.4
mistune	0.8.4
more-itertools	8.7.0
mpmath	1.2.1
nbclient	0.5.3
nbconvert	6.0.7
nbformat	5.1.2
nest-asyncio	1.5.1
networkx	2.5
notebook	6.3.0
numpy	1.19.5
packaging	20.9
pandas	1.1.5
pandocfilters	1.4.3
parso	0.8.1
pexpect	4.8.0
pickleshare	0.7.5
Pillow	8.1.2
pip	21.0.1
pluggy	0.13.1
prometheus-client	0.9.0
prompt-toolkit	3.0.18
ptyprocess	0.7.0

continues on next page

Table 2 – continued from previous page

Package	Version
ptyprocess	0.7.0
py	1.10.0
pycparser	2.20
Pygments	2.8.1
pymongo	3.11.3
pyparsing	2.4.7
pyrsistent	0.17.3
pytest	6.2.2
python-dateutil	2.8.1
pytz	2021.1
PyWavelets	1.1.1
PyYAML	5.4.1
pyzmq	22.0.3
qtconsole	5.0.3
QtPy	1.9.0
scikit-image	0.17.2
scikit-learn	0.24.1
scipy	1.5.4
seaborn	0.11.1
Send2Trash	1.5.0
setuptools	54.2.0
six	1.15.0
sympy	1.7.1
terminado	0.9.3
testpath	0.4.4
threadpoolctl	2.1.0
tifffile	2020.9.3
toml	0.10.2
toolz	0.11.1
tornado	6.1
traitlets	4.3.3
typing-extensions	3.7.4.3
virtualenv	20.4.3
wcwidth	0.2.5
webencodings	0.5.1
widgetsnbextension	3.5.1
xlrd	2.0.1
zipp	3.4.1

The modules for Pypy and Intel Python include their own list of preinstalled packages.

Another alternative to get Python is creating a conda environment. Load conda with the command:

```
source /shared/software/conda/etc/profile.d/conda.sh
```

This will activate the command conda and you can create conda environments for the version of Python of your choice. This is particularly useful if you want a very specific version of Python, as new as 3.9.2 or as old as 2.7.13. You can search for all the versions available with:

```
conda search python
```

Or including specific channels with:

```
conda search -c intel python
conda search -c conda-forge python
```

Both *intel* and *conda-forge* are popular channels for general purpose scientific packages.

For example to create a conda environment called *python392* installing insider Python version 3.9.2 execute:

```
$> conda create -n python392 python==3.9.2

Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /users/gufranco/.conda/envs/python392

added / updated specs:
- python==3.9.2
```

The following packages will be downloaded:

package	build	
ca-certificates-2021.1.19	h06a4308_1	118 KB
certifi-2020.12.5	py39h06a4308_0	140 KB
openssl-1.1.1j	h27cf23_0	2.5 MB
pip-21.0.1	py39h06a4308_0	1.8 MB
python-3.9.2	hdb3f193_0	18.2 MB
setuptools-52.0.0	py39h06a4308_0	724 KB
sqlite-3.35.2	hdfb4753_0	983 KB
tzdata-2020f	h52ac0ba_0	113 KB
<hr/>		
	Total:	24.5 MB

The following NEW packages will be INSTALLED:

_libgcc_mutex	pkgs/main/linux-64::_libgcc_mutex-0.1-main
ca-certificates	pkgs/main/linux-64::ca-certificates-2021.1.19-h06a4308_1
certifi	pkgs/main/linux-64::certifi-2020.12.5-py39h06a4308_0
ld_impl_linux-64	pkgs/main/linux-64::ld_impl_linux-64-2.33.1-h53a641e_7
libffi	pkgs/main/linux-64::libffi-3.3-he6710b0_2
libgcc-ng	pkgs/main/linux-64::libgcc-ng-9.1.0-hdf63c60_0
libstdcxx-ng	pkgs/main/linux-64::libstdcxx-ng-9.1.0-hdf63c60_0
ncurses	pkgs/main/linux-64::ncurses-6.2-he6710b0_1
openssl	pkgs/main/linux-64::openssl-1.1.1j-h27cf23_0
pip	pkgs/main/linux-64::pip-21.0.1-py39h06a4308_0
python	pkgs/main/linux-64::python-3.9.2-hdb3f193_0
readline	pkgs/main/linux-64::readline-8.1-h27cf23_0
setuptools	pkgs/main/linux-64::setuptools-52.0.0-py39h06a4308_0
sqlite	pkgs/main/linux-64::sqlite-3.35.2-hdfb4753_0
tk	pkgs/main/linux-64::tk-8.6.10-hbc83047_0

(continues on next page)

(continued from previous page)

```

tzdata          pkgs/main/noarch::tzdata-2020f-h52ac0ba_0
wheel           pkgs/main/noarch::wheel-0.36.2-pyhd3eb1b0_0
xz              pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
zlib            pkgs/main/linux-64::zlib-1.2.11-h7b6447c_3

Proceed ([y]/n)? y

Downloading and Extracting Packages
python-3.9.2      | 18.2 MB    | #####|#####
˓# | 100%
tzdata-2020f     | 113 KB     | #####|#####
˓# | 100%
setuptools-52.0.0 | 724 KB     | #####|#####
˓# | 100%
pip-21.0.1        | 1.8 MB     | #####|#####
˓# | 100%
openssl-1.1.1j   | 2.5 MB     | #####|#####
˓# | 100%
sqlite-3.35.2    | 983 KB     | #####|#####
˓# | 100%
certifi-2020.12.5 | 140 KB     | #####|#####
˓# | 100%
ca-certificates-2021 | 118 KB     | #####|#####
˓# | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#       $ conda activate python392
#
# To deactivate an active environment, use
#
#       $ conda deactivate

```

Finally, all that you have to do is activate the environment with:

```
conda activate python392
```

And deactivate the environment with:

```
conda deactivate
```

5.2.3 Installing python packages with pip

The package *pip* is a popular way of installing python packages. You cannot install packages on system-wide locations, but you still can install them on your \$HOME folder for personal use.

To install Python modules locally (within your user \$HOME directory) is by adding --user as argument for the command `pip install`. It is important to notice that in cases where you have Python 2.x and Python 3.x you need to pay attention to which pip command to use. The command pip usually refers to Python 2.x and there is an equivalent command pip3 for Python 3.x. As we are using Python 3.x, pip3 is the command that we will be using.

There are two ways of using the command pip3. One is calling the command directly:

```
pip3 install --user <package_name>
```

Another is using *pip* indirectly as a module:

```
python3 -m pip install --user <package_name>
```

The --user flag directs python to install the package in a user location rather than a system-wide location where you are not allowed to alter files.

The user location for python packages is \$HOME/.local/lib/pythonX.Y/site-packages, This is generally the preferred method of locally installing new python packages. There is no disadvantage on using a user installation other than the package is only available to you.

5.2.4 Installing python packages with a custom prefix

Another way to install Python modules locally is by using the --target flag:

```
pip3 install --target <dir> <package_name>
```

representing the directory location you want the package installed into. These flags essentially do the same thing by directing Python to install the module in the specified directory. These directories will not be searched by default with Python. Therefore, in order to use these modules in your Python scripts you will have to modify the \$PYTHONPATH environment variable to include the specified directory. Or alternatively, modify sys.path from within your python script (for this method, consult [python documentation](#)).

```
export PYTHONPATH=<dir>
```

5.2.5 Using Python virtualenv

The installing with `pip install --user` or `pip install --target <dir>` those locations are all searched secondary to the system-wide site packages.

This could be an issue if you are trying to install locally a different version of a module already installed system-wide. A way to get around this is by using Python Virtual Environments.

Python virtual environments are used to build completely isolated python workflows. Primarily they are used to solve the need for multiple versions within python modules. Often, you might have the need to use pkgA which needs pkgC version 1.24, but you also need pkgB which needs pkgC version 2.1. If you use setuptools to install the packages (i.e. pip or easy_install), you will create a dependency issue since both versions of pkgC will be installed to the same location.

To resolve this, you can create python virtual environments that all isolation of package dependencies, so you can successfully have different versions of packages installed and tied to separate python interpreters. Setting up python virtual environments is easy, and using them is no different than using python it's self.

Using Virtual Environments with python2

First, load which version of python2 you would like to use as your base python interpreter. For instance, if you want python 2.7.10, then load the 2.7.10 python modulefile. If you want to use the default system python (v. 2.6), then you do not need to load a python modulefile. However, you do need to load the virtualenv modulefile:

```
module load lang/gcc/8.2.0 lang/python/2.7.15_gcc82
```

Then create a virtualenv directory with the ‘virtualenv’ command:

```
virtualenv workflow1
```

You should now have a directory called ‘workflow1’. You can use whatever name you want for the virtualenv, so long as you remember what directory corresponds with what environment. You now need to simply activate the virtualenv:

```
source workflow1/bin/activate
```

Your command prompt will now be pre-empted by (workflow1) to remind you that you have an activate virtualenv. You can now proceed to use python, pip, and easy_install just as you would regularly.

Using Virtual Environments with python3

First, load the python3 modulefile. The python3 modulefile comes with its own virtual environment utility, so you do not need to load the virtualenv modulefile:

```
module load lang/gcc/8.2.0 lang/python/cpython_3.7.4_gcc82
```

Then create a virtualenv directory with the ‘pyvenv’ command:

```
pyvenv workflow1
```

You should now have a directory called ‘workflow1’. You can use whatever name you want for the virtualenv, so long as you remember what directory corresponds with what environment. You now need to simply activate the virtualenv:

```
source workflow1/bin/activate
```

Your command prompt will now be pre-empted by (workflow1) to remind you that you have an activate virtualenv. You can now proceed to use python, pip, and easy_install just as you would regularly.

Activating virtual environments using the C shell

If you are using the shells csh or tcsh, you will not be able to source the ‘activate’ file. Instead, you need to source the activate.csh file.

```
source workflow1/bin/activate.csh
```

Using site-wide system packages

The centrally installed python interpreters (python loaded with modulefiles), have some common scientific packages installed with them by default. To have your virtualenv keep using these packages you do not need to install them in your virtualenv, using the `--system-site-packages` option.

```
virtualenv --system-site-packages
```

or

```
pyvenv --system-site-packages
```

5.3 R Language



R is a programming language and environment for scientific computing with a particular emphasis on statistical analysis and plotting. The R project is supported by the R Foundation and it is released under a GNU license. The R language itself was created to be similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. Originally, R was considered as a different implementation of S, it has since grown into a major environment for statistical computing. There are some important differences with the original S language, but much code written for S runs unaltered under R.

The R source code is written primarily in C and Fortran, with some higher-level pieces of code in R itself. R provides a wide variety of statistical tools such as linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, and many others. R is capable of publication-ready graphics and is highly extensible. The R language is often the vehicle of choice for research in statistical methodology, and R provides an open source alternative for practitioners of statistical tools and academic research in the area.

R is particularly powerful for several reasons. R has an extensive collection of packages, literally thousands of packages available in the the Comprehensive R Archive Network (CRAN). Another strength is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Although R comes only with a command-line interface (CLI), there are several third-party graphical user interfaces (GUIs). One is RStudio, an integrated development environment (IDE), another being Jupyter, with a notebook interface. We will show here how you can access all those GUI interfaces. But before using GUIs it is important first focus on the basic command line interface (CLI).

5.3.1 The R interpreter and CRAN

The R interpreter is the software that allows you to interact with R. There are two commands in R. The command `R` opens a text-based interactive interface, and the command `Rscript` run scripts written in R language

Around R, a ecosystem of external packages have been developed many of them accessible in a repository called CRAN. The Comprehensive R Archive Network (CRAN) is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. On Thorny Flat there is a weekly updated mirror of the CRAN repository located at:

```
/gpfs/pub/CRAN
```

There several ways of accessing the different R on our clusters. The Operating System comes with its own installation of R, usually the version is too old for research purposes. There are also environment modules providing compiled versions of R centrally managed along many other scientific software packages. Another alternative is to use containers, containers are filesystems usually archived on a single file and including all the software needed to run R. Finally, you can create your own conda environments and choose between a variety of R versions. We will explore these options in the contexts of our two clusters: Spruce Knob and Thorny Flat.

Spruce Knob

On Spruce Knob, R is preinstalled from RedHat Enterprise Linux (RHEL) repositories. Spruce runs Red Hat Enterprise Linux Server release 6.10 (Santiago). On this Linux distribution, R is included with version 3.0.2. This version is too old for most scientific applications today. You should not have much reason to use that version.

In terms of environment modules Spruce Knob offers three versions: 3.6.3, 4.0.5 and 4.1.2. You can load any of those modules with the `module` command:

```
$> module load lang/R/3.6.3_gcc103  
$> module load lang/R/4.0.5_gcc103  
$> module load lang/R/4.1.2_gcc103
```

After that, the command `R` and `Rscript` became available on the shell. The third way of accessing the R interpreter is via a Singularity *container*. First, you need to load the singularity environment module:

```
$> module load singularity
```

Singularity images are located at `/shared/software/containers/` and the variable `$SNG_PATH` contains this path. There are two images that include the R interpreter inside:

```
RStudio-desktop-1.2.5042_R-3.4.4.sif  
RStudio-server-1.2.5042_R-3.6.2.sif
```

Getting shell access to inside any of these containers to access R and `Rscript`:

```
$> singularity shell $SNG_PATH/RStudio-server-1.2.5042_R-3.6.2.sif  
$> singularity shell $SNG_PATH/RStudio-desktop-1.2.5042_R-3.4.4.sif
```

Despite including extra software such as RStudio they can be used for their internal R interpreters, especially if you are interested in older versions of R such as 3.4.4.

Thorny Flat

On Thorny Flat the R interpreter is not preinstalled from RHEL packages and the only two options are to use the environment modules or using a singularity image. The environment modules for R available on Thorny are:

```
lang/r/3.6.3_gcc112
lang/r/4.0.5_gcc112
lang/r/4.1.2_gcc112
lang/r/4.2.0_gcc112
```

To load the module, use the command `module load` including `lang/gcc/11.2` before the module to be added as dependency. Execute any of the lines below:

```
$> module load lang/gcc/11.2.0 lang/r/3.6.3_gcc112
$> module load lang/gcc/11.2.0 lang/r/4.0.5_gcc112
$> module load lang/gcc/11.2.0 lang/r/4.1.2_gcc112
$> module load lang/gcc/11.2.0 lang/r/4.2.0_gcc112
```

After that, the command R and Rscript became available on the shell.

Another option for accessing the R interpreter is via a singularity *container*. Several images include the R interpreter, in particular, the two images below offer an alternative to equivalent versions from environment modules:

```
R-3.6.3.sif
R-4.0.5.sif
R-4.1.3.sif
R-4.2.0.sif
```

To get a shell prompt inside any of those images load the singularity module first:

```
$> module load singularity
```

After that, a shell inside the image can be obtained with:

```
$> singularity shell $SNG_PATH3/R-3.6.3.sif
$> singularity shell $SNG_PATH3/R-4.0.5.sif
$> singularity shell $SNG_PATH3/R-4.1.2.sif
$> singularity shell $SNG_PATH3/R-4.2.0.sif
```

After getting a shell inside the *container* the commands R and Rscript became available. A shortcut to access the R interpreter in one single step is with the `run` option, for example:

```
$> singularity run $SNG_PATH3/R-4.2.0.sif
```

Another way of getting R is using a conda environments. You can create your own conda environment and install the version of R that better fits your needs. For example, these are the commands to create a conda environment called R351 with R version 3.5.1 installed from the conda-forge channel.

Load conda module:

```
$> module load conda
```

Create a new conda environment, here we are calling it R351 but the name is arbitrary:

```
$> conda create -n R351
```

Activate the new conda environment:

```
$> conda activate R351
```

And install the version of conda of your preference. To see the list of conda versions execute:

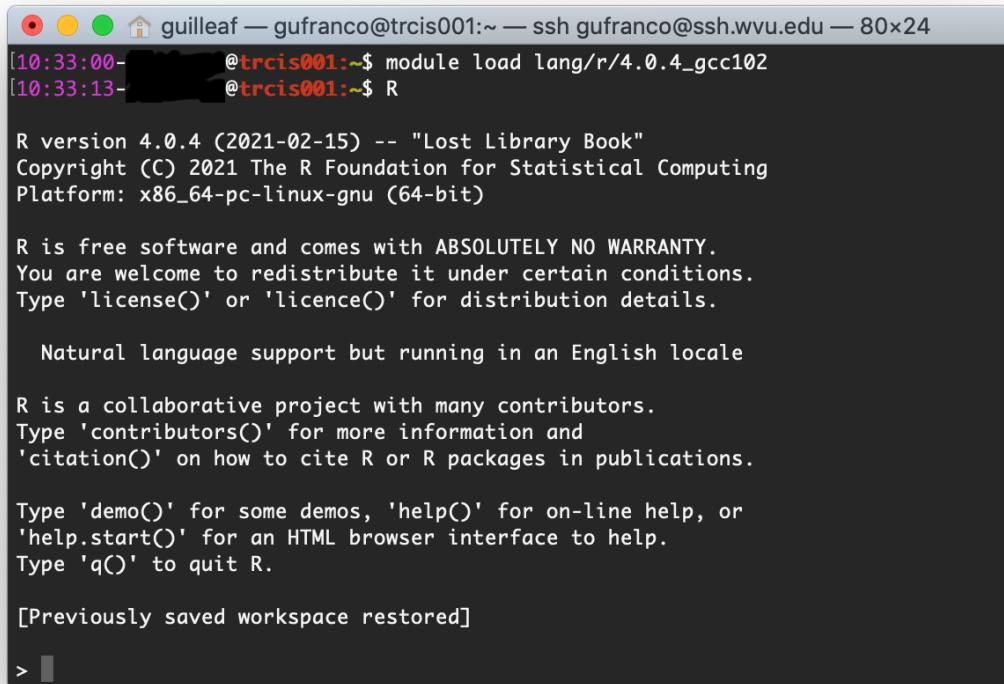
```
$> conda search -c conda-forge r
```

And to install a particular version execute:

```
$> conda install -c conda-forge r=3.5.1
```

5.3.2 Using R from the Terminal

Regardless of how you access R, the R package includes two executables, R and Rscript. The command R is used to start a text-based interactive session, on the terminal execute the command R.



The screenshot shows a terminal window titled "guilleaf — gufranco@trcis001:~ — ssh gufranco@ssh.wvu.edu — 80x24". The window displays the R command line interface. It starts with the command "module load lang/r/4.0.4_gcc102", followed by the R command "R". The R startup message follows, including the version (4.0.4), copyright information, platform (x86_64-pc-linux-gnu (64-bit)), and license details. It also mentions natural language support and collaborative contributors. At the bottom, it says "[Previously saved workspace restored]" and has a prompt ">".

From there you can start typing R commands and getting output interactively:

```
> 2 + 2  
[1] 4
```

Another way of working in R, and more suited for an HPC environment, is using R scripts. To do so, first, write your R commands as script in a file. It is customary to add the .R extension to the file. Consider for example a very minimal script, lets create a file hello.R with the content below:

```
#!/usr/bin/env Rscript

sayHello <- function(){
  print('hello')
}

sayHello()
```

The first line of the file must be `#!/usr/bin/env Rscript`. This line instructs the shell that this is a R script, meaning that all commands below the first line must be processed by the R internal interpreter.

To make this script like any other executable, we need to change permissions to the file:

```
$> chmod +x hello.R
```

The script is ready for execution. As this is such a small program you can run it directly on the head node:

```
$> ./hello.R
[1] "hello"
```

However, for large executions, you should always create a submission script and never run numerical intensive operations on the head node. Create a file `runjob.pbs` with this content:

```
#!/bin/bash

#PBS -N MY_R_SCRIPT
#PBS -q standby
#PBS -j oe
#PBS -l nodes=1:ppn=1

cd $PBS_O_WORKDIR

module purge
module load lang/r/4.0.4_gcc102

./hello.R
```

The name of the job is declared with `#PBS -N MY_R_SCRIPT`. The queue is declared with `#PBS -q standby`, for executions that run in less than 4 hours `standby` is the preferable choice. The line `#PBS -j oe` joins the error with the output in a single file. The line `#PBS -l nodes=1:ppn=1` is asking to run on a single node and using a single core for the execution. In most cases, R runs serially but we will discuss some parallel executions below. The line `cd $PBS_O_WORKDIR` changes the directory to the place where the command `qsub` was executed. The line `module purge` cleans the environment of any modules and `module load lang/r/4.0.4_gcc102` will load the module for executing R. The name of the module is for Thorny Flat, on Spruce you need to use the modules mentioned above for that cluster.

Finally `./hello.R` executes the script and the output will be written in a file like `MY_R_SCRIPT.o<JobID>` where JobID is a number that identifies the job in the queue system. The contents of the file are shown below:

```
$> cat MY_R_SCRIPT.o4714619
[1] "hello"
```

5.3.3 Installing R Packages as a normal user

The system-wide location for installed packages is not writable for normal users. That does not prevent you from installing packages for your usage. Packages installed as user will be installed at \$HOME/R/x86_64-pc-linux-gnu-library/<R_VERSION> and will be accessible to you as any system-wide package. For the versions that we currently have on our clusters, <R_VERSION> could be 3.6 or 4.0.

Packages can only be installed from the head node both on Thorny and Spruce. Compute nodes do not have internet access, so they are not capable to access CRAN from the internet to download any package.

There are two main ways to install R packages from CRAN. You can install packages from inside the interactive R interpreter. Execute the command *R* and from there install the package with:

```
install.packages('<Package Name>', repos="file:///gpfs21/pub/CRAN")
```

Another alternative is using the command *Rscript*, this time from the normal shell:

```
$> Rscript -e 'install.packages("<PACKAGE>", repos="file:///gpfs21/pub/CRAN")'
```

With those commands, you will automatically download the package from CRAN and eventually all dependencies needed for that package. The declaration `repos="file:///gpfs21/pub/CRAN"` is optional. Now declaring a URL for downloads and you will get a list of known mirrors where you can download the package. The local CRAN mirror on Thorny Flat is preferable to install packages from a remote internet server such as:

```
install.packages('<Package Name>', repos="https://cran.rstudio.com")
```

With any of those methods, R will automatically detect that you do not have permission to write in the system-wide R library folder and will prompt if you would like to install in a local directory from within your home directory. This folder will also be checked automatically when you run R for packages, allowing you to use anything you already installed.

For example, lets assume that you want to install the package *dplyr*:

```
> install.packages("dplyr")
Installing package into '/gpfs/home/<username>/R/x86_64-pc-linux-gnu-library/3.6'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
Secure CRAN mirrors

  1: Ø-Cloud [https]           2: Algeria [https]
  ...
  ...
 57: USA (CA 1) [https]       58: USA (IA) [https]
 59: USA (KS) [https]         60: USA (MI 1) [https]
 61: USA (MI 2) [https]       62: USA (OR) [https]
 63: USA (TN) [https]         64: USA (TX 1) [https]
 65: Uruguay [https]          66: (other mirrors)

Selection: 63

also installing the dependencies 'backports', 'ellipsis', \
'digest', 'zealot', 'utf8', 'vctrs', 'cli', 'crayon', \
'fansi', 'pillar', 'purrr', 'assertthat', 'glue', \
'magrittr', 'pkgconfig', 'R6', 'Rcpp', 'rlang', \
'tibble', 'tidyselect', 'BH', 'plogr'
```

After some downloads and compilations the code will be installed on your local folder \$HOME/R/x86_64-pc-linux-gnu-library/3.6

5.3.4 Installing R Packages in non-default directories

To use locally installed packages, before you execute R you just need to modify the R_LIBS environment variable to tell R where to search for local packages:

```
$> export R_LIBS=<dir>
```

Using install.packages()

To use `install.packages` from the R prompt, before you start R you need to modify the `R_LIBS` environment variable:

```
$> export R_LIBS=<dir>
```

Then inside the R prompt, you execute any of the procedures above such as `install.packages()` inside the R interactive shell.

Manual installation with R CMD INSTALL

There is a very manual way of installing packages in R using the command `R CMD INSTALL`. There are several restrictions to use this command: First, this option only works if you have already downloaded a copy of the package from CRAN, any of its mirrors, or an external site that provides a tar package. Second, as a normal user, you must specify the path with the `-l` flag, otherwise, the command will refuse to work as you have no write access to the system-wide location. The command must be used like this:

```
$> R CMD INSTALL -l <dir> <package_name>
```

The folder `<dir>` is the location where the package will be installed and the folder must be created in advance. `<package_name>` is the name of the package and you must ensure that all dependencies are already installed for the package, meaning that you will have to install several other packages before you can install the package that you want.

Consider for example installing the package `parallelly`, this package has no dependencies beyond those that already came with the R installation. The sources are available on CRAN and can be downloaded from one of its many mirrors. You can first download the package with:

```
$> wget https://cran.r-project.org/src/contrib/parallelly_1.24.0.tar.gz
```

Once the package is downloaded, ensure that the folder where you want to install the software exists, for example, if you want to install packages at `~/R/local_packages`, create the folder with:

```
$> mkdir -p ~/R/local_packages
```

Finally, install the packages with:

```
$> R CMD INSTALL -l ~/R/local_packages parallelly_1.24.0.tar.gz
```

The package is now installed but it can only become visible if you set up the `R_LIBS` environment for it:

```
$> export R_LIBS=~/R/local_packages
```

Finally load the package with:

```
> library("parallelly")
```

You need to be careful not to mix R packages created with different major versions of R, those are usually incompatible if you create the package with one version and try to use it with a different version. That is the reason why more automatic installers create separate folders for different R versions.

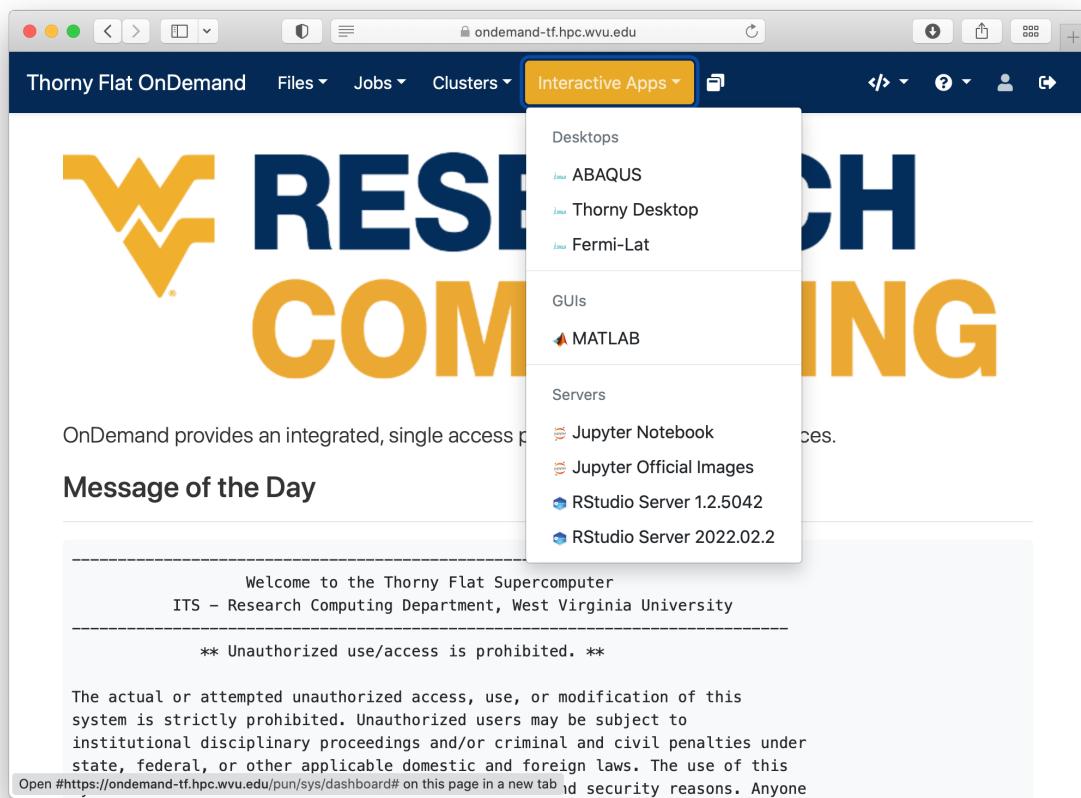
5.3.5 Graphical Interfaces: RStudio

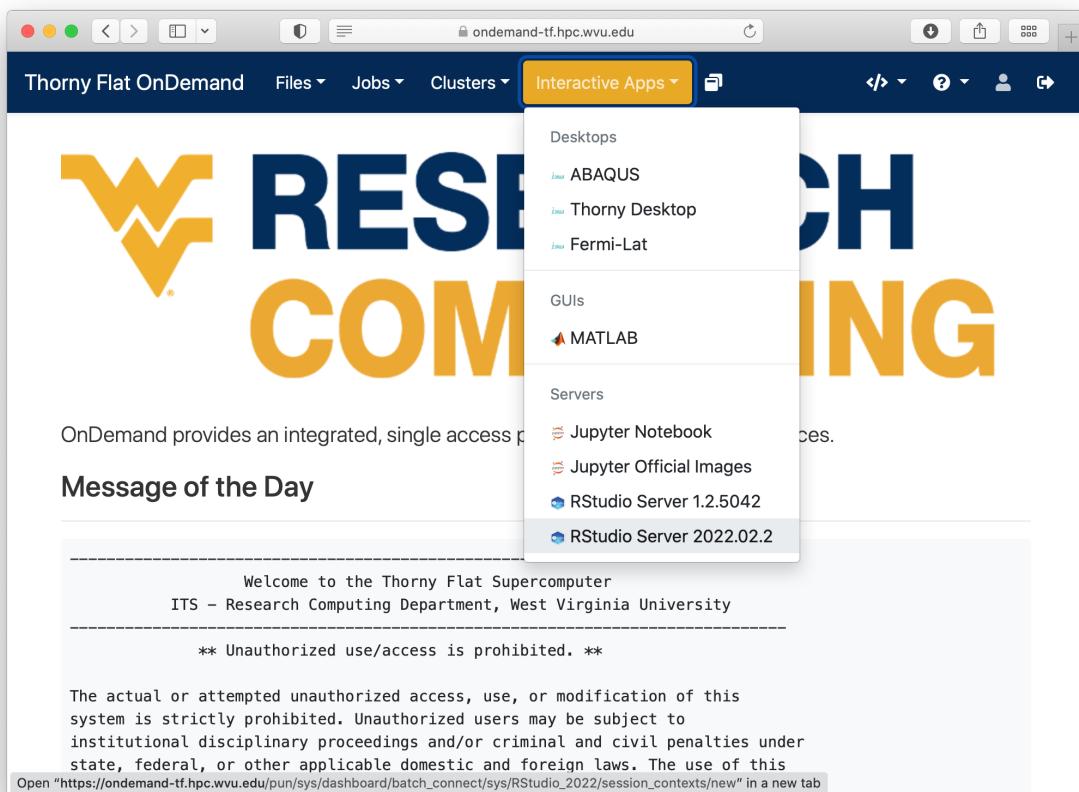
RStudio is a GUI application that allows you to interact with R from a window environment on your computer or a web interface on an HPC cluster. We will demonstrate how to access RStudio on the cluster.

We offer access to RStudio via Open On-Demand. Open On-Demand is web service that offers interactive access to the cluster over internet. A job is created transparently to the user and RStudio can run from a compute node for a certain amount of time.

The first step is to open a web browser on your local computer and go to <https://ondemand-tf.hpc.wvu.edu>

Once you enter your credentials, you land on the Open On-Demand Dashboard



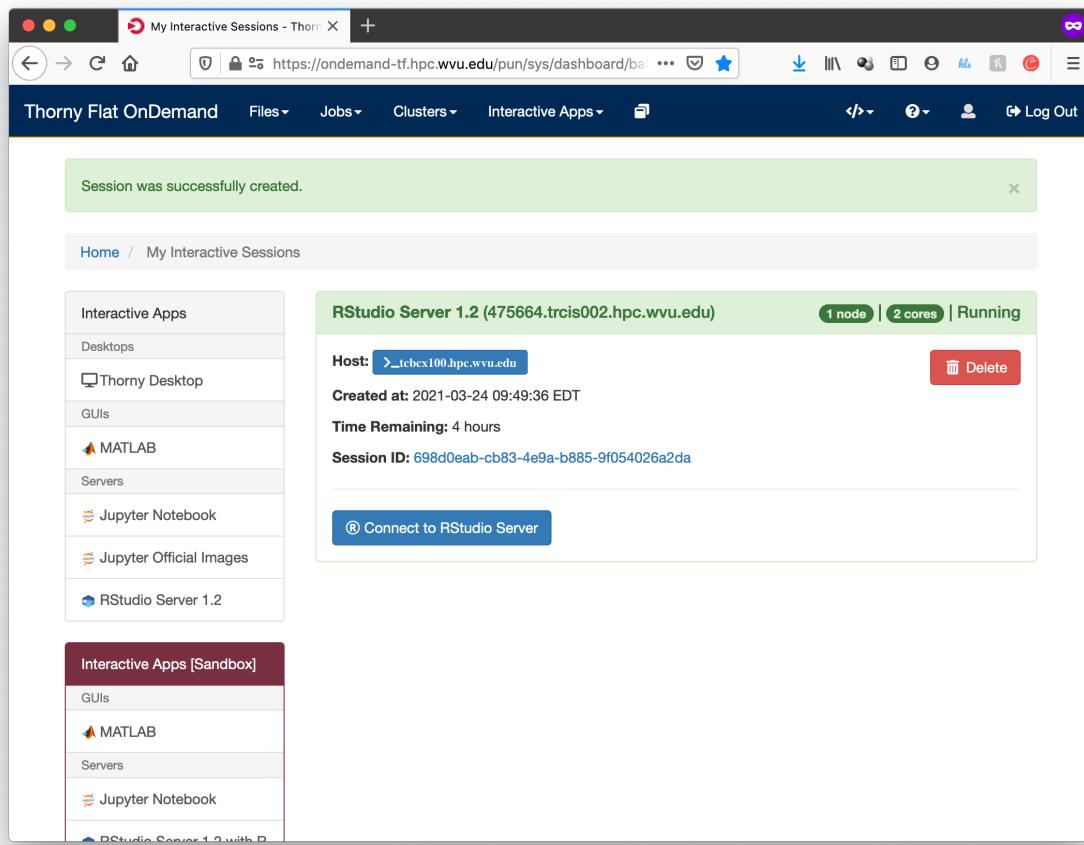


Go to *Interactive Apps*. There are two versions of RStudio, the older 1.2.5042 and the newer 2022.02.2. They offer different versions of R. Preferentially, select *RStudio Server 2022.02.2*. A form will be there for filling all the details needed to create a RStudio job.

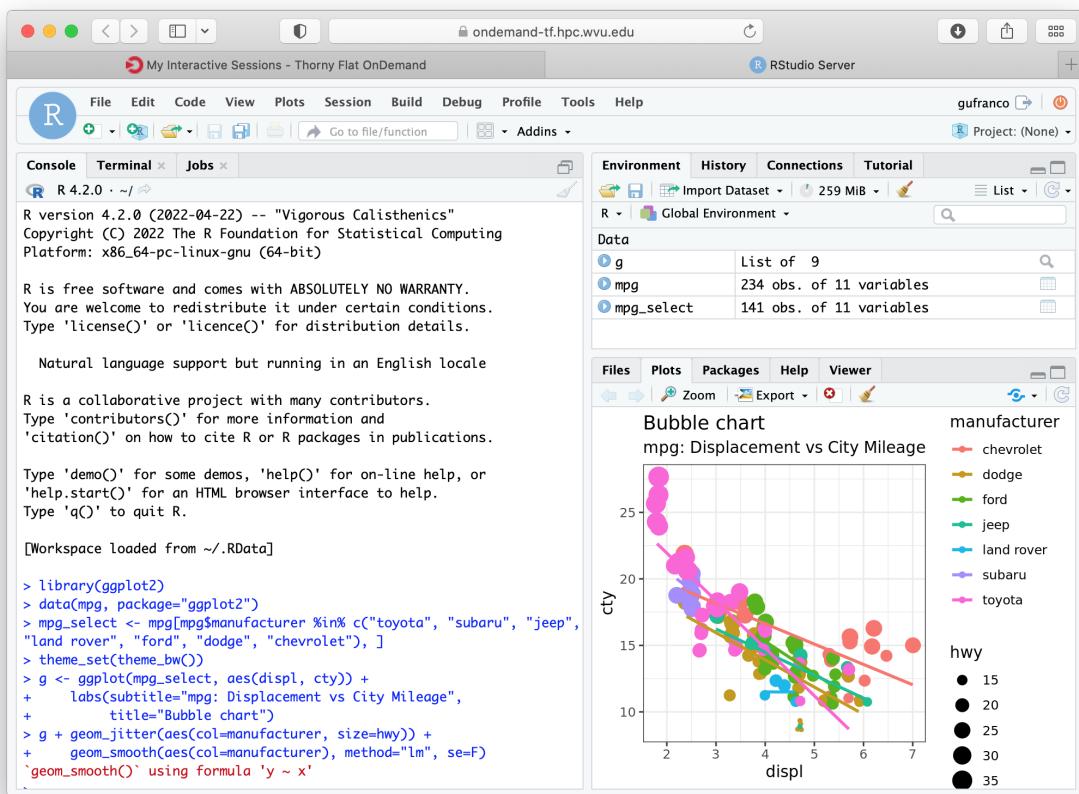
The screenshot shows a web browser window for the Thony Flat OnDemand system. The URL is `ondemand-tf.hpc.wvu.edu`. The main content area displays the title "RStudio Server 2022.02.2" and a sub-section titled "R version and Ubuntu/Linux filesystem". It includes a dropdown menu set to "RStudio Server:2022.02.2-485 R:4.2.0 Ubuntu:focal-2". Below this, it states "Singularity Image you want to use to access RStudio Server. RStudio Server version is 2022.02.2 R interpreter versions are 4.1.3 and 4.2.0 Ubuntu/Linux filesystem version is 20.04 (focal)". A section titled "Queue for Torque/Moab" contains a dropdown menu set to "standby". It explains that this selection affects the number of hours requested for execution. The bottom of the page has a note about "Number of hours for execution (walltime)".

The form ask for the Singularity image that you want to use for launching the server. Several images are presently available built from different versions of Ubuntu as the base filesystem. A good default value could be RStudio 2022.02.2 with R 4.2.0 with Ubuntu focal as base filesystem. The next question is about the queue to run the job. You can get 4 hours using the *standby* queue and that is a good choice in most cases. The next field is the number of hours to run the job. Using *standby*, the max walltime is 4 hours, so we use that. Finally the number of cores or processes per node. This is the number of cores reserved on a single compute node, the max number of cores on machines of Thorny is 40 but without special routines (for example Rparallel or sparklyr), R will be running serially, which means that no matter how many cores you ask only one core will be used. We are setting 4 cores in this case. Finally, press *Launch* to submit the job request.

The next screen will show the job being created for you. After a few seconds, you will see a button *Connect to RStudio Server*. Press the button.



RStudio will be running, and showing the IDE interface, from this interface you can execute R commands, see the variables declared and the plots being generated. The interface is very powerful for advanced users of R.



5.3.6 Graphical Interfaces: Jupyter

An alternative GUI for R is Jupyter. Jupyter presents a Notebook interface, where there are boxes with code and boxes with text, including titles, subtitles, and even equations.

Go to *Interactive Apps* and select *Jupyter Official Images*. A form will be there for filling all the details needed to create a RStudio job.

The screenshot shows a web browser window for the Thony Flat OnDemand system at ondemand-tf.hpc.wvu.edu. The main title bar says "Jupyter Official Images - Thony Flat OnDemand". The top navigation bar includes links for "Thony Flat OnDemand", "Files", "Jobs", "Clusters", "Interactive Apps", and "RStudio Server". Below the navigation is a breadcrumb trail: "Home / My Interactive Sessions / Jupyter Official Images".

Interactive Apps

- Desktops
 - ABAQUS
 - Thony Desktop
 - Fermi-Lat
- GUIs
 - MATLAB
- Servers
 - Jupyter Notebook
 - Jupyter Official Images
 - RStudio Server 12.5042

Jupyter Official Images

This app will launch a Jupyter Notebook server on a compute node. Jupyter notebooks running from Singularity images created from the official jupyter channel on docker.

Name of Singularity Image

Jupyter Notebook Python, Scala, R, Spark Stack (all-sp)

Singularity Image you want to use to access jupyter.

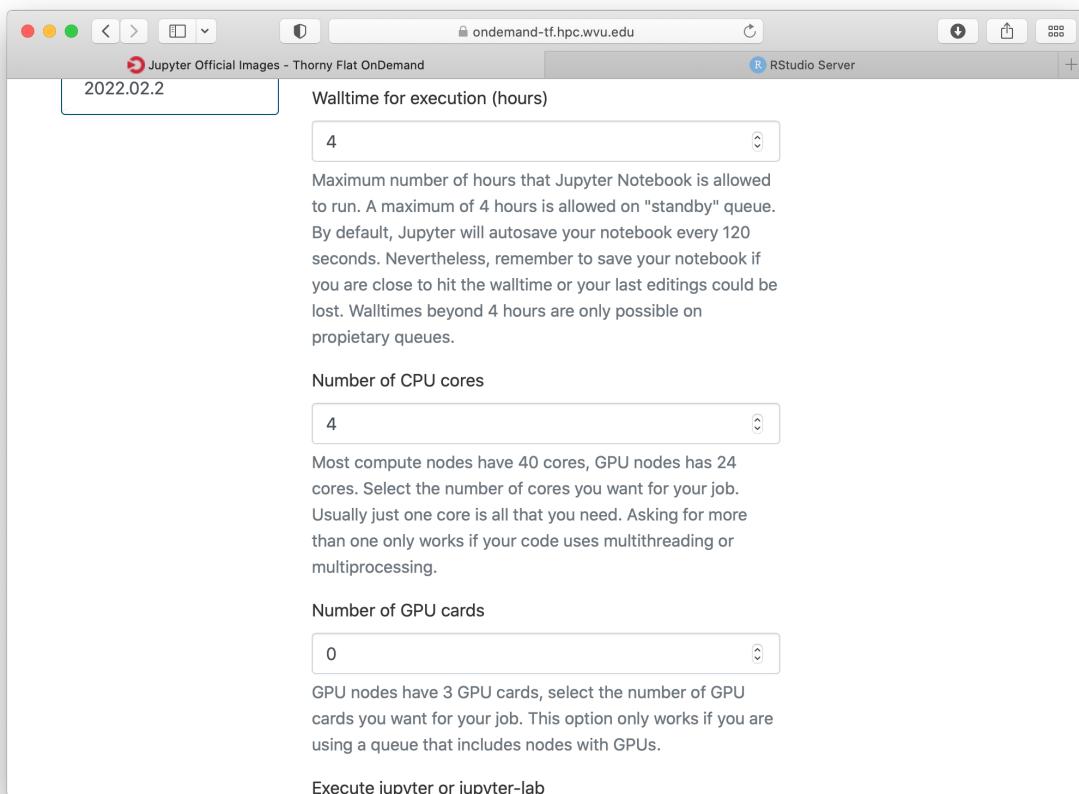
Queue for Torque/Moab

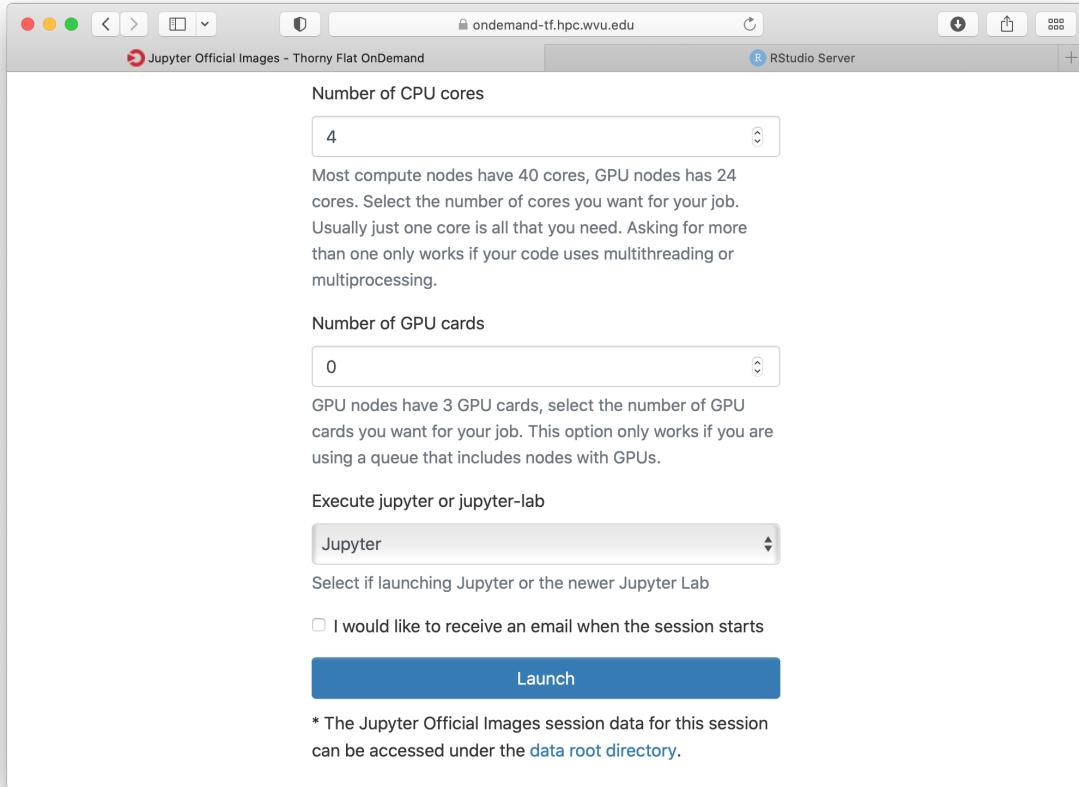
standby

Select queue for interactive job, community as well as proprietary queues can be selected. Community queues are these:

- debug: max 1 hour, CPU only
- standby: max 4 hours, CPU only
- comm_gpu_inter: max 4 hours, CPU + GPU

The selection of queue affects the maximum walltime and the





Select one of the images that include R, one of them is for example *r_notebook*. The next question is about the queue to run the job. You can get 4 hours using the *standby* queue and that is a good choice in most cases. The next field is the number of hours to run the job. Using *standby* the max walltime is 4 hours, so we use that. Finally the number of cores. This is the number of cores to use on the compute node, the max number of cores on one machine on Thorny is 40 but without special routines, R will be running serially, which means that no matter how many cores you ask only one core will be used. We are setting 4 cores in this case. Finally, press *Launch* to submit the job request.

The next screen will show the job being created for you. After a few seconds, you will see a button *Connect to Jupyter*. Press the button.

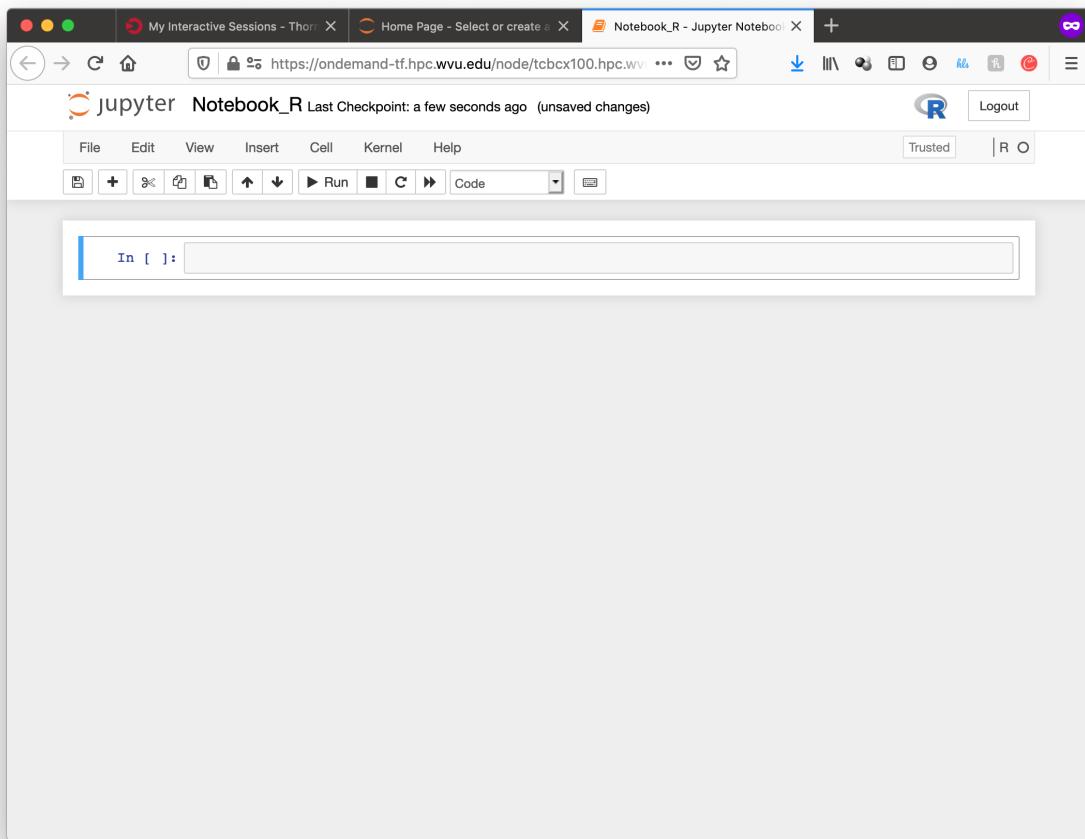
The screenshot shows the Thorney Flat OnDemand interface. At the top, there's a navigation bar with links for Home, Files, Jobs, Clusters, Interactive Apps, and Log Out. A message box at the top left says "Session was successfully created." Below the navigation, there are two main session cards.

Jupyter Official Images (475665.trcis002.hpc.wvu.edu)
 Host: >_tcbcx100.hpc.wvu.edu | 1 node | 4 cores | Running
 Created at: 2021-03-24 09:51:19 EDT
 Time Remaining: 4 hours
 Session ID: 1c187982-a9fe-44ba-9f92-528d946cbe4a
 Connect to Jupyter

RStudio Server 1.2 (475664.trcis002.hpc.wvu.edu)
 Host: >_tcbcx100.hpc.wvu.edu | 1 node | 2 cores | Running
 Created at: 2021-03-24 09:49:36 EDT
 Time Remaining: 3 hours and 58 minutes
 Session ID: 698d0eab-cb83-4e9a-b885-9f054026a2da
 Connect to RStudio Server

On the left, there's a sidebar with categories: Interactive Apps (Desktops, Thorney Desktop, GUIs, MATLAB, Servers, Jupyter Notebook, Jupyter Official Images, RStudio Server 1.2), and Interactive Apps [Sandbox] (GUIs, MATLAB, Servers, Jupyter Notebook, RStudio Server 1.2).

Jupyter will be running, and showing the Jupyter File Manager interface, from the manager you select *New >> R* and another tab on your browser will open with the Jupyter Notebook running a R kernel. From this interface you can execute R commands, commands can be entered in the boxes and executed with SHIFT ENTER. Boxes can also serve for text including titles, subtitles and text with equations.



5.3.7 Parallel Computing with R

By default R only uses one core, which is a big limitation when dealing with large amounts of data or complex calculations. However, R includes by default the *parallel* package which is the foundational tool for parallel computing in R. There are several other third-party tools for parallelism but we will be focusing on the tools available on any R installation.

The *parallel* package came installed by default on all modern versions of R. The package needs to be loaded before use with:

```
> library("parallel")
```

Once the library is loaded several routines became available that are able to take advantage of share memory systems, on machines such as Thorny, most compute nodes have 40 cores, however it is up to you to indicate the number of cores that you want to use and that should match the amount of resources that you requested via your submission script or your interactive job.

The first function that we will use from *parallel* is `detectCores()`. The function will return the total number of cores on the compute node, not the total number of cores that you requested for the job:

```
> detectCores()  
[1] 40
```

The *parallel* package offers a number of functions that replace the serial versions of *Apply* functions such as `lapply`, `sapply` and `apply`. The parallel replacements being `parLapply`, `parSapply` and `parApply`. To use any of those functions we need first to create a *cluster* and internal structure for the *parallel* package that instructs the number of cores that can be used when parallelizing the apply functions.

The function `makeCluster()` will create a cluster. If you have created a job asking to use 4 cores, you can create a *cluster* with 4 cores like this:

```
> cl <- makeCluster(4)
> cl
socket cluster with 4 nodes on host 'localhost'
```

Now with the *cluster* `cl` we can execute any of the alternative versions of `apply` functions:

```
> ans <- parLapply(cl, 1:10000000, function(x) sqrt(x^(1/3)) + sqrt(x^(1/2)))
```

This function will apply `function(x)` as it was defined for the first 10 million positive integers returning a vector of equal number of floats stored in `ans`.

To compute this function R will be using a cluster build from 4 execution instances and will take a fraction of the time compared with the serial version of `lapply`. We can see some timings here:

```
> system.time(parLapply(cl, 1:10000000, function(x) sqrt(x^(1/3)) + sqrt(x^(1/2))))
user  system elapsed
1.787  0.093  3.721
> system.time(lapply(1:10000000, function(x) sqrt(x^(1/3)) + sqrt(x^(1/2))))
user  system elapsed
6.484  0.023  6.522
```

We got almost an execution that was almost 4 times shorter than for the serial case. It is never the case that we get a perfect scaling, in more practical cases you should always expect that some time is lost in communication and extra overhead in orchestrating the parallel execution. The parallel versions of *apply* functions are an easy way of taking advantage of multicore architectures present of modern computers.

At the end of execution is always important to close the cluster with the `stopCluster` function:

```
> stopCluster(cl)
```

You should always stop the cluster to free resources. In more practical scenarios you should always ensure that `stopCluster` is called no matter if the function fails for some reason. For doing that you can use the `on.exit()` function:

```
simulate <- function(ncores) {
  cl = makeCluster(ncores)
  on.exit(stopCluster(cl))

  # The core of the function belows

}
```

5.3.8 Using Fortran inside R

R itself is not a fast language. Speed is not the designed goal of the language. The goal is to create a tool that makes data analysis and statistics easier for people who uses those tools. In many cases practitioners of statistical tools are not programmers, they are not planning to create the fastest code, they just need to have the analysis done and move on. R is slow compared to other programming languages, and sometimes users who are not very attentive to good programming best practices usually make things worse, however, for many purposes, R is fast enough.

People with knowledge on programming could improve the performance of R by executing numerical intensive operations on a different language retaining the advantages of working for most part on a high level language like R.

We will demonstrate how to achieve that with a code written in Fortran. Fortran is a low level language. A low level language uses an explicit compiler, a program that converts the source code into code that the computer can execute.

Consider the following Fortran code (facto.f90):

```
!
! simple subroutine to compute factorial
!
subroutine facto(n, answer)
    integer n, answer, i

    answer = 1
    do i = 2,n
        answer = answer * i
    end do
end subroutine
```

This routine in fortran computes the factorial of a number *n* and return the result in the variable `answer`.

We can compile this file with the GNU fortran compiler called **gfortran**:

```
$> gfortran -c facto.f90
```

The result of the compilation is a file called **facto.o**. The next step is to create a shared library from this function:

```
$> gfortran -shared -o facto.so facto.o
```

The resulting file is **facto.so** that can be imported from R and used (R Terminal):

```
> dyn.load("facto.so")
> .Fortran("facto",n=as.integer(40),answer=as.numeric(1.0))
$n
[1] 40

$answer
[1] 8.159153e+47
```

5.4 Julia Language

The Julia programming language is a flexible dynamic language, appropriate for scientific and numerical computing. Julia combines features of imperative, functional, and object-oriented programming. Julia provides ease and expressiveness for high-level numerical computing, similar to other popular languages such as R, MATLAB, and Python, but also supports general programming. Julia is built upon the lineage of mathematical programming languages, but also borrows much from popular dynamic languages, including Lisp, Perl, Python, Lua, and Ruby. Using the correct syntax, the performance of Julia programs can be comparable to traditional statically-typed languages like C or Fortran.

5.4.1 Loading Environment Modules

Julia is installed and accessible via software modules in terminal/shell sessions. To use Julia, load the following modules depending on the cluster you are working on.

Spruce Knob

The latest version of Julia on Spruce Knob is 1.7.2 Load the software module with the command:

```
$> module load lang/julia/1.7.2
```

There is also an equivalent version where dependencies are not preloaded. Load the module as:

```
$> module load lang/gcc/10.3.0 lang/julia/1.7.2_prereq
```

Thorny Flat

On Thorny Flat we have 2 versions of Julia 1.6.5 which is a Long Term Support (LTS) version and 1.7.1. Load any of those versions with the command:

```
$> module load lang/gcc/11.2.0 lang/julia/1.6.5_gcc112
$> module load lang/gcc/11.2.0 lang/julia/1.7.1_gcc112
```

5.4.2 Using Julia

Once the module is loaded, execute the command:

```
$> julia
```

It will open the Julia Interactive Shell:

```

      _      _ _(_)_      | Documentation: https://docs.julialang.org
      (--)    | (--) (--)    |
      - -   -| \|_ -__ -   | Type "?" for help, "]\?" for Pkg help.
      | | | | | | \|/_` | |
      | | \|_| | | | (--) | |
      / \|\_\_|_|_|_| \_\_| |
      \|_|/          |

julia>
```

From the interactive shell, start executing Julia commands. Learn about Julia from the [Official Documentation](#)

Consider a small example on how Julia code looks like:

```
# Function to Compute the volume of a sphere
function volume_sphere(r)
# julia allows Unicode names (in UTF-8 encoding)
# so either "pi" or the symbol \pi can be used
return 4/3*pi*r^3
end

# functions can also be defined as one-liners
quadratic_oneline(a, sqr_term, b) = (-b + sqr_term) / 2a

# calculates x for 0 = a*x^2+b*x+c, arguments types can be defined in function_
# definitions
function quadratic_solver(a::Float64, b::Float64, c::Float64)
# unlike other languages 2a is equivalent to 2*a
# a^2 is used instead of a**2 or pow(a,2)
sqr_term = sqrt(b^2-4a*c)
r1 = quadratic_oneline(a, sqr_term, b)
r2 = quadratic_oneline(a, -sqr_term, b)
# multiple values can be returned from a function using tuples
# if the return keyword is omitted, the last term is returned
r1, r2
end

vol = volume_sphere(3)
# @printf allows number formatting but does not automatically append the \n to_
# statements, see below
using Printf
@printf "Volume of sphere with radius 3 = %0.3f\n\n" vol
#> volume = 113.097

quad1, quad2 = quadratic_solver(2.0, -2.0, -12.0)
println("Solution 1 to quadratic 2x^2 - 2x - 12 is ", quad1)
#> result 1: 3.0
println("Solution 2 to quadratic 2x^2 - 2x - 12 is ", quad2)
#> result 2: -2.0
```

You can execute these lines one by one on the Julia interactive shell. To run a julia script create a file such as *example.jl* and copy there the lines above. Execute the command:

```
$> julia example.jl
```

You should get:

```
Volume of sphere with radius 3 = 113.097

Solution 1 to quadratic 2x^2 - 2x - 12 is 3.0
Solution 2 to quadratic 2x^2 - 2x - 12 is -2.0
```

5.5 MATLAB

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

There are several ways of working with Matlab. As a desktop application you can work interactively using Matlab's graphical user interface (GUI). There you can type commands and see those executed in real time. You can also load scripts and produce plots. The GUI offers a variety of benefits, autocompletion, plot displaying and documentation.

An alternative to the GUI is a terminal-based interactive interface. You can execute the same commands but you work on a terminal. Depending if the terminal can create windows or not, you could be forced to export plots as images instead of visualizing those directly on your screen.

Finally, there is the option of executing Matlab scripts (.m files) calling Matlab as the interpreter of the code. This option is the closest to the original spirit of working on an HPC cluster. The work is prepared to be executed without human intervention once the job executes.

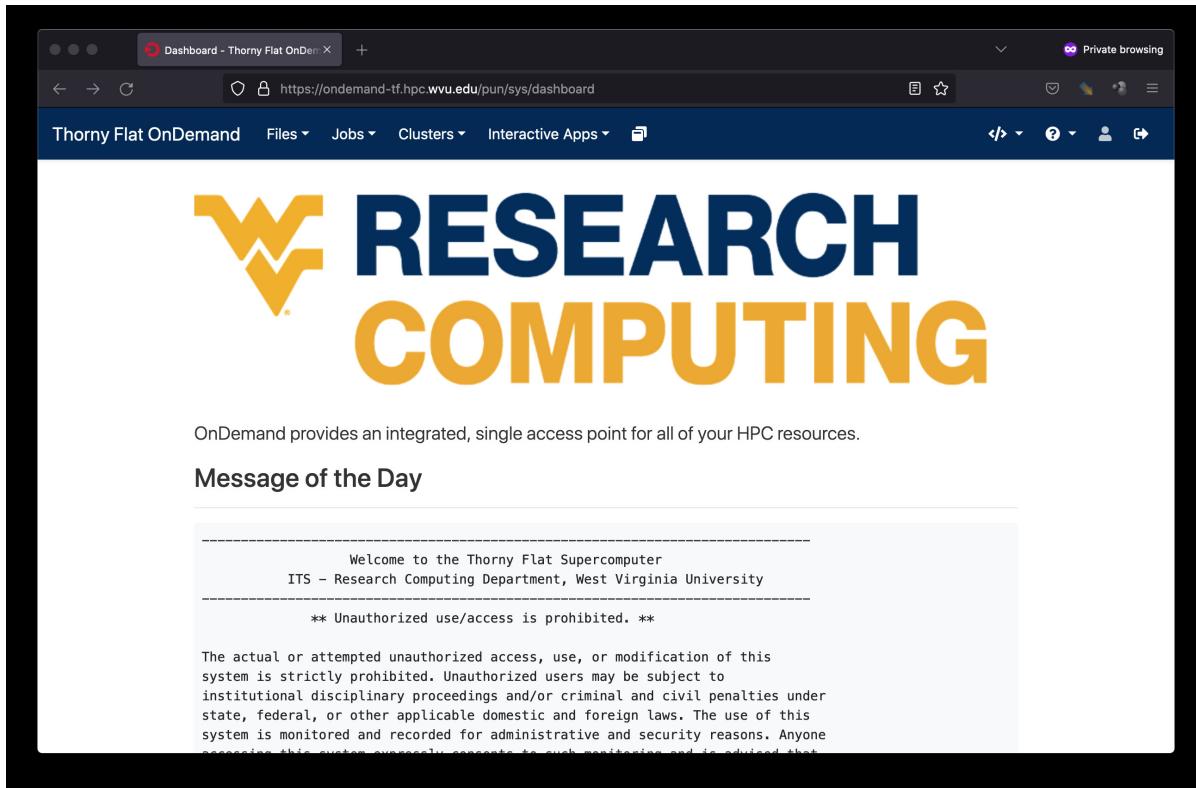
In the following sections we will cover how to request jobs under these three modes.

At the end we will present a small introduction to strategies to improve performance of Matlab jobs. One way is via Compiling the code, another is adding some parallelization to take advantage of multicore compute nodes.

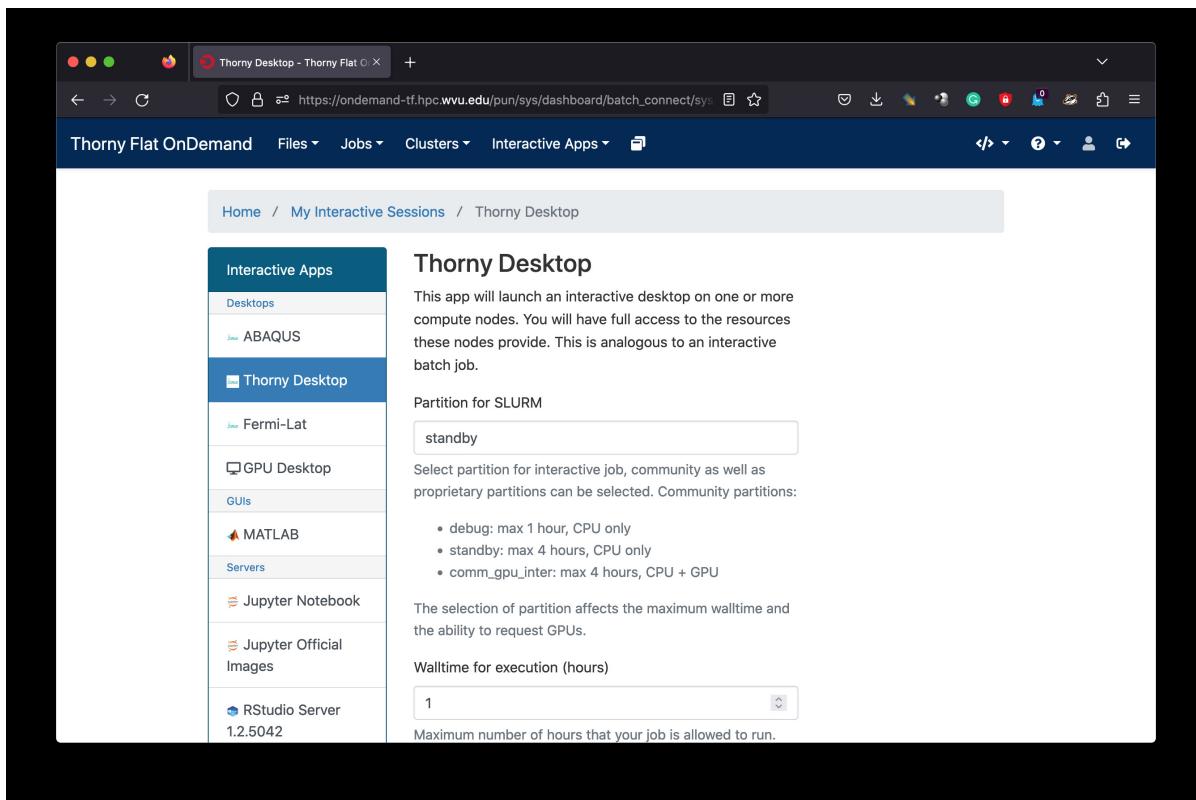
5.5.1 Interactive execution using Open On-Demand

An HPC cluster is a conglomerate of computers intended to be operated as a single machine. This is different from the way you work on a normal desktop computer, where you are completely in control of the applications launched, many of them with graphical interfaces. The HPC cluster offers a web interface called Open On-Demand where you can launch an interactive job that creates a virtual desktop, allowing you to work on the cluster as close as possible as working on a desktop machine. The virtual desktop will be running on a compute node with all the computational power associated with it. The only downside could be the responsiveness to commands depending on how fast is the network between your computer and the HPC cluster.

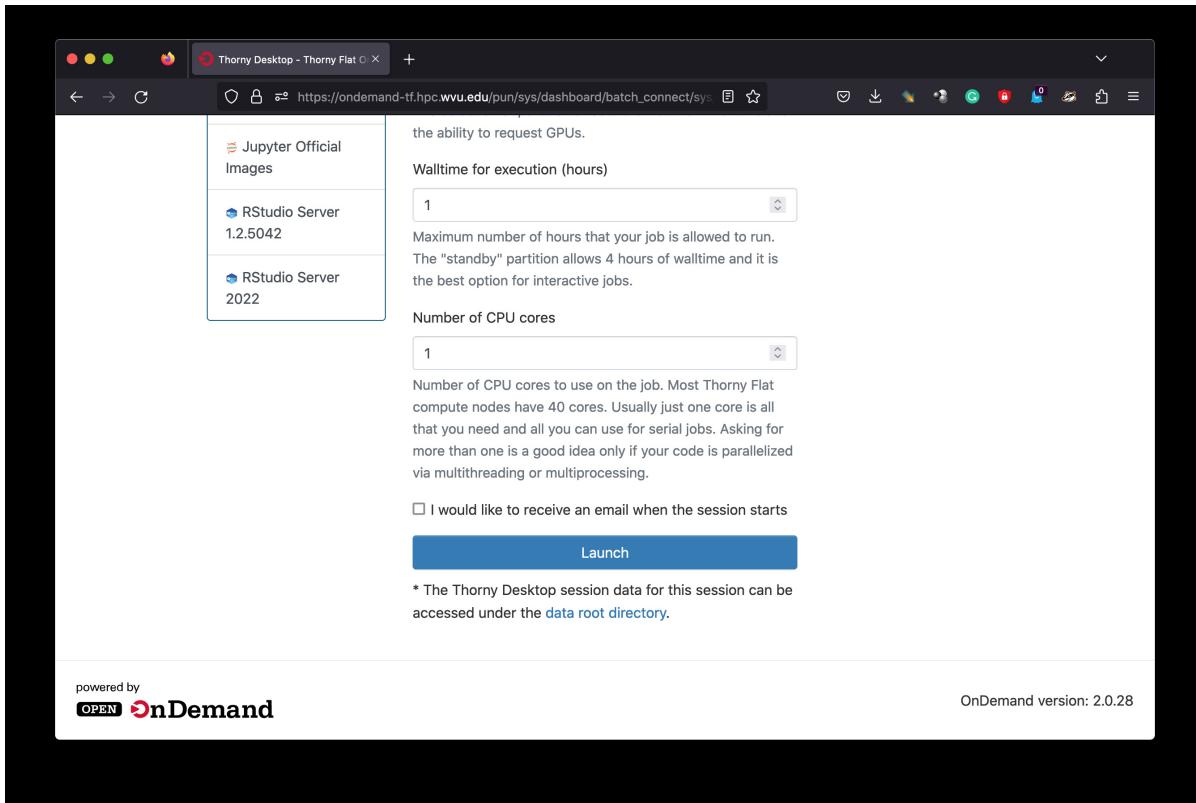
To Access Open On-Demand, direct your web browser to <https://ondemand-tf.hpc.wvu.edu>. Open On-Demand supports modern web browsers such as Mozilla Firefox, Google Chrome, macOS Safari and Microsoft Edge. Once you connect to <https://ondemand-tf.hpc.wvu.edu> you will be directed to WVU Single Sign-On (SSO). You will have to enter your username and password and authorize your access with DUO. Once you succeed in the authorization you will land on the Open On-Demand dashboard shown below.



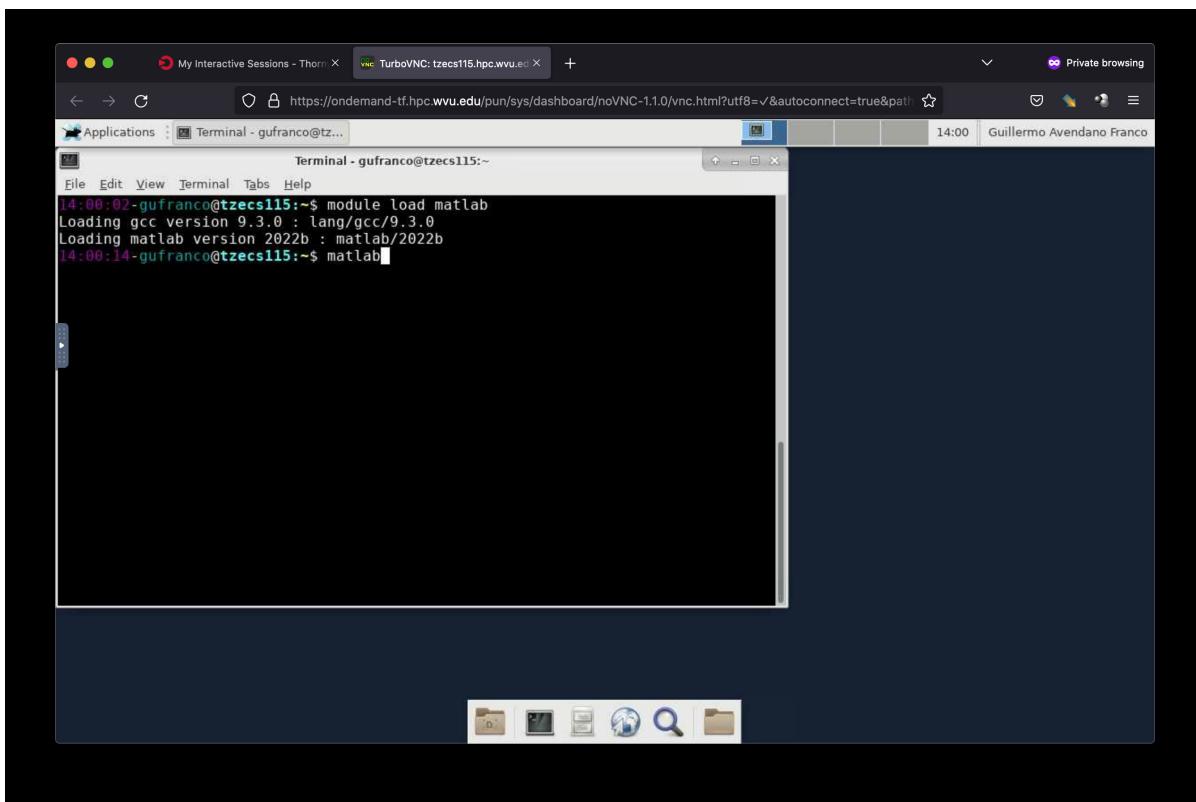
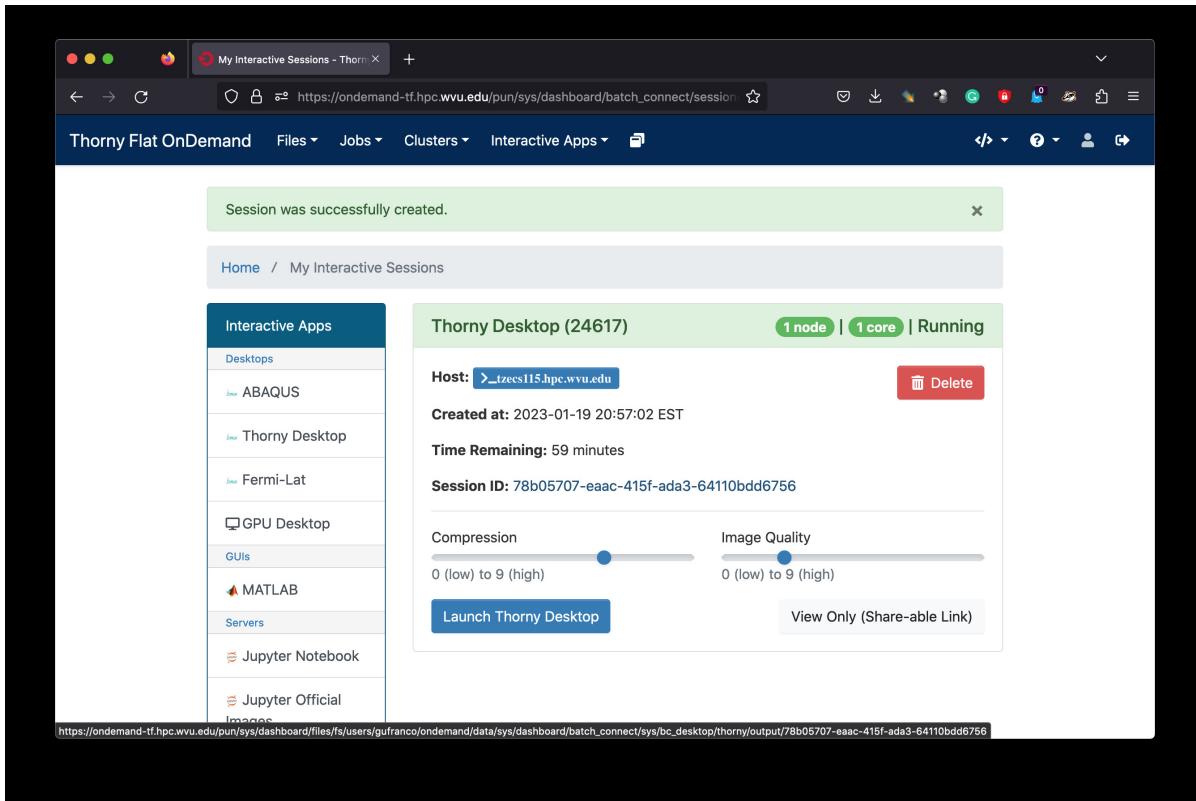
Click on “Interactive Apps” and select “Thorny Desktop” A formulary will ask you to selected the “partition,” “walltime” and number of “CPU cores” to use during your interactive session. The screenshots below show some good values for a normal execution of Matlab on the cluster.



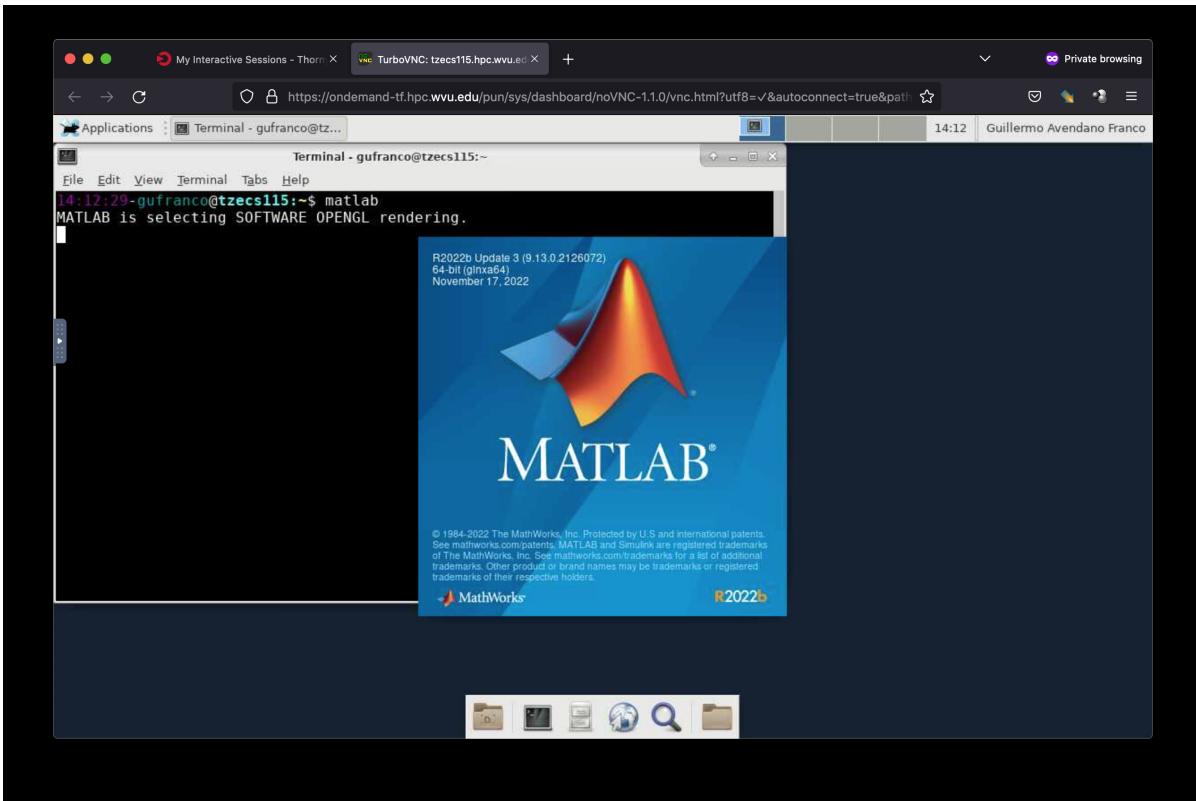
Here we are using “standby” as SLURM partition, 1 hour of walltime and 1 CPU core. There is not much advantage on requesting more than 2 CPU cores if your Matlab code is not using some sort of parallel programming. Go to the bottom and click “Launch”



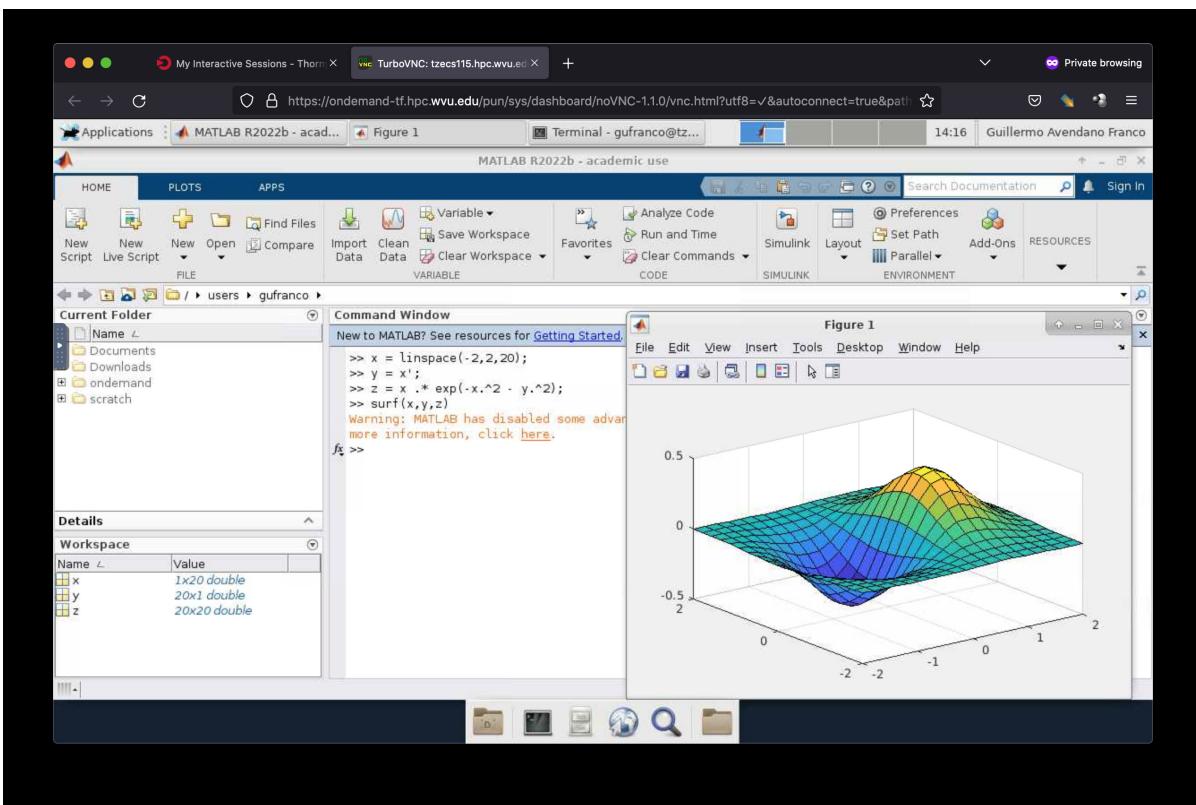
The browser tab will change to show you the current list of “Interactive Sessions” running in Open On-Demand. On the top of the list is your newly created “Thony Desktop” session. It will be waiting from a few seconds to some minutes until a compute node is assigned to the job and the Virtual Desktop is created there. Once the virtual desktop is created a button “Launch Thony Desktop” appears and clicking there will open a new browser tab with a view on a Linux desktop. The Linux desktop selected is very clean to prevent wasting resources. The screenshot below shows how the desktop looks like. Click on the Terminal button and load the module for Matlab



Once the module is loaded execute the command *matlab* on the terminal. With GUI of matlab will open after a few seconds. A screen will show the version being loaded



Now you can start executing matlab commands in the “Command Window”. You can also load .m files and create plots as shown below.



5.5.2 Interactive execution on the terminal

5.5.3 Loading Matlab using Environment Modules

MATLAB is accessible via environment Modules on Thorny Flat. To check which versions are available execute:

```
trcis001:~$ module avail matlab
----- /shared/modulefiles/tier1 -----
matlab/2021a matlab/2022b
```

Load the desired version with:

```
trcis001:~$ module avail matlab/2021a
```

or simply load the latest version executing:

```
trcis001:~$ module load matlab
```

Notice that the commands above load matlab on the login node. You must never use the login node for any execution that last for more than a few seconds. Computationally demanding calculations must be executed via job submissions.

From the login node, request a interactive session with SLURM:

```
trcis001:~$ module load sched/slurm/22.05
trcis001:~$ srun -n 4 --pty bash
srun: job 24328 queued and waiting for resources
srun: job 24328 has been allocated resources
tarcl100:~$
```

```
tarcl100:~$ module load matlab
Loading gcc version 9.3.0 : lang/gcc/9.3.0
Loading matlab version 2022b : matlab/2022b
```

```
tarcl100:~$ matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

< M A T L A B (R) >
Copyright 1984-2022 The MathWorks, Inc.
R2022b Update 3 (9.13.0.2126072) 64-bit (glnxa64)
November 17, 2022
```

To get started, type doc.
For product information, visit www.mathworks.com.

```
>> 2 + 2
```

```
ans =
```

```
4
```

5.5.4 Non-interactive execution via job submission

This simple tutorial explains how to use Matlab without launching the graphical interface and using the submission script.

Consider this simple script that computes first and second derivatives of a function to find extrema and inflection points of a given function.

To download the code you can execute the following command directly on the cluster:

```
trcis001:~$ wget https://docs.hpc.wvu.edu/_static/derivatives.zip
```

Or download the file `derivatives.zip`. And upload the file to the cluster.

You can create a directory for this, for example `MATLAB_TUT1` and uncompress there the file `derivatives.zip`:

```
trcis001:~$ mkdir MATLAB_TUT1
trcis001:~$ cd MATLAB_TUT1
trcis001:~$ wget https://docs.hpc.wvu.edu/_static/derivatives.zip
trcis001:~$ unzip derivatives.zip
```

It will uncompress a file called `derivatives.m`. The file contains the matlab code that we would like to execute on the cluster.

Write a submission script for SLURM. For this example we will use a single core on a single machine (on Thorny Flat):: Use your prefered text editor and type the following lines:

```
#!/bin/bash

#SBATCH --job-name=MATLAB
#SBATCH --output=SLURM-%x.%j.out
#SBATCH --partition=standby

#SBATCH --time=10:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=5000

module purge
module load matlab

cd $SLURM_SUBMIT_DIR
matlab -nodisplay -r derivatives
```

Store this lines into a file called `runjob.slurm`

This submission script is telling the SLURM workload manager that we are creating a job called “MATLAB”, that will use one node (`-ntasks=1`) and one core per node (`-cpus-per-task=2`), during 10 minutes (`-time=10:00`). The partition were the job is submitted is called `standby` but you can also choose other partitions such as `comm_mmem_week` or `comm_mmem_day`. that will offer extended periods of time.

The lines `module purge` and `module load statistics/matlab/2014a` will prepare the environment for executing Matlab on the HPC cluster.

Finally, the script will execute the matlab script “`derivatives.m`” without opening the graphical user interface (GUI). Notice that you should execute `matlab -nodisplay -r derivatives` without adding the “`.m`”. Matlab will always search for a file called “`derivatives.m`”

You submit the job from the command line executing:

```
trcis001:~$ module load sched/slurm/22.05
trcis001:~$ sbatch runjob.slurm
Submitted batch job 26212
```

When the job is submitted you get a number called Job ID. This number is an identifier that allow you to keep track of lifetime of the job. You can monitor the execution using the command `squeue`:

trcis001:~\$ squeue -j 26212	JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
	26212	standby	MATLAB	gufranco	R	0:10	1	tgmcs101

You notice the job is running when the state (ST) shows as *R* and you see under *TIME* for how long the job has been running.

When finished, *squeue* will return an error as the job no longer exist in queue:

```
trcis001:~$ squeue -j 26212
slurm_load_jobs error: Invalid job id specified
```

You can see the job with *sacct* a command for accounting, ie keeping track of the jobs running and complete. When the job is still runnning you get:

trcis001:~\$ sacct -j 26212	JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
	26212	MATLAB	standby	its-rc-ad+	2	RUNNING	0:0
	26212.batch	batch		its-rc-ad+	2	RUNNING	0:0

Once the job finishes, either by a normal termination or because it was canceled:

trcis001:~\$ sacct -j 26212	JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
	26212	MATLAB	standby	its-rc-ad+	2	COMPLETED	0:0
	26212.batch	batch		its-rc-ad+	2	COMPLETED	0:0

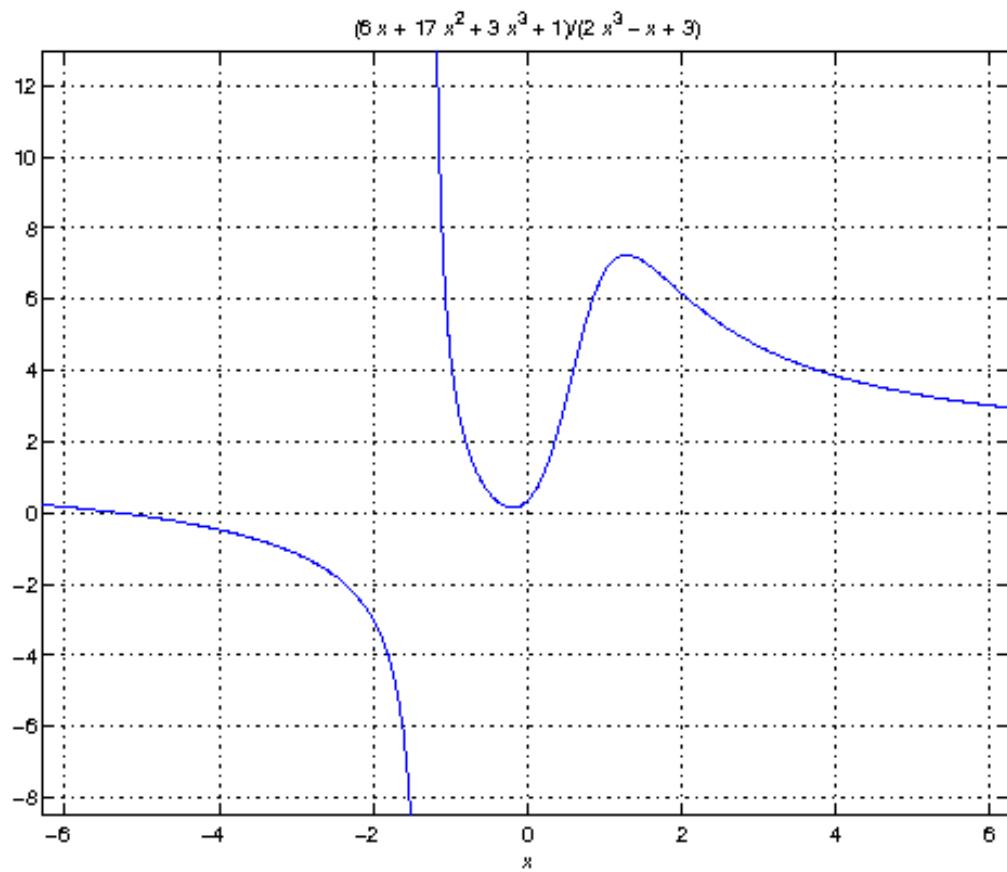
When the job concludes, you will get new files that were created by the job:

```
trcis001:~$ ls -1
derivatives.fig
derivatives.m
derivatives.png
derivatives.zip
runjob.slurm
SLURM-MATLAB.26206.out
```

The figures where generated and saved on the same folder you submit your job. The output of your execution is stored on *SLURM-MATLAB.26206.out*

```
< M A T L A B (R) >
Copyright 1984-2014 The MathWorks, Inc.
R2014a (8.3.0.532) 64-bit (glnxa64)
February 11, 2014
```

(continues on next page)



(continued from previous page)

To get started, **type** one of these: **helpwin**, **helpdesk**, **or demo**.
 For product information, visit www.mathworks.com.

First Derivatives: Finding Local Minima **and** Maxima

f =

$$(3*x^3 + 17*x^2 + 6*x + 1)/(2*x^3 - x + 3)$$

ans =

$$3/2$$

ans =

$$3/2$$

ans =

$$-1/(6*(3/4 - (241^(1/2)*432^(1/2))/432)^(1/3)) - (3/4 - (241^(1/2)*432^(1/2))/432)^(1/3)$$

ans =

$$-1.2896$$

First Derivative: Local extremum Points

g =

$$(9*x^2 + 34*x + 6)/(2*x^3 - x + 3) - ((6*x^2 - 1)*(3*x^3 + 17*x^2 + 6*x + 1))/(2*x^3 - x + 3)^2$$

ans =

$$\begin{aligned} & ((2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3))/1156 + 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/2)/(6*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/6)) + ((337491*6^(1/2)*(3*3^(1/2)*178939632355^(1/2))/9826 + 2198209/9826)^(1/2))/39304 + (2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3)*(2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3))/1156 + 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/2)/578 - 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3)*(2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3))/1156 + 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/2) - (361*((2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3))/176868 + 2198209/530604)^(1/3))/1156 + 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/2) - 289)^(1/2)/(6*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/6)*(2841*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(1/3))/1156 + 9*((3^(1/2)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/2) - 289)^(1/2) - 186)*178939632355^(1/2))/176868 + 2198209/530604)^(2/3) + 361/289)^(1/4)) \end{aligned}$$

(continued from previous page)

$$\begin{aligned}
& ((2841*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^1/3)/1156 + 9*((3^1/ \\
& \rightarrow 2)^178939632355^1/2)/176868 + 2198209/530604)^2/3 + 361/289)^1/2)/(6^*((3^1/ \\
& \rightarrow 2)^178939632355^1/2)/176868 + 2198209/530604)^1/6) - ((337491^*6^1/2)^*((3^3^1/ \\
& \rightarrow 2)^178939632355^1/2)/9826 + 2198209/9826)^1/2)/39304 + (2841*((3^1/ \\
& \rightarrow 2)^178939632355^1/2)/176868 + 2198209/530604)^1/3)^*((2841*((3^1/2)^178939632355^1/ \\
& \rightarrow 2))/176868 + 2198209/530604)^1/3)/1156 + 9*((3^1/2)^178939632355^1/2)/176868 + \\
& \rightarrow 2198209/530604)^2/3 + 361/289)^1/2)/578 - 9*((3^1/2)^178939632355^1/2)/176868 + \\
& \rightarrow 2198209/530604)^2/3)^*((2841*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^1/ \\
& \rightarrow 3)/1156 + 9*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^2/3 + 361/289)^1/2) \\
& \rightarrow (1/2) - (361*((2841*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^1/3))/1156 \\
& \rightarrow + 9*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^2/3 + 361/289)^1/2)/ \\
& \rightarrow 289)^1/2)/(6^*((3^1/2)^178939632355^1/2)/176868 + 2198209/530604)^1/6)^*((2841*((3^1/ \\
& \rightarrow 2)^178939632355^1/2)/176868 + 2198209/530604)^1/3)/1156 + 9*((3^1/ \\
& \rightarrow 2)^178939632355^1/2)/176868 + 2198209/530604)^2/3 + 361/289)^1/4)) - 15/68
\end{aligned}$$

ans =

1.2860
-0.1892

Second Derivatives: Finding Inflection Points

| h =

$$\begin{aligned} & \frac{(18*x + 34)/(2*x^3 - x + 3) - (2*(6*x^2 - 1)*(9*x^2 + 34*x + 6))/(2*x^3 - x + 3)^2}{(12*x^3 + 17*x^2 + 6*x + 1)/(2*x^3 - x + 3)^2} \\ & + \frac{(2*(6*x^2 - 1)^2*(3*x^3 + 17*x^2 + 6*x + 1))/(2*x^3 - x + 3)^3}{(2*x^3 - x + 3)^3} \end{aligned}$$

ans =

**1.8651543689917122385037075917613
0.57871842655441748319601085860196**

5.5.5 Using the Matlab Compiler

The first step is to load matlab to get access to its executables:

```
trcis001:~$ module load matlab
```

Prepare the compilation environment with:

```
trcis001:~$ mbuild -setup  
MBUILD configured to use 'gcc' for C language compilation.
```

```
To choose a different language, execute one from the following:  
mex -setup C++ -client MBUILD  
mex -setup FORTRAN -client MBUILD
```

You cannot compile matlab scripts that uses the symbolic toolbox

https://www.mathworks.com/products/ineligible_programs.html

So we will use another script for this tutorial.

`mandelbrot.zip`

Download the file `mandelbrot.zip`. Or use `wget` directly from the cluster:

```
trcis001:~$ wget https://docs.hpc.wvu.edu/_static/mandelbrot.zip
```

After uncompress the file “`mandelbrot.m`”:

```
trcis001:~$ mcc -m mandelbrot.m
```

It takes a while, when finished you will get some extra files:

```
trcis001:~$ ls -1
mandelbrot
run_mandelbrot.sh
```

Write a submission script for SLURM (Thorny Flat):

```
#!/bin/bash

#SBATCH --job-name=MATLAB
#SBATCH --output=SLURM-%x.%j.out
#SBATCH --partition=standby

#SBATCH --time=10:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=5000

module purge
module load matlab

cd $SLURM_SUBMIT_DIR
./run_mandelbrot.sh $MD_MATLAB
```

After submit the job with:

```
trcis001:~$ module load sched/slurm/22.05
trcis001:~$ sbatch runjob.slurm
Submitted batch job 26219
```

You get the results on “`SLURM-MATLAB.26219.out`” with the corresponding JobID

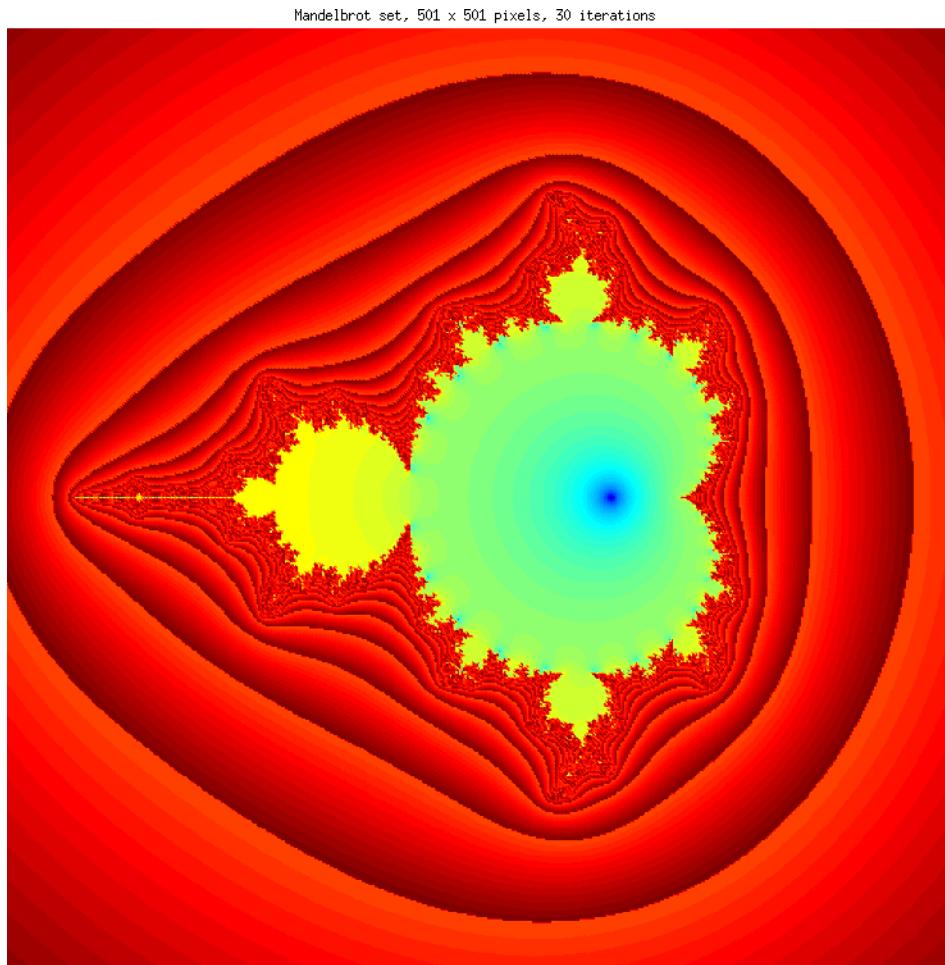


Fig. 1: Mandelbrot fractal generated from Matlab

5.5.6 Summary

5.6 Perl Language

Perl stands in for “Practical Extraction and Reporting Language” even though there is no authorized acronym for Perl. Perl was created by Larry Wall in 1987 when he was employed on a bug reporting system and “AWK”- a programming language he was using for the purpose was not helping him much. He is still the chief architect and developer of Perl. If we want to define Perl in one sentence: Perl is a high-level, interpreted, dynamic programming language.

Perl is a programming language specially designed for text editing. It is now widely used for a variety of purposes including Linux system administration, network programming, web development, etc. In scientific computing, Perl is particularly used in Bioinformatics

Advances in biology have generated an ocean of data that needs careful computational analysis to extract useful information. Bioinformatics uses computation and mathematics to interpret biological data, such as DNA sequence, protein sequence, or three-dimensional structures, and depends on the ability to integrate data of various types. The Perl language is well suited to this purpose.

An important practical skill in bioinformatics is the ability to quickly develop scripts (short programs) for scanning or transforming large amounts of data. Perl is an excellent language for scripting, because of its compact syntax, broad array of functions, and data orientation. The following are a few of the attributes of Perl that make it an attractive choice.

- Perl provides powerful ways to match and **manipulate strings through the use of regular expressions**. Changing file formats from one to another is a matter of contorting strings as required.
- **Modularity** that makes it easy to write programs as libraries (called modules).
- Perl system calls and pipes can be used to incorporate **external programs**.
- Dynamic loaders of Perl that help **extend Perl with programs written in C** as well as create compiled libraries that can be interpreted by the Perl interpreter.
- **Perl is a good prototyping language** and is easy to code. New algorithms can be easily tested in Perl before using a rigorous language.

5.6.1 Using the CPAN module

The CPAN module automates or at least simplifies the make and install of perl modules and extensions. It includes some primitive searching capabilities and knows how to use LWP, [HTTP::Tiny](#), Net::FTP and certain external download clients to fetch distributions from the net.

These are fetched from one or more mirrored CPAN (Comprehensive Perl Archive Network) sites and unpacked in a dedicated directory.

The CPAN module also supports named and versioned bundles of modules. Bundles simplify handling of sets of related modules. See Bundles below.

The package contains a session manager and a cache manager. The session manager keeps track of what has been fetched, built, and installed in the current session. The cache manager keeps track of the disk space occupied by the make processes and deletes excess space using a simple FIFO mechanism.

All methods provided are accessible in a programmer style and in an interactive shell style.

Users can install their own modules from the head node and they will be installed on the \$HOME folder for usage later on any compute node.

5.6.2 Preparing the CPAN Shell

Enter interactive mode by running:

```
perl -MCPAN -e shell
```

or directly by:

```
cpan
```

If this is the first time you are configuring the shell, you will be greeted with a message like:

```
CPAN.pm requires configuration, but most of it can be done automatically.  
If you answer 'no' below, you will enter an interactive dialog for each  
configuration option instead.
```

```
Would you like to configure as much as possible automatically? [yes]
```

You can continue with the automatic configuration, suitable for most users. Next you will see this warning and message:

```
Warning: You do not have write permission for Perl library directories.
```

```
To install modules, you need to configure a local Perl library directory or  
escalate your privileges. CPAN can help you by bootstrapping the local::lib  
module or by configuring itself to use 'sudo' (if available). You may also  
resolve this problem manually if you need to customize your setup.
```

```
What approach do you want? (Choose 'local::lib', 'sudo' or 'manual')  
[local::lib]
```

Normal users are not allowed to add, delete or change files on system folders. The warning is telling you that. On a cluster, as normal user the best choice is to use [local::lib] as it will write all the files in your \$HOME. The folders are \$HOME/.cpan and \$HOME/perl5.

The next message is:

```
Autoconfigured everything but 'urllist'.
```

```
Now you need to choose your CPAN mirror sites. You can let me  
pick mirrors for you, you can select them from a list or you  
can enter them by hand.
```

```
Would you like me to automatically choose some CPAN mirror  
sites for you? (This means connecting to the Internet) [yes]
```

You can select yes to this question and you will be directed to one of the mirrors for CPAN. At WVU we have one mirror that can be used instead. If you want to use it, answer no to the question and add the mirror as shown below:

```
Would you like me to automatically choose some CPAN mirror  
sites for you? (This means connecting to the Internet) [yes] no
```

```
Would you like to pick from the CPAN mirror list? [yes] no  
Now you can enter your own CPAN URLs by hand. A local CPAN mirror can be  
listed using a 'file:' URL like 'file:///path/to/cpan/'
```

(continues on next page)

(continued from previous page)

CPAN.pm needs at least one URL where it can fetch CPAN files from.

```
Please enter your CPAN site: [] http://fireball-public.phys.wvu.edu/mirror/CPAN/
Enter another URL or ENTER to quit: []
New urllist
http://fireball-public.phys.wvu.edu/mirror/CPAN/
```

The next message is about add some lines to your .bashrc:

```
local::lib is installed. You must now add the following environment variables
to your shell configuration files (or registry, if you are on Windows) and
then restart your command line shell and CPAN before installing modules:
```

```
PATH="/users/username/perl5/bin${PATH:+:$PATH}"; export PATH;
PERL5LIB="/users/username/perl5/lib/perl5${PERL5LIB:+:$PERL5LIB}"; export PERL5LIB;
PERL_LOCAL_LIB_ROOT="/users/username/perl5${PERL_LOCAL_LIB_ROOT:+:$PERL_LOCAL_LIB_ROOT}"
˓→"; export PERL_LOCAL_LIB_ROOT;
PERL_MB_OPT="--install_base \"/users/username/perl5\""; export PERL_MB_OPT;
PERL_MM_OPT="INSTALL_BASE=/users/username/perl5"; export PERL_MM_OPT;
```

```
Would you like me to append that to /users/username/.bashrc now? [yes]
```

Answering yes, will add those lines to your .bashrc, something that you can later edit directly.

Finally, you get into the shell:

```
ommit: wrote '/users/gufranco/.cpan/CPAN/MyConfig.pm'

You can re-run configuration any time with 'o conf init' in the CPAN shell
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.9800)
Enter 'h' for help.

cpan[1]>
```

5.6.3 Installing Perl modules

The Comprehensive Perl Archive Network (CPAN) currently has more than 180000 Perl modules and you can install those for your own usage without requesting global installation on the cluster.

For example, lets assume that you want to install `Math::EllipticCurve::Prime` a package for elliptic curves over a prime field. Log into the CPAN shell and execute:

```
cpan[1]> install Math::EllipticCurve::Prime
```

After installation you will see a message and your prompt is returned:

```
Appending installation info to /users/username/perl5/lib/perl5/x86_64-linux-thread-multi/
˓→perllocal.pod
BRIANC/Math-EllipticCurve-Prime-0.003.tar.gz
/usr/bin/make install -- OK
```

(continues on next page)

(continued from previous page)

```
cpan[2]>
```

5.7 Parallel Programming: OpenMP

5.8 Parallel Programming: MPI

5.9 Parallel Programming: OpenACC

The fastest supercomputers today include alongside with the normal CPUs, extra electronic devices specialized to perform certain computational tasks much faster than CPUs, those devices are called Accelerators and the most prominent of those today are Graphical Processing Units (GPUs).

Accelerators typically have their own memory and programming those devices requires being aware of memory management and data movement. The memory on Accelerators is usually smaller than the RAM available on compute nodes and the transfer of data from RAM to the internal memory of the accelerator is usually one order of magnitude slower than between RAM and the CPU.

As GPUs are the most common accelerator today and those are the devices on our clusters we will concentrate the rest of this section to them. We will present two solutions for programming GPUs. OpenACC is a directive based standard for parallel computing with GPUs from several vendors, Intel's Xeon Phi (Discontinued in 2020), Field-programmable gate arrays (FPGAs), and exotic devices such as Digital Signal Processors (DSPs). We will present how to use OpenACC using the NVIDIA HPC SDK.

For the specific case of NVIDIA GPUs is the solution for parallel computing . CUDA is a platform and application programming interface (API) model created by Nvidia for their GPUs, the API have evolved over the years and several libraries have been created to take advantage of those GPUS. On our clusters we have NVIDIA GPUs and we will present how to compile and run basic programs written in CUDA.

5.9.1 OpenACC

OpenACC is a directive based standard to allow developers to take advantage of various kinds of accelerators such as GPUs from NVIDIA. As being a directive base approach it is similar to OpenMP that we saw in a previous section. All that is needed is to add some directives to the original source code in such a way that a OpenACC aware compiler can use those directives to parallelize certain portions of the code. The compiler will generate parallel multithreaded code similar to OpenMP but with the addition of data movement in and out the device as most accelerators will have their own memory and data must be transferred before the accelerator can process that data. For compilers that are not OpenACC capable, will simply ignore those directives and compile a serial code as usual.

Compiler with OpenACC support

The best set of compilers for using OpenACC is the NVIDIA HPC SDK. There is support from GNU Compiler Collection (GCC) starting from GCC 8 that supports OpenACC 2.0 up to GCC 10 that supports OpenACC 2.6. For this section we will use the NVIDIA HPC SDK compilers. To load the compilers execute:

```
module load lang/nvidia/nvhpc
```

Now that you have loaded the module, the three compilers that you can use for OpenACC are *nvfortran*, *nvc* and *nvc++* for Fortran, C and C++ respectively.

Submit a job interactively

You do not need to compile the code on a compute node with GPUs, but for the purpose of learning is good to work directly on an interactive node and test executions and recompile as needed.

The command to request one GPU card for an interactive job is:

```
$> qsub -I -l nodes=1:ppn=8:gpus=1 -q comm_gpu_inter
```

Now, lets start with a very simple example. Consider the following code:

```
#include <stdio.h>
#include <limits.h>
#include <math.h>

// LONG MAX 9223372036854775807
#define N 1000000000

int main()
{
    double pi= 0.0;
    long int i,nmax;

    nmax=(long int)(LONG_MAX/N);

    #pragma acc kernels
    for (i=0; i<(long int)nmax; i++)
    {
        double t = (double)((i+0.05)/nmax);
        pi += 4.0/(1.0+t*t);
    }

    printf("PI using %ld terms in the series is equal to %20.15le\n", nmax, pi/nmax);
    printf("Difference with reference value %e\n", fabs(M_PI-pi/nmax));
}
```

Notice that we are using the a line with the pragma:

```
#pragma acc kernels
```

That line, instructs a compiler with support for OpenACC that the block of code after the pragma, ie, the for loop, could be parallelized. Lets first compile the code without requesting OpenACC:

```
$> nvc pi_acc_d.c
```

After the code is compile the executable by default is the file *a.out*. We can execute this file and timing the execution with:

```
$> time ./a.out
PI using 9223372036 terms in the series is equal to 3.141592653687312e+00
Difference with reference value 9.751933e-11

real    0m24.704s
user    0m24.643s
sys     0m0.003s
```

On the current machine, it takes more than 20 seconds to compute all the series. Now lets compile again using the arguments to *nvc* and ask it to introduce OpenACC for the pragma in the sources:

```
$> nvc pi_acc_d.c -acc -Minfo=all
```

With the addition of *-Minfo=all* we will get some extra information about which portions of the code will be parallelized and how. Repeat the execution for the new binary:

```
$> time ./a.out
PI using 9223372036 terms in the series is equal to 3.141592653687371e+00
Difference with reference value 9.757795e-11

real    0m1.922s
user    0m0.399s
sys     0m1.459s
```

You will observe that it takes less than 2 seconds, a very significant improvement over the serial version.

Example from the XSEDE workshop on OpenACC

Now, we will show how to compile and execute the same code proposed during the XSEDE workshop on OpenACC. To compile the serial code for the fortran example use:

```
$> nvfortran laplace_serial.f90
```

Execute the code with a limit of 4000 iterations:

```
$> echo 4000 | time ./a.out
```

The final lines from the execution are:

```
----- Iteration number:      3300 -----
( 995, 995): 97.66 ( 996, 996): 98.24 ( 997, 997): 98.75 ( 998, 998): 99.19 ( 999, 999):
( 999): 99.56 (1000,1000): 99.87
Max error at iteration      3372 was      9.9953310357534519E-003
Total time was      17.64035      seconds.
17.59user 0.00system 0:17.64elapsed 99%CPU (@avgtext+@avgdata 18484maxresident)k
@inputs+@outputs (@major+5181minor)pagefaults @swaps
```

Now, lets compile the solution proposed during the workshop:

```
$> nvfortran laplace_acc.f90 -acc -Minfo=all
serial:
 44, Generating copy(temperature_last(:,::)) [if not already present]
    Generating create(temperature(:,::)) [if not already present]
 48, Loop is parallelizable
 49, Loop is parallelizable
    Generating Tesla code
 48, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
 49, !$acc loop gang, vector(32) ! blockidx%x threadidx%x
 59, Generating implicit copy(dt) [if not already present]
 60, Loop is parallelizable
 61, Loop is parallelizable
    Generating Tesla code
 60, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
    Generating implicit reduction(max:dt)
 61, !$acc loop gang, vector(32) ! blockidx%x threadidx%x
 70, Generating update self(temperature(:,::))
initialize:
 98, Memory zero idiom, array assignment replaced by call to pgf90_mzero8
```

And test the new binary:

```
----- Iteration number: 3300 -----
( 995, 995): 97.66 ( 996, 996): 98.24 ( 997, 997): 98.75 ( 998, 998): 99.19 ( 999, 999): 99.56 (1000,1000): 99.87
Max error at iteration 3372 was 9.9953310357534519E-003
Total time was 1.402100 seconds.
0.30user 1.32system 0:01.68elapsed 96%CPU (@avgtext+@avgdata 208792maxresident)k
0inputs+0outputs (@major+24657minor)pagefaults 0swaps
```

There is a significant improvement in the execution of the Laplace solver on a compute node using one GPU card.

5.10 Parallel Programming: CUDA

The fastest supercomputers today include alongside with the normal CPUs, extra electronic devices specialized to perform certain computational tasks much faster than CPUs, those devices are called Accelerators and the most prominent of those today are Graphical Processing Units (GPUs).

Accelerators typically have their own memory and programming those devices requires being aware of memory management and data movement. The memory on Accelerators is usually smaller than the RAM available on compute nodes and the transfer of data from RAM to the internal memory of the accelerator is usually one order of magnitude slower than between RAM and the CPU.

As GPUs are the most common accelerator today and those are the devices on our clusters we will concentrate the rest of this section to them. We will present two solutions for programming GPUs. OpenACC is a directive based standard for parallel computing with GPUs from several vendors, Intel's Xeon Phi (Discontinued in 2020), Field-programmable gate arrays (FPGAs), and exotic devices such as Digital Signal Processors (DSPs). We will present how to use OpenACC using the NVIDIA HPC SDK.

For the specific case of NVIDIA GPUs is the solution for parallel computing . CUDA is a platform and application programming interface (API) model created by Nvidia for their GPUs, the API have evolved over the years and several libraries have been created to take advantage of those GPUS. On our clusters we have NVIDIA GPUs and we will present how to compile and run basic programs written in CUDA.

SOFTWARE ADMINISTRATION

6.1 Editing these documents

WVU Research Computing releases documentation in two formats: As HTML pages hosted at <https://docs.hpc.wvu.edu> and a PDF file that can be downloaded from https://docs.hpc.wvu.edu/docs_hpc_wvu.pdf

Github provides the hosting and version control of sources and generated HTML pages and PDF documents. The documentation itself is generated using [Sphinx](#).

[Sphinx](#) is a tool that makes it easy to create beautiful documentation with multiple output formats. Sphinx uses [reStructuredText \(rst\)](#) as its markup language, and many of its strengths come from the power and straightforwardness of [reStructuredText \(rst\)](#) and its parsing and translating suite, the [Docutils](#).

In practice, that means that both sources and HTML files are version controlled using git and stored on Github and editing the webpages implies knowing a bit how sphinx manages the creation of HTML pages and editing files in *rst* format.

This tutorial covers the basics of learning how to download the sources, introducing changes and uploading the resulting sources and HTML files. There are two ways of accomplishing this, the manual process involves having a git client, a recent version of sphinx on your computer and fairly complete installation of LaTeX for the PDF document. The manual procedure should work fine but require the user to be responsible for installing many packages on his/her system eventually installing some of them by hand if the versions on the distro are too old. Currently, the documentation is known to work with Sphinx v1.6.7 released on Feb 4, 2018.

We have simplified the process by using a singularity image that contains all the functionality needed on a single image. The singularity image is based on *ubuntu-bionic* and should be the method of choice in most cases.

We will describe the usage of the singularity image and the convenience script, at the end of the page we describe briefly the manual process.

6.1.1 Using the Singularity image

Both the sources and resulting output formats are stored on Github. The location of the repository is <https://github.com/WVUResearchComputing/WVUResearchComputing.github.io>.

There is a single repository for both using independent branches, the *sphinx* branch contains the sources and the *master* branch the output formats (HTML files and PDF).

The actual documentation can be seen at the URL: <https://docs.hpc.wvu.edu>

You need to have singularity installed on your machine. See [Singularity docs](#) for instructions on how to install singularity on your computer.

Getting the singularity image

The singularity image has two purposes, first, it offers a complete *ubuntu-bionic* installation with recent versions of all the packages needed to download, build and upload the documentation. In addition to that, the image itself has a simple script that automatizes some of the routinely commands that need to be done to clone or pull both branches, build the documentation from sources and commit and push both branches back to Github. In some cases, a single call to the script is preventing the user to execution a number of git commands and makes.

The singularity image is called `docs_hpc_wvu.simg` and it is stored in the Globus endpoint called `wvu#datadepot`, inside `rcadmin/docs_hpc_wvu`, there is nothing secret with this image but its access is currently restricted to HPC administrators.

Getting a shell inside the container

Get a shell inside the container with:

```
singularity shell docs_hpc_wvu.simg
```

Your prompt should change to something like

```
Singularity docs_hpc_wvu.simg:~/>
```

All the normal commands executed inside this shell are contained in the image. As an exercise you can verify the versions of several packages that we will need to compile the documentation files:

```
Singularity docs_hpc_wvu.simg:~> sphinx-build --version
Sphinx (sphinx-build) 1.6.7

Singularity docs_hpc_wvu.simg:~> make --version
GNU Make 4.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Singularity docs_hpc_wvu.simg:~> pdflatex --version
pdfTeX 3.14159265-2.6-1.40.18 (TeX Live 2017/Debian)
kpathsea version 6.2.3
Copyright 2017 Han The Thanh (pdfTeX) et al.
There is NO warranty. Redistribution of this software is
covered by the terms of both the pdfTeX copyright and
the Lesser GNU General Public License.
For more information about these matters, see the file
named COPYING and the pdfTeX source.
Primary author of pdfTeX: Han The Thanh (pdfTeX) et al.
Compiled with libpng 1.6.34; using libpng 1.6.34
Compiled with zlib 1.2.11; using zlib 1.2.11
Compiled with poppler version 0.62.0
```

Downloading the ‘sphinx’ and ‘master’ branches

Once you are inside the container shell. Execute this command to download the branches

```
docs_hpc_wvu -b DOCS download
```

DOCS is the folder where both branches will be cloned. The name of the folder is arbitrary but use it consistently in all your executions as the script expects two folders to be there `master` and `sphinx`

Editing the sources

The singularity image offers atom as an editor. You can use whatever editor you want, inside or outside the image. If you want to use atom from inside the container execute:

```
docs_hpc_wvu -b DOCS edit
```

The convenience of using atom comes from the tree view of files and eventually the ability to add extra packages to work with rst files more effectively. One interesting package in this context is called `language-restructuredtext`. This package provides snippets and syntax highlighting for reStructuredText files.

To install it go to *Edit > Preferences* a new tab opens called *Settings* there you will see a menu called *Install* and search for `language-restructuredtext`, install the package and you will see the different elements of your rst files properly colored.

Editing the documentation involves introducing changes to the rst files, adding or deleting rst files or reordering them. Notice that all rst files are located inside the `text` folder and the name contains a number that follows the structure of the final pages. The pages that are multiples of 10 are chapter covers that add introductions for the chapter and a table of contents. Any renaming of rst files involves reordering those chapter covers to reflect the modifications.

Build the output formats

We are currently supporting two main output formats, multiple HTML files that are the main format on <https://docs.hpc.wvu.edu> and the PDF that is called `docs_hpc_wvu.pdf`

The simplified command will compile both formats and copy the resulting facilitates to master branch:

```
docs_hpc_wvu -b DOCS build
```

The command to compile the PDF version is:

```
docs_hpc_wvu -b DOCS build_pdf
```

You do not need to compile always the PDF version, the HTML is considered the primary format and PDF outputs will be created at least once per semester.

Update the sphinx branch

You can add, commit and push directly with git commands. However, the script offers a quick version that will do all those tasks in one single call:

```
docs_hpc_wvu -b DOCS update_sphinx -m "Changes to ... adding ..."
```

The script will add your *username* and date and append those to the message used for commit.

Update the master branch

The master branch contains HTML files and PDF. The contents are the result of compiling the sources and should not receive direct human intervention. The script provides a single command that copies the HTML files from _build/html to the master branch and the PDF as well. It will clean the index and commit all the files, finally will push the commit to remote. The command is as follows.

```
docs_hpc_wvu -b DOCS update_master -m "Changes to ... adding ..."
```

6.1.2 Editing the documentation manually

Editing the documentation without the assistance of the script could be necessary for large changes complex selection of the files to be staged, testing that sources are able to build successfully and a variety of other conditions. These instructions guide you on the manual execution of git and make commands without using the script.

No matter if you are using the singularity image or you have the packages installed on your computer the first step is to clone both “sphinx” and “master” branches

```
mkdir DOCS
cd DOCS
REPO="https://github.com/WVUREsearchComputing/WVUREsearchComputing.github.io.git"
ORIGIN="git@github.com:WVUREsearchComputing/WVUREsearchComputing.github.io.git"
git clone --single-branch -b sphinx ${REPO} sphinx
git clone --single-branch -b master ${REPO} master
cd sphinx
git remote set-url origin ${ORIGIN}
cd ../master
git remote set-url origin ${ORIGIN}
cd ..
```

After this commands, the DOCS folder will contain two sub-folders, **sphinx** and **master**. The branch **sphinx** has the sources and **master** the generated web pages.

Edit the rst files as needed and go to the *sphinx* folder and execute:

```
make html
make latexpdf
```

The first command is to generate the HTML files that will be located at *_build* and the second command will create the PDF file **docs_hpc_wvu.pdf** also located at *_build*

You can check the HTML files by opening *_build/html/index.html*. Verify that all changes are as expected before updating the master branch.

Another target in the Makefile is *linkcheck* that will search for broken and redirected links all across the pages:

```
make linkcheck
```

Finally, update the master branch, this will remove all contents of that branch and put in place fresh copies of HTML and PDF documents.

```
make update-master
```

At this point, all the remaining commands are git related. Adding the changes to the index with *git add*, commit the changes with *git commit -m "Message"* and pushing the current commit to the remote with *git push*.

You have to do all steps above on both `master` and `sphinx` branches. Remember that the `master` branch is not intended to be edited by hand but being the result of compiling the sources from `sphinx`.

6.2 Installing Packages in User Locations

6.2.1 Overview: GNU Build System

To install packages on GNU/Linux systems, packages that are programmed in C/C++ or other compiled languages need to be compiled and installed. A large majority of these packages will use the GNU Build system, which can be broken into three distinct stages: 1) Configuring the package, 2) Building the package and 3) installing the package. These three stages are controlled by two programs: ‘configure’ and ‘make’. ‘make’ is a Linux utility that is pre-installed on all Linux systems. While ‘configure’ will be come with the package as it provides package specific information to the ‘make’ utility. Installing these packages can be done in three steps, but user specific options will need to be given to make sure the package is installed correctly. This page will provide instructions on how to build these packages without superuser privileges.

Setting up the Directory Structure

Before you can start the build/install process, you need to set up the directory structure. There are many different ideologies and principles on how to do this. Here, I will show only the strategy we use on the Research Computing clusters to install and manage software. Throughout this page, I’m going to use the package name ‘example_pkg-1.4.7.tar.gz’ to demonstrate the commands. This package naming style is very common, and will most likely be the naming convention you encounter.

The directory structure we use, is to have a parent directory for the package name, and then three subsequent directories named ‘build’, ‘source’, and ‘install’. This directory topology allows us to distinguish the source and build trees from each other (more on what this means below), and keep versions in the install directory. That way, if you upgrade or change versions of software, all versions of the same software package can be located in the same parent directory. In our example, this directory structure can be made using:

```
mkdir example_pkg
cd example_pkg
mkdir build source install
```

You then move the tarball into the source directory, and extract the files:

```
mv example_pkg-1.4.7.tar.gz example_pkg/source
cd example_pkg/source
tar xzvf example_pkg-1.4.7.tar.gz
```

This should create the directory `example_pkg-1.4.7`, and this extracted directory is called the ‘source tree’ since it contains only source files and nothing compiled or built.

The Configure Stage

Once the directory structure is set up, we now have to configure the build tree, to compile and install the software package. This is done by running the ‘configure’ script. There are two general steps of configure. First, you need to load the appropriate dependency environments. Second, you need to tell configure where you want to install the software.

On most systems, you will not have to load dependency environments, however, on you do on Research Computing clusters; and this will be the norm for almost all High Performance Computing clusters. Loading dependency environments are done through [module files](#). The dependencies for each particular package will be different, but they should be listed within the installation manual either on the package website or within the readme/install file. The readme/install file is almost always within the top directory of the extracted package directory as README and/or INSTALL. For example, it might be example_pkg-1.4.7/README in our example.

With the correct module files loaded, you now can run the configure script. You will want to provide the ‘–prefix’ option to tell configure where to install the software. You will additionally, want to run the configure script from the build directory that we set up earlier. This is good practice, and allows easy cleaning (what is referred to as VPATH building).

```
cd example_pkg/build
mkdir 1.4.7
cd 1.4.7
../../source/example_pkg-1.4.7/configure --prefix=/users/username/example_pkg/install/1.
↳ 4.7
```

This command will tell configure to install the binaries, libraries, documentation, etc.. in /users/username/example_pkg/install/1.4.7 directory. The configure process will output a large amount of package information to the screen. And if the correct dependencies are loaded and the package is well built, this process should complete without error. If not, it’s almost certainly because of a missing dependency. If you do not understand the error (most will be missing package ‘XXYY’), and you need help to resolve it, please use our [helpdesk](#) ticket system to request help.

The Make/Build Stage

If the configure step finishes without error. The Make/Build step can be completed with the single command from within the build directory (directory you ran configure in):

```
make
```

The Install Stage

If the configure step finishes without error, it is almost certain that the make step will finish (unless the package maintainers incorrectly built the configure script). Therefore, the install step can be completed with the single command. Again from within the build directory:

```
make install
```

This command will put all the binaries, libraries, documentation wherever you specified using the –prefix option.

6.2.2 Additional Documentation

- [GNU Autoconf Manual](#)
- [GNU Automake Manual](#)
- [GNU Make Manual](#)

6.3 Linear Algebra

Linear Algebra is at the core of many of the algorithms used in High Performance Computing. Operations like the addition of two vectors or the product of matrices have been highly optimized for the kind of computers used today for HPC.

The term *array* is used to generically denote vectors matrices and even higher-dimensional arrays of numbers.

There are basically two classes of Linear Algebra algorithms: Dense Linear Algebra deals with operations over arrays where most of the values are non-zero and Sparse Linear Algebra works with arrays where most of the values are zero. Algorithms on both sides are radically different even from the way the arrays are stored in memory.

In the case of dense linear algebra there are well established algorithms and the *de-facto* standard for many scientific applications is BLAS.

BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The routines are organized in three levels:

- Level 1 BLAS perform scalar, vector and vector-vector operations.
- Level 2 BLAS perform matrix-vector operations
- Level 3 BLAS perform matrix-matrix operations.

Because the BLAS are efficient, portable, and widely available, they are commonly used as the foundation for more complex linear algebra functionality like LAPACK.

LAPACK is a set of routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers.

There are several implementations of BLAS and LAPACK. Just to mention three examples we have ATLAS, a self-optimized generation of BLAS routines. OpenBLAS a fork of GotoBLAS, a set of hand-crafted optimizations for specific processor types and MKL, with optimized math routines. Core math functions include BLAS, LAPACK, ScaLAPACK, sparse solvers, fast Fourier transforms, and vector math. The routines in MKL are hand-optimized specifically for Intel processors.

The reference BLAS and LAPACK are available on most Linux Distributions and named `libblas.{a,so}` and `liblapack.{a,so}`.

However, reference versions of BLAS and LAPACK are typically the last packages that you want use for Dense Linear algebra. In general OpenBLAS and MKL outperform the reference BLAS and LAPACK, they provide multithreading and they are optimized for modern multicore processors with vector extensions like AVX.

We have on Spruce and Thorny at least two much better alternatives to the reference BLAS and LAPACK. The Intel MKL and OpenBLAS.

Intel MKL can be used even if you compile with gcc or gfortran. OpenBLAS was compiled on Thorny enabling all the AVX512 extensions available on the CPUs on Thorny.

OpenBLAS and MKL perform very similar. MKL being a bit better but also more numerically unstable.

Atlas is also an implementation of BLAS that is part of the system and several other packages depend of it. Atlas can be found here:

```
/usr/lib64/atlas/libsatlas.so.3  
/usr/lib64/atlas/libsatlas.so.3.10
```

I do not recommend using that for serious Linear Algebra usage. It performs worst than the two options above.

Notice that the standard compiler on RedHat Enterprise Linux (RHEL) is GCC 4.x. Thorny runs RHEL 7.6 and the compiler is GCC 4.8. Spruce Knob uses RHEL 6.10 and the compiler is 4.4.

GCC 4.x has no programming for optimize code for AVX512, that is the main reason to use newer compilers like GCC 8.x when compiling scientific code.

6.3.1 Compiling Reference BLAS and LAPACK

Now we will show how to compile the reference BLAS and LAPACK. The instructions were created for LAPACK 3.8

Download the code

The Reference BLAS and LAPACK are hosted on Netlib <http://www.netlib.org/lapack/> The sources for LAPACK also contain the reference BLAS so all that we need is to download is the package for lapack:

```
wget http://www.netlib.org/lapack/lapack-3.8.0.tar.gz
```

Uncompress and untar the file with:

```
tar -zxvf lapack-3.8.0.tar.gz
```

That will generate a folder called lapack-3.8.0 with all the sources inside.

Patching the sources to support shared libraries and optimization for AVX512

As they are the sources only allow for static libraries when using the Makefile. There is also an option to use cmake, and we will cover that later on.

Using make, several files need to be changed. The patch file can be downloaded from here [lapack.patch](#) The content of the file is this:

```
diff -ruN lapack-3.8.0/BLAS/SRC/Makefile lapack-3.8.0_new/BLAS/SRC/Makefile
--- lapack-3.8.0/BLAS/SRC/Makefile      2017-11-12 23:15:54.000000000 -0500
+++ lapack-3.8.0_new/BLAS/SRC/Makefile      2019-11-12 13:23:07.985953000 -0500
@@ -55,7 +55,10 @@
 #
 #####
 
-all: $(BLASLIB)
+all: $(BLASLIB) libblas.so
+
+libblas.so: $(ALLOBJ)
+    gfortran -shared -Wl,-soname,$@ -o $@ $(ALLOBJ)
+
#-----
```

(continues on next page)

(continued from previous page)

```

# Comment out the next 6 definitions if you already have
diff -ruN lapack-3.8.0/Makefile lapack-3.8.0_new/Makefile
--- lapack-3.8.0/Makefile      2017-11-12 23:15:54.000000000 -0500
+++ lapack-3.8.0_new/Makefile 2019-11-12 13:11:37.379427000 -0500
@@ -147,3 +147,9 @@
        $(MAKE) -C BLAS cleantest
        $(MAKE) -C CBLAS cleantest
        $(MAKE) -C TESTING cleantest
+
+install: all
+    cp BLAS/SRC/libblas.so $(PREFIX)/lib
+    cp SRC/liblapack.so $(PREFIX)/lib
+    cp *.a $(PREFIX)/lib
+
diff -ruN lapack-3.8.0/make.inc lapack-3.8.0_new/make.inc
--- lapack-3.8.0/make.inc      1969-12-31 19:00:00.000000000 -0500
+++ lapack-3.8.0_new/make.inc 2019-11-12 13:30:49.341408000 -0500
@@ -0,0 +1,88 @@
+#####
+# LAPACK make include file. #
+# LAPACK, Version 3.8.0 #
+# November 2017 #
+#####
+
+SHELL = /bin/sh
+
+##INSTALLATION PATH
+PREFIX=/shared/software/libs/blas_lapack/3.8_gcc82
+
+## CC is the C compiler, normally invoked with options CFLAGS.
+#
+CC      = gcc
+CFLAGS = -O2 -fPIC -g -pipe -march=skylake-avx512
+
+## Modify the FORTRAN and OPTS definitions to refer to the compiler
+## and desired compiler options for your machine. NOOPT refers to
+## the compiler options desired when NO OPTIMIZATION is selected.
+#
+## Note: During a regular execution, LAPACK might create NaN and Inf
+## and handle these quantities appropriately. As a consequence, one
+## should not compile LAPACK with flags such as -ffpe-trap=overflow.
+#
+FORTTRAN = gfortran
+OPTS     = -O2 -frecursive -fPIC -g -pipe -march=skylake-avx512
+DRVOPTS = $(OPTS)
+NOOPT    = -O0 -frecursive -fPIC -g -pipe -march=skylake-avx512
+
+## Define LOADER and LOADOPTS to refer to the loader and desired
+## load options for your machine.
+#
+LOADER   = gfortran
+LOADOPTS =

```

(continues on next page)

(continued from previous page)

```

+
+## The archiver and the flag(s) to use when building an archive
+## (library). If your system has no ranlib, set RANLIB = echo.
+#
+ARCH      = ar
+ARCHFLAGS = cr
+RANLIB    = ranlib
+
+## Timer for the SECOND and DSECND routines
+#
+## Default: SECOND and DSECND will use a call to the
+## EXTERNAL FUNCTION ETIME
+##TIMER = EXTETIME
+## For RS6K: SECOND and DSECND will use a call to the
+## EXTERNAL FUNCTION ETIME_
+##TIMER = EXTETIME_
+## For gfortran compiler: SECOND and DSECND will use a call to the
+## INTERNAL FUNCTION ETIME
+##TIMER = INTETIME
+## If your Fortran compiler does not provide etime (like Nag Fortran
+## Compiler, etc...) SECOND and DSECND will use a call to the
+## INTERNAL FUNCTION CPU_TIME
+##TIMER = INTCPU_TIME
+## If none of these work, you can use the NONE value.
+## In that case, SECOND and DSECND will always return 0.
+##TIMER = NONE
+
+## Uncomment the following line to include deprecated routines in
+## the LAPACK library.
+#
+##BUILD_DEPRECATED = Yes
+
+## LAPACKE has the interface to some routines from tmglbl.
+## If LAPACKE_WITH_TMG is defined, add those routines to LAPACKE.
+#
+##LAPACKE_WITH_TMG = Yes
+
+## Location of the extended-precision BLAS (XBLAS) Fortran library
+## used for building and testing extended-precision routines. The
+## relevant routines will be compiled and XBLAS will be linked only
+## if USEXBLAS is defined.
+#
+##USEXBLAS = Yes
+##XBLASLIB = -lxblas
+
+## The location of the libraries to which you will link. (The
+## machine-specific, optimized BLAS library should be used whenever
+## possible.)
+#
+##BLASLIB      = ../../librefblas.a
+##CBLASLIB     = ../../libcblas.a
+##LAPACKLIB    = liblapack.a

```

(continues on next page)

(continued from previous page)

```
+TMGLIB      = libtmglib.a
+LAPACKELIB  = liblapacke.a
diff -ruN lapack-3.8.0/SRC/Makefile lapack-3.8.0_new/SRC/Makefile
--- lapack-3.8.0/SRC/Makefile 2017-11-12 23:15:54.000000000 -0500
+++ lapack-3.8.0_new/SRC/Makefile      2019-11-12 13:26:04.437198491 -0500
@@ -512,7 +512,10 @@
 DEPRECATED = $(DEPRECSRC)
 endif

-all: ..$(LAPACKLIB)
+all: ..$(LAPACKLIB)  liblapack.so
+
+liblapack.so: $(ALLOBJ)
+    gfortran -shared -Wl,-soname,$@ -o $@ $(ALLOBJ)

 ..$(LAPACKLIB): $(ALLOBJ) $(ALLXOBJ) $(DEPRECATED)
 $(ARCH) $(ARCHFLAGS) $@ $^
```

You can download the patch to the folder `lapack-3.8.0` and execute:

```
patch -s -p1 < lapack.patch
```

Another way is to edit the files one by one.

For the file `lapack-3.8./BLAS/SRC/Makefile` you need to add a new target “`libblas.so`” and adding the target to “`all`”:

```
all: $(BLASLIB) libblas.so

libblas.so: $(ALLOBJ)
gfortran -shared -Wl,-soname,$@ -o $@ $(ALLOBJ)
```

The file `lapack-3.8./Makefile` should have now:

```
install: all
    cp BLAS/SRC/libblas.so $(PREFIX)/lib
    cp SRC/liblapack.so $(PREFIX)/lib
    cp *.a $(PREFIX)/lib
```

6.3.2 Installing Numpy and Scipy using OpenBLAS

For the particular case of Python BLAS/LAPACK are very important for two packages Numpy and Scipy many scientific code written in Python relies on them.

The various environment modules for Python include Numpy and Scipy and they were specifically compiled to use OpenBLAS for linear algebra routines.

The instructions below show how you can do the same for clean virtual environments where you want to install numpy and scipy.

Assume that you want to use PyPy3 from module:

```
$ module purge
$ module load lang/python/pypy3.6-7.1.1-portable
```

The first thing to do is to also load GCC 8.2 and OpenBLAS 0.3.7:

```
$ module load lang/gcc/8.2.0 libs/openblas/0.3.7_gcc82
```

We start creating the virtual environment with `virtualenv`. Notice that PyPy comes with its own version of `virtualenv` called `virtualenv-pypy`:

```
$ virtualenv-pypy pypy3.6
```

Activate the environment, create a build folder inside and change to that folder:

```
$ source pypy3.6/bin/activate
$ mkdir pypy3.6/build
$ cd pypy3.6/build
```

Now we want to install numpy and scipy, do not try to use `pip install ...`, it will bring generic versions using the reference BLAS and Lapack.

Download and extract the packages manually:

```
$ wget https://github.com/numpy/numpy/releases/download/v1.17.3/numpy-1.17.3.tar.gz
$ wget https://github.com/scipy/scipy/releases/download/v1.3.1/scipy-1.3.1.tar.gz
$ tar -zxf numpy-1.17.3.tar.gz
$ tar -zxf scipy-1.3.1.tar.gz
```

First configure numpy. Go to the folder and create a file `site.cfg` with this content:

```
[ALL]
library_dirs = /shared/software/libs/openblas/0.3.7_gcc82/lib
include_dirs = /shared/software/libs/openblas/0.3.7_gcc82/include

[openblas]
libraries = openblas
library_dirs = /shared/software/libs/openblas/0.3.7_gcc82/lib
include_dirs = /shared/software/libs/openblas/0.3.7_gcc82/include
runtime_library_dirs = /shared/software/libs/openblas/0.3.7_gcc82/lib
```

You can build and install numpy with pip, just execute:

```
$ pip3 install .
```

It takes a while and you get numpy installed and the installation tracked by pip. The configuration and installation Scipy is very similar:

```
$ cd ../scipy-1.3.1/
$ cp ../numpy-1.17.3/site.cfg .
$ pip3 install .
```

With numpy and scipy installed we can install any other packages that you plan for the virtual environment. As an example, we will install ipython and pytest:

```
$ pip3 install pytest ipython
(pypy3.6) 12:31:11-gufranco@trcis001:~/scratch/pypy3.6$ ipython
Python 3.6.1 (784b254d669919c872a505b807db8462b6140973, Apr 16 2019, 18:18:28)
```

(continues on next page)

(continued from previous page)

```
Type 'copyright', 'credits' or 'license' for more information
IPython 7.9.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np

In [2]: np.test()
NumPy version 1.17.3
NumPy relaxed strides checking option: True
..... [ 0%]
.....
...sssssss..... [100%]
10090 passed, 116 skipped, 173 deselected, 18 xfailed, 4 xpassed in 274.53s (0:04:34)
```

The configuration flags for numpy and scipy can be obtained from the command below:

```
(pypy3.6) 13:21:20-gufranco@trcis001:~/scratch/pypy3.6$ ipython
Python 3.6.1 (784b254d669919c872a505b807db8462b6140973, Apr 16 2019, 18:18:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.9.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import scipy

In [2]: scipy.__config__.show()
lapack_mkl_info:
    NOT AVAILABLE
openblas_lapack_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
lapack_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
blas_mkl_info:
    NOT AVAILABLE
blis_info:
    NOT AVAILABLE
openblas_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
    runtime_library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
blas_opt_info:
    libraries = ['openblas', 'openblas']
    library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
    language = c
    define_macros = [('HAVE_CBLAS', None)]
```

(continues on next page)

(continued from previous page)

```
runtime_library_dirs = ['/shared/software/libs/openblas/0.3.7_gcc82/lib']
```

6.3.3 Multithreading with OpenBLAS

Another reason to prefer OpenBLAS or MKL over the reference BLAS and LAPACK is that optimized versions provide multithreading, multithreading allows several cores that share the same memory space to work on the arrays concurrently.

OpenBLAS was compiled multithreaded and the number of threads can be controlled with several environment variables:

```
export OPENBLAS_NUM_THREADS=4  
export GOTO_NUM_THREADS=4  
export OMP_NUM_THREADS=4
```

The priorities are OPENBLAS_NUM_THREADS > GOTO_NUM_THREADS > OMP_NUM_THREADS.

See more about this here:

<https://github.com/xianyi/OpenBLAS>

On the case of MKL there are two versions -mkl_sequential and -lmkl_intel_thread Setting the selection of libraries for MKL is complex, but I always use the link advisor to help me up

<https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor>

The only case where I can think someone would like to use the reference BLAS and LAPACK is if you want to check a numerical issue with the optimized versions but in production, those versions should never be used.

6.3.4 ScaLapack 2.2.0 on Spruce with GCC 10.3.0

Modules:

```
$> module load lang/gcc/10.3.0 parallel/openmpi/3.1.6_gcc103 libs/openblas/0.3.19_gcc103  
→ dev/cmake/3.23.0
```

Download the software:

```
$> wget https://github.com/Reference-ScaLAPACK/scalapack/archive/refs/tags/v2.2.0.tar.gz  
→ -O scalapack-2.2.0.tar.gz  
$> tar -zvxf scalapack-2.2.0.tar.gz  
$> cd scalapack-2.2.0
```

Use cmake:

```
$> cmake -DCMAKE_Fortran_COMPILER=mpif90 -DCMAKE_C_COMPILER=mpicc \  
-DCMAKE_Fortran_FLAGS=-fallow-argument-mismatch \  
-DCMAKE_INSTALL_PREFIX=/shared/software/libs/scalapack/2.2.0_gcc103_ompi316 \  
-DCMAKE_POSITION_INDEPENDENT_CODE=ON ..
```

Run make:

```
$> make
```

6.3.5 ELPA 2021.11 on Spruce with GCC 10.3

Modules:

```
$> module load lang/gcc/10.3.0 libs/openblas/0.3.19_gcc103 libs/scalapack/2.2.0_gcc103_
  ↵ompi407 \
  dev/cmake/3.23.0 parallel/openmpi/4.0.7_gcc103
```

Configure

```
$> ./configure --prefix=/shared/software/libs/elpa/2021.11_gcc103_ompi407 \
LDFLAGS="-L${MD_OPENBLAS}/lib -L${MD_SCALAPACK}/lib" \
--disable-sse --disable-avx --disable-avx2 --disable-avx512 CFLAGS=-fPIC
```

6.3.6 ELPA 2021.11 on Thorny with Intel 2022

Modules:

```
$> module load lang/gcc/9.3.0 parallel/cuda/11.7 compiler mpi mkl
```

Configure:

```
$> export SCALAPACK="-L/gpfs20/shared/software/intel/oneapi/mkl/2022.1.0/lib/intel64 -
  ↵lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lmkl_blacs_
  ↵intelmpi_lp64 -liomp5 -lpthread -lm -ldl"
$> ./configure --enable-avx512 --enable-nvidia-gpu CC=mpiicc CXX=mpiicpc FC=mpiifort_
  ↵SCALAPACK_LDFLAGS="$SCALAPACK" --with-NVIDIA-GPU-compute-capability=sm_61 --prefix=/
  ↵shared/software/libs/elpa/2021.11.002_intel22_impi22
```

6.4 Boost 1.79

Boost provides free peer-reviewed portable C++ source libraries. Many scientific codes uses Boost as it provides template-based sources that facilitate development of numerical algorithms.

Compilation of boost libraries is a bit unusual. Boost can be used a template only library with no compilation involved. However, if you want to use any of the separately-compiled Boost libraries, you'll need to acquire library binaries.

On Spruce and Thorny, the native versions of GCC and Python are too old for compiling boost. For that reason, GCC 9.3 and Python 3.9.7 will be used instead.

6.4.1 Downloading the code

By April, 2022, the latest version available is 1.79 that can be downloaded with [boost_1_79_0.tar.bz2](#)

On the clusters the canonical location for sources is /shared/src. Download the archive there and uncompress it:

```
wget https://boostorg.jfrog.io/artifactory/main/release/1.79.0/source/boost_1_79_0.tar.
  ↵bz2
tar -jxvf boost_1_79_0.tar.bz2
cd boost_1_79_0
```

6.4.2 Bootstrap for GCC 9.3

Load the modules for GCC and Python:

```
module load lang/gcc/9.3.0 lang/python/cpython_3.9.7_gcc93
```

With those two modules, before compiling boost, run the bootstrap script indicating python 3.x executable and location for installation:

```
./bootstrap.sh --with-python=python3 --prefix=/shared/software/libs/boost/1.79_gcc93 --  
--with-libraries=all
```

6.4.3 Compile and install boost

Boost libraries can be compiled and installed with:

```
./b2 install --prefix=/shared/software/libs/boost/1.79_gcc93
```

6.5 Message Passing Interface

Many parallel programs take advantage of Message Passing Interface (MPI) to communicate across compute nodes and efficiently scale up computational analysis. While we can't give instructions on how to run every program using MPI, all the programs using MPI need to be told which hosts and cpus to run. Generally, most programs use the mpirun executable that comes pre-packaged with MPI libraries to accomplish this task. The general syntax for running parallel programs with mpirun is:

```
mpirun -np <processor number> -machinefile <procs file> command [..args]
```

With -np option specifying the number of processors the job requires, and the -machinefile option with a file that contains the host address and number of CPUs to use on each host. On a distributed computing cluster with a scheduler in place (both RC clusters), the number of processors will be the same number of processors requested with the [qsub command](#). Further, it is almost impossible to know which hosts your job will run on, since that is determined by a number of factors with the current state of the system. Therefore Moab/Torque provides a machinefile for each job that is executed through the scheduler, and it's location is stored in the \$PBS_NODEFILE variable. Therefore, on our clusters, the general syntax of running a program parallel with MPI is:

```
module load <mpi/modulefile>  
  
mpirun -np <processor number> -machinefile $PBS_NODEFILE command [..args]
```

with being the appropriate [module file](#) for the MPI package the program was compiled with. Of course, you should consult the programs documentation for details, as some programs come with their own mpirun substitute and/or work better with certain MPI libraries than others. Also, each MPI libraries have differences with their mpirun commands, and therefore you should take a look at the MPI's own documentation (links below).

6.5.1 MPI Standards

Message Passing Interface standards are available free for download at [MPI Forum](#).

6.5.2 Useful Books

Parallel Programming with MPI. Peter Pacheco. ISBN-13:978-1558603394

Parallel Programming in C with MPI and OpenMP. Michael Quinn. ISBN-13:978-0072822564

MPI: The Complete Reference. Snir et al. ISBN-13:978-0262692168

6.5.3 Libraries

Current information about location and version of library can be found in their respective modulefiles.

Library	Documentation
Intel	https://software.intel.com/en-us/mpi-library/documentation/get-started
Mpich2	http://www.mpich.org/documentation/guides/
Mvapich2	http://mvapich.cse.ohio-state.edu/support/
OpenMPI	http://www.open-mpi.org/doc/

6.6 HDF5 and NetCDF

HDF5 and NetCDF are open source formats for storing large numerical data in binary format. The latest version of NetCDF uses HDF5 as underling format so it is a dependency when compiled from sources.

6.6.1 HDF5

Several options can be used for compiling HDF5 in particular is important to activate the fortran, C++ and parallel IO with MPI, however C++ is not compatible with MPI and needs to be disable when compiling with MPI

The configure lines below were used to compile HDF5 version 1.10.5

Native compiler on Red Hat Enterprise Linux (RHEL) on Spruce

Serial Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc44 --enable-fortran --enable-cxx
```

Parallel Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc44_ompi31 --enable-fortran --enable-parallel
```

Native compiler on Red Hat Enterprise Linux (RHEL) on Thorny

Serial Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc48 --enable-fortran --enable-
↪cxx
```

Parallel Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc48_ompi31 --enable-fortran --
↪enable-parallel
```

GCC 8.2.0 on Spruce and Thorny

Serial Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc82 --enable-fortran --enable-
↪cxx
```

Parallel Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_gcc82_ompi31 --enable-fortran --
↪enable-parallel
```

Intel Compiler Suite 2018 on Spruce and Thorny

Serial Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_intel18 --enable-fortran --
↪enable-cxx
```

Parallel Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_intel18_impi18 --enable-fortran -
↪enable-parallel
```

Intel Compiler Suite 2019 on Spruce and Thorny

Serial Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_intel19 --enable-fortran --
↪enable-cxx
```

Parallel Version:

```
./configure --prefix=/shared/software/libs/hdf5/1.10.5_intel19_impi19 --enable-fortran -
↪enable-parallel
```

6.6.2 NetCDF

There are separate source for C, Fortran and C++,

Modules loaded:

```
module load lang/intel/2019 libs/hdf5/1.10.5_intel19
```

Configuration line for the C library:

```
CC=icc FC=ifort CXX=icpc ./configure --prefix=/shared/software/libs/netcdf/4.7.1_intel19
```

Results from tests for C:

```
=====
Testsuite summary for netCDF 4.7.1
=====
# TOTAL: 1
# PASS: 1
# SKIP: 0
# XFAIL: 0
# FAIL: 0
# XPASS: 0
# ERROR: 0
=====
```

At this point the module for NetCDF is created and loaded before compiling the Fortran and C++ libraries that rely on it.

Modules loaded:

```
module load lang/intel/2019 libs/hdf5/1.10.5_intel19 libs/netcdf/4.7.1_intel19
```

Configuration line for the fortran wrappers:

```
CC=icc FC=ifort CXX=icpc ./configure --prefix=/shared/software/libs/netcdf/4.7.1_intel19
```

Results from tests for Fortran:

```
=====
Testsuite summary for netCDF-Fortran 4.5.2
=====
# TOTAL: 2
# PASS: 2
# SKIP: 0
# XFAIL: 0
# FAIL: 0
# XPASS: 0
# ERROR: 0
=====
```

The C++ library is configured the same:

```
CC=icc FC=ifort CXX=icpc ./configure --prefix=/shared/software/libs/netcdf/4.7.1_intel19
```

Results from tests for C++ are:

```
=====  
Testsuite summary for netCDF-cxx4 4.3.1  
=====
```

```
# TOTAL: 8  
# PASS: 8  
# SKIP: 0  
# XFAIL: 0  
# FAIL: 0  
# XPASS: 0  
# ERROR: 0  
=====
```

6.7 Fast Fourier Transforms

6.8 Force Field Molecular Dynamics

These are instructions for compiling a variety of atomistic codes. By atomistic codes, we include codes that simulate the behavior of particles such as LAMMPS, codes for Classical Molecular Dynamics (CMD) such as AMBER, GROMACS, and NAMD, Tight Binding codes such as DFTB+ and DFT codes such as ABINIT, OCTOPUS, VASP, and Quantum Espresso.

For compiling these codes we need Fortran, C, and C++ compilers, we will use GCC 9.3 and Intel 2019 for most of these codes.

A set of numerical libraries are used very often by these codes. Dense Linear Algebra routines such as BLAS and LAPACK are used. Optimized versions such as OpenBLAS and Intel MKL are preferable over the reference versions from NetLIB.

All atomistic codes in our list take advantage of parallelization, either OpenMP, MPI, or support GPUs. OpenMP is implemented on modern compilers such as GCC, Intel, and NVIDIA. For MPI we will use MPICH 3.4.1, OpenMPI 3.1.6, and Intel MPI from 2019.

Other libraries needed for compiling these codes include an FFT library such as FFTW 3.3.9 or the implementation in MKL. The libraries need to be compiled for single and double precision as some codes use both. An HDF5 and NetCDF libraries provide hierarchical data storage for numerical data. Finally, a Python implementation is often needed, because these codes include a Python interface or use it for building or testing.

We will start with PLUMED, a library for sampling algorithms, free-energy calculations, and other high-level calculations on top of MD packages. W

6.8.1 Plumed

PLUMED is an open-source library that provides a wide range of different methods that work on top of other Molecular Dynamics codes. Capabilities of the code include:

- Enhanced-sampling algorithms
- Free-energy methods
- Tools to analyze the vast amounts of data produced by molecular dynamics (MD) simulations.

We compile PLUMED with the following modules:

```
module load lang/gcc/9.3.0 parallel/mpich/3.4.1_gcc93 \
lang/python/cpython_3.9.5_gcc93 libs/openblas/0.3.13_gcc93 \
libs/fftw/3.3.9_gcc93
```

Download PLUMED from:

```
wget https://github.com/plumed/plumed2/releases/download/v2.7.1/plumed-2.7.1.tgz
```

Uncompress and configure the build:

```
tar -zxf plumed-2.7.1.tgz
cd plumed-2.7.1
./configure --prefix=/shared/software/atomistic/plumed/2.7.1_gcc93_mpic341 PYTHON_
BIN=python3 --enable-external-blas --enable-external-lapack LDFLAGS=-L${MD_OPENBLAS}/
--lib LIBS=-lopenblas
```

After configuring the build. Execute:

```
make -j 12
make install
```

PLUMED includes a test suite and it is always a good practice to test the compilation. Call make to run the test suite:

```
make check
```

The final lines of the tests show no failures:

```
...
+ check file ves/rt-td-vonmises/report.txt for more information
+ test ves/rt-VesDeltaF/ NOT APPLICABLE
+ check file ves/rt-VesDeltaF/report.txt for more information
+ test ves/rt-VesDeltaF-mwalkers/ NOT APPLICABLE
+ check file ves/rt-VesDeltaF-mwalkers/report.txt for more information
+++++
+ Final report:
+ 298 tests performed, 223 tests not applicable
+ 0 errors found
+ Well done!!
+++++
make[1]: Leaving directory '/gpfs20/shared/src/plumed-2.7.1/regtest'
```

After installation, PLUMED offers a very convenient prepared module file. The file can be copied along with the rest of modulefiles with a simple edit for loading the modules used during compilation:

```
cp /shared/software/atomistic/plumed/2.7.1_gcc93_mpic341/lib/plumed/modulefile /shared/
modulefiles/tier2/atomistic/plumed/2.7.1_gcc93_mpic341
```

The only change needed for this modulefile is adding one line loading the other modules:

```
#%Module1.0#####
# Manually add here dependencies and conflicts
```

(continues on next page)

(continued from previous page)

```
module load lang/gcc/9.3.0 parallel/mpich/3.4.1_gcc93 lang/python/cpython_3.9.5_gcc93
  ↵libs/openblas/0.3.13_gcc93 libs/fftw/3.3.9_gcc93
  ...
```

6.8.2 AMBER

Amber is a suite of biomolecular simulation programs. It is used to simulate large and complex atomic compounds using a set of molecular mechanical force fields for the simulation of biomolecules.

In this document, we provide instructions to compile Amber using GCC 9.3, 11.1, and Intel Compilers 2021. Amber can be compiled with a variety of options for parallelization. Amber can be compiled as pure serial code, using multithreading with OpenMP, distributed parallelism with MPI, and using GPUs with CUDA. We will be building Amber with each option and one final compilation enabling OpenMP, MPI, and CUDA.

Amber as a package is composed of two pieces, Amber itself and AmberTools. Amber facilitates faster simulations (on parallel CPU or GPU hardware) and is distributed with a paid license. AmberTools is a free package that collect open-source code to be used in conjunction with Amber. The version used to produce these notes is Amber 20, the latest version available by mid-2021.

We start with two files, `Amber20.tar.bz2` and `AmberTools20.tar.bz2`. The first step is to uncompress both of them. The two compressed tars will uncompress on the same target folder `amber20_src`:

```
tar -jxvf Amber20.tar.bz2
tar -jxvf AmberTools20.tar.bz2
cd amber20_src
```

We start with GCC 9.3 where most of the tools can be enabled and the compilation is easier. We start with a pure Serial build, we move after to enable OpenMP, after that, we disable OpenMP and enable MPI. We move the building process to a GPU node to compile a CUDA version disabling OpenMP and MPI and we finalize enabling all options for a final build. The reason for doing this is that keeping just one parallelization active will facilitate the usage for casual users, instead of dealing with complex combinations of threads, MPI processes, and GPU cards.

GCC 9.3 (Serial)

Decompressing `Amber20.tar.bz2` and `AmberTools20.tar.bz2` will create a folder `amber20_src`. There is one file `amber20_src/build/run_cmake.sample` that we will use as a template changing it for each build from now on. It is always convenient to build the code on a folder separated from the sources. Create a folder for each build, for the serial case, we suggest:

```
cd amber20_src
mkdir build_gcc93_mpich341
```

The reason for the name is that in this folder we will compile all builds, including the MPI version using MPICH 3.4.1. Inside this folder copy the file `amber20_src/build/run_cmake.sample`. The original content of this file is:

```
#!/bin/bash

# This file gives some sample cmake invocations. You may wish to
# edit some options that are chosen here.

# For information on how to get cmake, visit this page:
# https://ambermd.org/pmwiki/pmwiki.php/Main/CMake-Quick-Start
```

(continues on next page)

(continued from previous page)

```

# For information on common options for cmake, visit this page:
# http://ambermd.org/pmwiki/pmwiki.php/Main/CMake-Common-Options

# (Note that you can change the value of CMAKE_INSTALL_PREFIX from what
# is suggested below, but it cannot coincide with the amber20_src
# folder.)

AMBER_PREFIX=$(dirname $(dirname `pwd`))

if [ `uname -s|awk '{print $1}'` = "Darwin" ]; then

# For macOS:

    if [ -x /Applications/CMake.app/Contents/bin/cmake ]; then
        cmake=/Applications/CMake.app/Contents/bin/cmake
    else
        cmake=cmake
    fi

    $cmake $AMBER_PREFIX/amber20_src \
        -DCMAKE_INSTALL_PREFIX=$AMBER_PREFIX/amber20 \
        -DCOMPILER=CLANG -DBLA_VENDOR=Apple \
        -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
        -DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
        2>&1 | tee cmake.log

else

# Assume this is Linux:

    cmake $AMBER_PREFIX/amber20_src \
        -DCMAKE_INSTALL_PREFIX=$AMBER_PREFIX/amber20 \
        -DCOMPILER=GNU \
        -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
        -DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
        2>&1 | tee cmake.log

fi

if [ ! -s cmake.log ]; then
    echo ""
    echo "Error: No cmake.log file created: you may need to edit run_cmake"
    exit 1
fi

echo ""
echo "If the cmake build report looks OK, you should now do the following:"
echo ""
echo "    make install"
echo "    source $AMBER_PREFIX/amber20/amber.sh"
echo ""

```

(continues on next page)

(continued from previous page)

```
echo "Consider adding the last line to your login startup script, e.g. ~/.bashrc"
echo ""
```

We will only make modifications to this file in two locations. We add an extra variable for defining the PREFIX:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc93_serial
```

This is the PREFIX that we will use for the Serial compilation The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DBZIP2_LIBRARIES=/shared/software/lang/gcc/9.3.0/lib/libbz2.a \
-DOPENMP=FALSE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

The lines above are for a purely serial build of Amber.

Now we need to load some modules for compiling the code. Amber uses CMAKE as a software builder. The version included with RedHat 7.x (2.18) is too old for most scientific codes. We will load modules for cmake 3.21.1 and GCC 9.3:

```
module load dev/cmake/3.21.1 lang/gcc/9.3.0 lang/python/cpython_3.9.5_gcc93
```

There is a bug in the script for Miniconda, that will prevent cmake of building a complete environment. On the line 153 of `amber20_src/cmake/UseMiniconda.cmake` add a line to install pip via conda:

```
execute_process(COMMAND ${CONDA} update conda -y)
execute_process(COMMAND ${CONDA} install pip -y)
execute_process(COMMAND ${MINICONDA PYTHON} -m pip install pip --upgrade)
```

If you do skip this step, conda will remove pip after the update and several other python packages that are installed with pip will fail. This is the only change in the sources. No other changes will be done directly to sources, if something fails, we simply disable the corresponding package. Run the `run_cmake` inside the corresponding build folder:

```
cd build_gcc93_mpich341
./run_cmake
```

When running the script, Miniconda will be downloaded and installed, this is the portion of the execution that requires internet access and cannot be executed from a GPU node. We will preserve this folder for all other builds, in particular those for CUDA that are executed inside GPU nodes with no internet accesss.

After CMAKE have prepared the folder for compilation, execute:

```
make
```

The code will compile in a few minutes. To install the build execute:

```
make install
```

GCC 9.3 (OpenMP)

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc93_openmp
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DBZIP2_LIBRARIES=/shared/software/lang/gcc/9.3.0/lib/libbz2.a \
-DOPENMP=TRUE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

The running of `make` followed by `make install` is not necessary, the later will run `make` and build the binaries before installing them.

GCC 9.3 (MPICH 3.4.1)

We will add a new module:

```
module load parallel/mpich/3.4.1_gcc93
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc93_mpich341
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DBZIP2_LIBRARIES=/shared/software/lang/gcc/9.3.0/lib/libbz2.a \
-DOPENMP=FALSE -DMPI=TRUE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

GCC 9.3 (CUDA)

This build is done on a GPU node, request an interactive execution:

```
qsub -I -l nodes=1:ppn=8:gpus=3 -q comm_gpu_inter
```

Load all the modules used in the previous builds adding the module for CUDA:

```
module load dev/cmake/3.21.1 lang/gcc/9.3.0 lang/python/cpython_3.9.5_gcc93_parallel/
˓mpich/3.4.1_gcc93_parallel/cuda/11.3
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc93_cuda113
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DBZIP2_LIBRARIES=/shared/software/lang/gcc/9.3.0/lib/libbz2.a \
-DOPENMP=FALSE -DMPI=FALSE -DCUDA=TRUE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

GCC 9.3 (OpenMP + MPI + CUDA)

This build is done on a GPU node, reuse the previous interactive session or request a new one:

```
qsub -I -l nodes=1:ppn=8:gpus=3 -q comm_gpu_inter
```

Load all the modules used in the previous builds adding the module for CUDA:

```
module load dev/cmake/3.21.1 lang/gcc/9.3.0 lang/python/cpython_3.9.5_gcc93_parallel/
˓mpich/3.4.1_gcc93_parallel/cuda/11.3
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc93_mpic341_cuda113
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DBZIP2_LIBRARIES=/shared/software/lang/gcc/9.3.0/lib/libbz2.a \
-DOPENMP=TRUE -DMPI=TRUE -DCUDA=TRUE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

Once AMBER is compiled and the modulefile created, use a GPU node to run the testsuite. There are two versions of it running AMBER the CUDA tests are either serial or parallel. Here are the results:

```
==> /shared/software/atomistic/amber/20_gcc93_mpich341_cuda/logs/test_amber_cuda/2021-05-
↳ 26_11-31-03.log <==
diffing md_SC_NVT_MBAR_SC_2.o.DPFP with md_SC_NVT_MBAR_SC_2.o
PASSED
=====
make[1]: Leaving directory '/gpfs20/shared/src/AMBER/amber20/amber20_src/test/cuda'

Finished CUDA test suite for Amber 20 at Wed May 26 11:40:28 EDT 2021.

242 file comparisons passed
7 file comparisons failed (1 of which can be ignored)
0 tests experienced errors

==> /shared/software/atomistic/amber/20_gcc93_mpich341_cuda/logs/test_amber_cuda_parallel/
↳ 2021-05-26_11-42-11.log <==
Note: The following floating-point exceptions are signalling: IEEE_DENORMAL
Note: The following floating-point exceptions are signalling: IEEE_DENORMAL
Note: The following floating-point exceptions are signalling: IEEE_DENORMAL
diffing mdout.pme.gamd3.GPU_DPFP with mdout.pme.gamd3
PASSED
=====
make[1]: Leaving directory '/gpfs20/shared/src/AMBER/amber20/amber20_src/test/cuda'
179 file comparisons passed
43 file comparisons failed (3 of which can be ignored)
2 tests experienced errors
```

GCC 11.1 (Serial)

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc111_serial
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DDISABLE_TOOLS="gbnsr6;cifparse;gbnsr6;sff" \
-DCMAKE_Fortran_FLAGS="-fallow-invalid-boz -fallow-argument-mismatch" \
-DBZIP2_LIBRARIES=/shared/software/lang/gcc/11.1.0/lib/libbz2.a \
-DOPENMP=FALSE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

GCC 11.1 (OpenMP)

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc111_openmp
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DDISABLE_TOOLS="gbnsr6;cifparse;gbnsr6;sff" \
-DCMAKE_Fortran_FLAGS="-fallow-invalid-boz -fallow-argument-mismatch" \
-DBZIP2_LIBRARIES=/shared/software/lang/gcc/11.1.0/lib/libbz2.a \
-DOPENMP=TRUE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

GCC 11.1 (OpenMPI 4.1.1)

Loading the modules:

```
module load lang/python/cpython_3.9.5_gcc111 parallel/openmpi/4.1.1_gcc111
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_gcc111_ompi411
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=GNU -DDISABLE_TOOLS="gbnsr6;cifparse;gbnsr6;sff" \
-DCMAKE_Fortran_FLAGS="-fallow-invalid-boz -fallow-argument-mismatch" \
-DBZIP2_LIBRARIES=/shared/software/lang/gcc/11.1.0/lib/libbz2.a \
-DOPENMP=FALSE -DMPI=TRUE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

Intel Compilers 2021 (Serial)

Loading the modules:

```
module load dev/cmake/3.21.1 compiler/2021.2.0 mpi/2021.2.0 mkl/2021.2.0 lang/gcc/9.3.0
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))
PREFIX=/shared/software/atomistic/amber/20_intel21_serial
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:

cmake $AMBER_PREFIX/amber20_src \
-DCMAKE_INSTALL_PREFIX=$PREFIX \
-DCOMPILER=INTEL -DDISABLE_TOOLS="reduce" \
-DBISON_EXECUTABLE=/shared/software/lang/gcc/9.3.0/bin/bison \
-DOPENMP=FALSE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake  
make install
```

Intel Compilers 2021 (OpenMP)

Loading the modules:

```
module load dev/cmake/3.21.1 compiler/2021.2.0 mpi/2021.2.0 mkl/2021.2.0 lang/gcc/9.3.0
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))  
PREFIX=/shared/software/atomistic/amber/20_intel21_openmp
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:  
  
cmake $AMBER_PREFIX/amber20_src \  
-DCMAKE_INSTALL_PREFIX=$PREFIX \  
-DCOMPILER=INTEL -DDISABLE_TOOLS="reduce" \  
-DBISON_EXECUTABLE=/shared/software/lang/gcc/9.3.0/bin/bison \  
-DOPENMP=TRUE -DMPI=FALSE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \  
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \  
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake  
make install
```

Intel Compilers 2021 (Intel MPI 2021)

Loading the modules:

```
module load dev/cmake/3.21.1 compiler/2021.2.0 mpi/2021.2.0 mkl/2021.2.0 lang/gcc/9.3.0
```

The variable defining the PREFIX will be:

```
AMBER_PREFIX=$(dirname $(dirname `pwd`))  
PREFIX=/shared/software/atomistic/amber/20_intel21_impi21
```

This is the PREFIX that we will use for the OpenMP compilation. The other change is inside the block for running cmake for the Linux build:

```
# Assume this is Linux:  
  
cmake $AMBER_PREFIX/amber20_src \  
-DCMAKE_INSTALL_PREFIX=$PREFIX \  
-
```

(continues on next page)

(continued from previous page)

```
-DCOMPILER=INTEL -DDISABLE_TOOLS="reduce" \
-DBISON_EXECUTABLE=/shared/software/lang/gcc/9.3.0/bin/bison \
-DOPENMP=FALSE -DMPI=TRUE -DCUDA=FALSE -DINSTALL_TESTS=TRUE \
-DDOWNLOAD_MINICONDA=TRUE -DMINICONDA_USE_PY3=TRUE \
2>&1 | tee cmake.log
```

After this execute:

```
./run_cmake
make install
```

Intel Compilers 2021 (CUDA)

Nvidia CUDA compiler nvcc does not support Intel Compilers 2021 as the base compiler for CUDA builds.

Running Tests

To run the testsuite, the modulefile needs to be created and loaded. The module must set the variable \$AMBERHOME needed to run the tests. Go to the folder amber20_src/test that contains the tests. For the parallel tests, set the variable \$DO_PARALLEL to the right command for running MPI executions, for example:

```
export DO_PARALLEL="mpirun -np 4"
```

These are the results of several tests:

GCC 9.3 Serial:

```
$> ./test_amber_serial.sh
...
...
Finished serial test suite for Amber 20 at Sat Aug 21 17:52:10 EDT 2021.

→           196 file comparisons passed
0 file comparisons failed
0 tests experienced errors
Test log file saved as /shared/software/atomistic/amber/20_gcc93_serial/logs/test_amber_
→_serial/2021-08-21_17-45-18.log
No test diffs to save!
```

GCC 9.3 OpenMP:

```
$> ./test_amber_serial.sh
...
...
Finished serial test suite for Amber 20 at Sat Aug 21 18:04:09 EDT 2021.

196 file comparisons passed
0 file comparisons failed
0 tests experienced errors
```

(continues on next page)

(continued from previous page)

```
Test log file saved as /shared/software/atomistic/amber/20_gcc93_openmp/logs/test_amber_
˓→serial/2021-08-21_17-57-23.log
No test diffs to save!
```

GCC 9.3 CUDA:

```
$> ./test_amber_cuda_serial.sh
...
...
Finished CUDA test suite for Amber 20 at Sat Aug 21 18:15:52 EDT 2021.

243 file comparisons passed
6 file comparisons failed (1 of which can be ignored)
0 tests experienced errors
Test log file saved as /shared/software/atomistic/amber/20_gcc93_cuda113/logs/test_amber_
˓→cuda/2021-08-21_18-12-26.log
Test diffs file saved as /shared/software/atomistic/amber/20_gcc93_cuda113/logs/test_amber_
˓→cuda/2021-08-21_18-12-26.diff
```

GCC 9.3 MPICH 3.4.1:

```
$> export DO_PARALLEL="mpirun -np 8"
$> ./test_amber_parallel.sh
...
...
Finished parallel test suite for Amber 20 on Sat Aug 21 18:31:15 EDT 2021.
Some tests require 4 threads to run, while some will not
run with more than 2. Please run further parallel tests with the appropriate number of
˓→processors. See /shared/software/atomistic/amber/20_gcc93_mpich341/test/README.

274 file comparisons passed
4 file comparisons failed (1 of which can be ignored)
0 tests experienced an error
Test log file saved as /shared/software/atomistic/amber/20_gcc93_mpich341/logs/test_amber_
˓→parallel/2021-08-21_18-27-41.log
Test diffs file saved as /shared/software/atomistic/amber/20_gcc93_mpich341/logs/test_amber_
˓→parallel/2021-08-21_18-27-41.diff
```

GCC 9.3 OpenMP + MPI + CUDA:

```
$> cat $PBS_NODEFILE | uniq > NODEFILE
$> export DO_PARALLEL="mpirun -np 2 -f /shared/src/AMBER/amber20/amber20_src/test/
˓→NODEFILE"
$> ./test_amber_cuda_parallel.sh
...
...
135 file comparisons passed
13 file comparisons failed (1 of which can be ignored)
0 tests experienced errors
Test log file saved as /shared/software/atomistic/amber/20_gcc93_mpich341_cuda113/logs/
˓→test_amber_cuda_parallel/2021-08-21_19-27-18.log
Test diffs file saved as /shared/software/atomistic/amber/20_gcc93_mpich341_cuda113/logs/
˓→test_amber_cuda_parallel/2021-08-21_19-27-18.diff
```

6.8.3 Gromacs

Gromacs is a Classical Molecular Dynamics code. The version compiled was 2021.2 Several versions were compiled using GCC 9.3 and 11.1

Gromacs 2021.2 on Thorny Flat

The download page is:

```
https://manual.gromacs.org
```

It is a good practice to compile from a separate folder instead of compiling directly alongside with the sources, create a folder build_gcc93_mpich341 inside the sources:

```
wget https://ftp.gromacs.org/gromacs/gromacs-2021.2.tar.gz
tar -zxf gromacs-2021.2.tar.gz
cd gromacs-2021.2/
mkdir build_gcc93_mpich341
cd build_gcc93_mpich341
```

Cmake is only used during configuration and it is not needed at runtime The modules can be loaded with this command line:

```
module purge
module load lang/gcc/9.3.0 parallel/mpich/3.4.1_gcc93 dev/cmake/3.18.3 \
lang/python/cpython_3.9.5_gcc93 libs/openblas/0.3.13_gcc93
```

The first configuration is the standard one (Single Precision)

The cmake configuration line was:

```
cmake -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=on \
-DCMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx \
-DGMX_LAPACK_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DGMX_BLAS_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DCMAKE_INSTALL_PREFIX=/shared/software/atomistic/gromacs/2021.2_gcc93_mpich341 ..
```

The results of the tests were:

```
...
...
70/73 Test #70: regressiontests/complex ..... Passed 135.38 sec
  Start 71: regressiontests/freeenergy
71/73 Test #71: regressiontests/freeenergy ..... Passed 34.09 sec
  Start 72: regressiontests/rotation
72/73 Test #72: regressiontests/rotation ..... Passed 28.83 sec
  Start 73: regressiontests/essentialdynamics
73/73 Test #73: regressiontests/essentialdynamics ..... Passed 10.23 sec

100% tests passed, 0 tests failed out of 73

Label Time Summary:
GTest          = 108.89 sec*proc (67 tests)
IntegrationTest = 32.96 sec*proc (20 tests)
```

(continues on next page)

(continued from previous page)

```
MpiTest      =  52.50 sec*proc (10 tests)
SlowTest     =  57.26 sec*proc (8 tests)
UnitTest     =  18.67 sec*proc (39 tests)

Total Test time (real) = 317.75 sec
[100%] Built target run-ctest-nophys
Scanning dependencies of target check
[100%] Built target check
```

The second configuration enables the double precision for gromacs:

The cmake configuration line was:

```
cmake -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=on -DCMAKE_C_
-COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx -DCMAKE_INSTALL_PREFIX=/shared/software/
-atomistic/gromacs/2021.2_double_gcc93_mpicc341 -DGMX_LAPACK_USER="-L${MD_OPENBLAS}/lib -
-lopenblas" -DGMX_BLAS_USER="-L${MD_OPENBLAS}/lib -lopenblas" -DGMX_DOUBLE=on ..
```

The results of the tests were:

```
98% tests passed, 1 tests failed out of 46

Label Time Summary:
GTest          = 117.18 sec*proc (40 tests)
IntegrationTest = 13.81 sec*proc (5 tests)
MpiTest        = 2.60 sec*proc (3 tests)
SlowTest        = 12.88 sec*proc (1 test)
UnitTest        = 90.49 sec*proc (34 tests)

Total Test time (real) = 2075.93 sec
```

Gromacs 5.1.5 on Thorny Flat

The modules used were:

```
module load lang/intel/2018 dev/cmake/3.18.3 libs/boost/1.73
```

Cmake is only used during configuration and it is not needed at runtime. The configuration line for cmake is executed on a folder created to contain the compiled code:

```
mkdir build_intel18
cd build_intel18
cmake -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=on \
-DCMAKE_C_COMPILER=mpiicc -DCMAKE_CXX_COMPILER=mpiicpc \
-DCMAKE_INSTALL_PREFIX=/shared/software/atomistic/gromacs/5.1.5_intel18 ..
make -j12
make check
make install
```

A similar compilation was done using Intel 2019 compilers. One test fails from the test suite:

```
96% tests passed, 1 tests failed out of 26

Label Time Summary:
GTest           = 2.29 sec*proc (17 tests)
IntegrationTest = 2.01 sec*proc (2 tests)
MpiIntegrationTest = 0.56 sec*proc (1 test)
UnitTest        = 2.29 sec*proc (17 tests)
Total Test time (real) = 120.90 secs.

The following tests FAILED:
  17 - SelectionUnitTests (Failed)
```

Gromacs 2021.3 on Spruce Knob

The modules used were:

```
$> module purge
$> module load lang/gcc/9.3.0 parallel/mpich/3.3.2_gcc93 dev/cmake/3.15.2 lang/python/
  ↵ cpython_3.9.7_gcc93 libs/openblas/0.3.10_gcc93
```

Download and uncompress the sources:

```
$> wget https://ftp.gromacs.org/gromacs/gromacs-2021.3.tar.gz
$> tar -zvxf gromacs-2021.3.tar.gz
$> cd gromacs-2021.3
$> mkdir build_gcc93
$> cd build_gcc93
```

Cmake is only used during configuration and it is not needed at runtime

The first configuration is the standard one (Single Precision)

The cmake configuration line was:

```
cmake -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=on \
-DCKMAKE_C_COMPILER=mpicc -DCMAKE_CXX_COMPILER=mpicxx \
-DGMX_LAPACK_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DGMX_BLAS_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DCMAKE_INSTALL_PREFIX=/shared/software/atomistic/gromacs/2021.3_gcc93_mpicc32 ..
```

After configuration compile and install the code:

```
$> make
$> make install
```

The results of the tests were:

```
$> make check

67/73 Test #67: GmxapiMPITests ..... Passed    2.46 sec
          Start 68: GmxapiInternalInterfaceTests
68/73 Test #68: GmxapiInternalInterfaceTests ..... Passed    0.76 sec
          Start 69: GmxapiInternalsMPITests
```

(continues on next page)

(continued from previous page)

69/73 Test #69: GmxapiInternalsMPI Tests	Passed	0.86 sec
Start 70: regressiontests/complex		
70/73 Test #70: regressiontests/complex	Passed	45.75 sec
Start 71: regressiontests/freeenergy		
71/73 Test #71: regressiontests/freeenergy	Passed	14.99 sec
Start 72: regressiontests/rotation		
72/73 Test #72: regressiontests/rotation	Passed	10.50 sec
Start 73: regressiontests/essentialdynamics		
73/73 Test #73: regressiontests/essentialdynamics	Passed	4.22 sec

100% tests passed, 0 tests failed out of 73

Label Time Summary:

GTest	= 105.49 sec*proc (67 tests)
IntegrationTest	= 22.65 sec*proc (20 tests)
MpiTest	= 68.16 sec*proc (10 tests)
SlowTest	= 79.18 sec*proc (8 tests)
UnitTest	= 3.66 sec*proc (39 tests)

Total Test time (real) = 181.11 sec

```
make[3]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_gcc93'
[100%] Built target run-ctest-nophys
make[3]: Entering directory '/gpfs/shared/src/gromacs-2021.3/build_gcc93'
Scanning dependencies of target check
make[3]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_gcc93'
[100%] Built target check
make[2]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_gcc93'
make[1]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_gcc93'
```

The second configuration enables double precision floating point numbers. The cmake configuration line was:

```
cmake -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=on \
-DMAKE_C_COMPILER=mpicc -DMAKE_CXX_COMPILER=mpicxx \
-DGMX_LAPACK_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DGMX_BLAS_USER="-L${MD_OPENBLAS}/lib -lopenblas" \
-DMAKE_INSTALL_PREFIX=/shared/software/atomistic/gromacs/2021.3_double_gcc93_mpic332 \
-DGMX_DOUBLE=on ...
```

The results of the tests were:

\$> make check		
Start 65: MdrunSimulatorComparison		
65/73 Test #65: MdrunSimulatorComparison	Passed	5.42 sec
Start 66: GmxapiExternalInterfaceTests		
66/73 Test #66: GmxapiExternalInterfaceTests	Passed	3.32 sec
Start 67: GmxapiMPITests		
67/73 Test #67: GmxapiMPITests	Passed	3.19 sec
Start 68: GmxapiInternalInterfaceTests		
68/73 Test #68: GmxapiInternalInterfaceTests	Passed	0.98 sec
Start 69: GmxapiInternalsMPITests		
69/73 Test #69: GmxapiInternalsMPITests	Passed	1.42 sec

(continues on next page)

(continued from previous page)

```

Start 70: regressiontests/complex
70/73 Test #70: regressiontests/complex ..... Passed 49.69 sec
    Start 71: regressiontests/freeenergy
71/73 Test #71: regressiontests/freeenergy ..... Passed 16.65 sec
    Start 72: regressiontests/rotation
72/73 Test #72: regressiontests/rotation ..... Passed 12.70 sec
    Start 73: regressiontests/essentialdynamics
73/73 Test #73: regressiontests/essentialdynamics ..... Passed 4.17 sec

```

100% tests passed, 0 tests failed out of 73

Label Time Summary:

GTest	= 138.11 sec*proc (67 tests)
IntegrationTest	= 34.03 sec*proc (20 tests)
MpiTest	= 89.03 sec*proc (10 tests)
SlowTest	= 100.02 sec*proc (8 tests)
UnitTest	= 4.06 sec*proc (39 tests)

Total Test time (real) = 221.48 sec

```

make[3]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_double_gcc93'
[100%] Built target run-ctest-nophys
make[3]: Entering directory '/gpfs/shared/src/gromacs-2021.3/build_double_gcc93'
Scanning dependencies of target check
make[3]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_double_gcc93'
[100%] Built target check
make[2]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_double_gcc93'
make[1]: Leaving directory '/gpfs/shared/src/gromacs-2021.3/build_double_gcc93'

```

6.8.4 LAMMPS

LAMMPS is a packages for atomistic and particle simulations. The latest stable version by the time (May 2021) is from October 29, 2020. LAMMPS was compiled using these modules:

```

lang/gcc/11.1.0
parallel/openmpi/3.1.6_gcc111
libs/fftw/3.3.9_gcc111
libs/hdf5/1.12.0_gcc111

```

LAMMPS was compiled using GCC 11.1, OpenMPI 3.4.1, FFTW 3.3.9, and HDF5 1.12

The first step is to download the code from:

```
wget https://lammps.sandia.gov/tars/lammps-29Oct20.tar.gz
```

Uncompress the code:

```
tar -zxf lammps-29Oct20.tar.gz
```

Change to the src folder inside the uncompressed folder:

```
cd lammps-29Oct20/src
```

You need a customized Makefile for compiling LAMMPS with the right compilers and libraries. The file is called Makefile.gcc111_ompi316 and must be located at `src/MAKE`, the content of the file follows:

```
# mpi = MPI with its default compiler

SHELL = /bin/sh

# -----
# compiler/linker settings
# specify flags and libraries needed for your compiler

CC = mpicxx
CFLAGS = -g -O3
SHFLAGS = -fPIC
DEPFLAGS = -M

LINK = mpicxx
LINKFLAGS = -g -O3
LIB =
SIZE = size

ARCHIVE = ar
ARFLAGS = -rc
SHLIBFLAGS = -shared

# -----
# LAMMPS-specific settings, all OPTIONAL
# specify settings for LAMMPS features you will use
# if you change any -D setting, do full re-compile after "make clean"

# LAMMPS ifdef settings
# see possible settings in Section 3.5 of the manual

LMP_INC = -DLAMMPS_GZIP -DLAMMPS_MEMALIGN=64 # -DLAMMPS_CXX98

# MPI library
# see discussion in Section 3.4 of the manual
# MPI wrapper compiler/linker can provide this info
# can point to dummy MPI library in src/STUBS as in Makefile.serial
# use -D MPICH and OMPI settings in INC to avoid C++ lib conflicts
# INC = path for mpi.h, MPI compiler settings
# PATH = path for MPI library
# LIB = name of MPI library

MPI_INC = -DMPICH_SKIP_MPICXX -DOMPI_SKIP_MPICXX=1
MPI_PATH =
MPI_LIB =

# FFT library
# see discussion in Section 3.5.2 of manual
# can be left blank to use provided KISS FFT library
# INC = -DFFT setting, e.g. -DFFT_FFTW, FFT compiler settings
# PATH = path for FFT library
```

(continues on next page)

(continued from previous page)

```

# LIB = name of FFT library

FFT_INC = -DFFT_FFTW3
FFT_PATH =
FFT_LIB = -L${MD_FFTW}/lib -lfftw3

# JPEG and/or PNG library
# see discussion in Section 3.5.4 of manual
# only needed if -DLAMMPS_JPEG or -DLAMMPS_PNG listed with LMP_INC
# INC = path(s) for jpeglib.h and/or png.h
# PATH = path(s) for JPEG library and/or PNG library
# LIB = name(s) of JPEG library and/or PNG library

JPG_INC = -I${MD_GCC}/include
JPG_PATH = -L${MD_GCC}/lib
JPG_LIB = -lpng -ljpeg -lz

# -----
# build rules and dependencies
# do not edit this section

include Makefile.package.settings
include Makefile.package

EXTRA_INC = $(LMP_INC) $(PKG_INC) $(MPI_INC) $(FFT_INC) $(JPG_INC) $(PKG_SYSINC)
EXTRA_PATH = $(PKG_PATH) $(MPI_PATH) $(FFT_PATH) $(JPG_PATH) $(PKG_SYSPATH)
EXTRA_LIB = $(PKG_LIB) $(MPI_LIB) $(FFT_LIB) $(JPG_LIB) $(PKG_SYSLIB)
EXTRA_CPP_DEPENDS = $(PKG_CPP_DEPENDS)
EXTRA_LINK_DEPENDS = $(PKG_LINK_DEPENDS)

# Path to src files

vpath %.cpp ..
vpath %.h ..

# Link target

$(EXE): main.o $(LMPLIB) $(EXTRA_LINK_DEPENDS)
        $(LINK) $(LINKFLAGS) main.o $(EXTRA_PATH) $(LMPLINK) $(EXTRA_LIB) $(LIB) -o $@
        $(SIZE) $@

# Library targets

$(ARLIB): $(OBJ) $(EXTRA_LINK_DEPENDS)
        @rm -f ../$ARLIB
        $(ARCHIVE) $(ARFLAGS) ../$ARLIB $(OBJ)
        @rm -f $(ARLIB)
        @ln -s ../$ARLIB $(ARLIB)

$(SHLIB): $(OBJ) $(EXTRA_LINK_DEPENDS)
        $(CC) $(CCFLAGS) $(SHFLAGS) $(SHLIBFLAGS) $(EXTRA_PATH) -o ../$SHLIB \
        $(OBJ) $(EXTRA_LIB) $(LIB)

```

(continues on next page)

(continued from previous page)

```

@rm -f $(SHLIB)
@ln -s ../$(SHLIB) $(SHLIB)

# Compilation rules

%.o:%.cpp
    $(CC) $(CCFLAGS) $(SHFLAGS) $(EXTRA_INC) -c $<

# Individual dependencies

depend : fastdep.exe $(SRC)
    ./fastdep.exe $(EXTRA_INC) -- $^ > .depend || exit 1

fastdep.exe: ../DEPEND/fastdep.c
    cc -O -o $@ $<

sinclue .depend

```

The file should be located inside “src/MAKE” or “src/MAKE/MACHINES”. Now inside the “src” folder, there is a Makefile that allow you to select which packages will be compiled along side LAMMPS. A good selection comes from adding all followed by removing those depend on libraries and after adding a few:

```

make yes-all
make no-lib
make yes-user-reaxc
make yes-user-molfile

```

A few external subpackages must be configured first. We want to add HDF5 and COLVARS with the following lines compiling the corresponding subpackages and enabling them for LAMMPS:

```

cd ../lib/h5md
make -f Makefile.h5cc
cd ../../src/
make yes-user-h5md
make lib-colvars args="-m mpi"
make yes-user-colvars

```

And LAMMPS itself after that:

```
make gcc111_ompi316
```

After compiled the binary is called lmp_gcc82_ompi31

For testing the build, you can use one of the benchmarks inside the *bench* folder. The benchmark runs on one of the compute nodes using 40 cores. The simulation involves more than 10 million atoms. The command line is:

```
mpirun -np 40 lmp_mpi -var x 4 -var y 8 -var z 10 -in in.rhodo.scaled
```

This is the final output:

```
Loop time of 303.71 on 40 procs for 100 steps with 10240000 atoms
```

```
Performance: 0.057 ns/day, 421.820 hours/ns, 0.329 timesteps/s
```

(continues on next page)

(continued from previous page)

```
99.1% CPU use with 40 MPI tasks x no OpenMP threads
```

MPI task timing breakdown:

Section	min time	avg time	max time	%varavg	%total
Pair	201.28	205.41	210.7	15.7	67.63
Bond	9.4133	9.5518	9.7418	2.2	3.15
Kspace	31.517	36.816	40.964	36.8	12.12
Neigh	36.55	36.561	36.571	0.1	12.04
Comm	1.3529	1.5481	1.6872	7.0	0.51
Output	0.0038965	0.0040967	0.0043541	0.1	0.00
Modify	12.757	13.067	13.53	5.7	4.30
Other		0.7489			0.25
Nlocal:	256000.0	ave 256004	max 255996	min	
Histogram:	8 0 0 0 0 24 0 0 0 8				
Nghost:	163342.0	ave 163347	max 163335	min	
Histogram:	8 0 8 0 0 0 8 0 0 16				
Neighs:	9.62247e+07	ave 9.65195e+07	max 9.59192e+07	min	
Histogram:	4 4 0 8 0 8 8 0 4 4				
Total # of neighbors =	3.8489892e+09				
Ave neighs/atom =	375.87785				
Ave special neighs/atom =	7.4318750				
Neighbor list builds =	11				
Dangerous builds =	0				
Total wall time:	0:05:13				

The table below shows the timings using the different builds created.

Module	Total wall time
atomistic/lammps/2020.10.29_gcc111_impi19 atomistic/lammps/2020.10.29_gcc111_ompi316	0:04:54 0:05:13
atomistic/lammps/2020.10.29_gcc93_mpic341	0:05:11

6.9 CHARM++ and NAMD

NAMD is a computer software for classical molecular dynamics simulation, written using the Charm++ parallel programming model. It is noted for its parallel efficiency and is often used to simulate large systems (millions of atoms). It has been developed by the collaboration of the Theoretical and Computational Biophysics Group (TCB) and the Parallel Programming Laboratory (PPL) at the University of Illinois at Urbana–Champaign.

The purpose of this document is to show how Charm++ and NAMD were compiled on Thorny Flat, a cluster build from Skylake CPUs. The current stable sources 2.13 fail compilation using OFI network library. For that reason we will be using the Nightly archived sources available on the official NAMD site.

The sources can be downloaded for the most recent date at the moment of download. The sources can be download using this command:

```
VERSION=2021-10-05
wget https://www.ks.uiuc.edu/Research/namd/cvs/download/741376/NAMD_Git-${VERSION}_
↪Source.tar.gz
```

This is the current Nightly build by the time of writing this document (2020-01-02). The nightly build can be downloaded from the browser on <https://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD>

The sources are archived in a tar and compressed file. Untar and uncompress them. Go into the created folder and untar the charm sources too:

```
tar -zvxf NAMD_Git-${VERSION}_Source.tar.gz
cd NAMD_Git-${VERSION}_Source/
tar -xvf charm-6.10.2.tar
```

We will compile Charm and NAMD using the Intel Compilers 2021, the latest version at the moment of writing this document:

```
module purge
module load compiler/2021.2.0 mkl/2021.2.0 mpi/2021.2.0
```

6.9.1 Compiling Charm++

We start compiling charm++, go into the charm-6.10.2 folder:

```
cd charm-6.10.2
```

We will compile 4 versions. The MPI version and the OFI version for both single threaded and shared memory versions:

```
MPICXX=mpiicpc ./build charm++ mpi-linux-x86_64 mpicxx ifort -j12 --with-production
MPICXX=mpiicpc ./build charm++ mpi-linux-x86_64 mpicxx ifort smp -j12 --with-production
./build charm++ ofi-linux-x86_64 icc ifort -j12 --with-production
./build charm++ ofi-linux-x86_64 icc ifort smp -j12 --with-production
```

Executing those 4 commands will compile charm++ in a variety of ways that take advantage of Intel Omni-Path. The `smp` version takes one extra thread for managing communication reducing the amount of usable cores but allows to share memory efficiently allowing for computations with larger systems. That is why having the `smp` and `non-smp` versions is recommended.

You can test those 4 compilations of charm++ with the testsuite provided with the sources. Notice that before each new tests the previous binaries must be deleted. The following commands will perform the 4 tests suite runs:

```
cd mpi-linux-x86_64-ifort-mpicxx/tests/charm++
make clean && make && make test
cd ../../..
cd mpi-linux-x86_64-ifort-smp-mpicxx/tests/charm++
make clean && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-smp-icc/tests/charm++
make clean && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-icc/tests/charm++
make clean && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-smp-icc/tests/charm++
make clean && make
```

(continues on next page)

(continued from previous page)

```
cd megatest && make && make test
cd ../../..
```

The last case is a bit different as one test fail with smp. We should now return to the NAMD sources to continue the compilation:

```
cd ..
```

6.9.2 Compiling NAMD2

Now we proceed to compile NAMD. Download and install TCL and FFTW libraries assuming that you are now on the root folder for NAMD sources:

```
wget http://www.ks.uiuc.edu/Research/namd/libraries/fftw-linux-x86_64.tar.gz
tar xzf fftw-linux-x86_64.tar.gz
mv linux-x86_64 fftw
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64.tar.gz
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64-threaded.tar.gz
tar xzf tcl8.5.9-linux-x86_64.tar.gz
tar xzf tcl8.5.9-linux-x86_64-threaded.tar.gz
mv tcl8.5.9-linux-x86_64 tcl
mv tcl8.5.9-linux-x86_64-threaded tcl-threaded
```

The next step is to edit the file `Make.charm` to edit the variable `CHARMBASE`. Another option is to create a symbolic link called `charm` pointing to the location of the charm sources, like this:

```
ln -s charm-6.10.2 charm
```

The configuration of NAMD is done via a text file located at the `arch` folder. Create the following files with the commands:

```
cat << EOF > arch/Linux-x86_64-mpi-mpicxx.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$FLOATOPTS
CXXNOALIASOPTS = -O2 -fno-alias \$FLOATOPTS
CXXCOLVAROPTS = -O2 -ip

CC =icc
COPTS = -O2 \$FLOATOPTS
EOF

cat << EOF > arch/Linux-x86_64-mpi-smp-mpicxx.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-smp-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd
```

(continues on next page)

(continued from previous page)

```
CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

cat << EOF > arch/Linux-x86_64-ofi-icc.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

cat << EOF > arch/Linux-x86_64-ofi-smp-icc.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-smp-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF
```

Executing the code above will produce 4 files with the following contents.

File `Linux-x86_64-mpi-mpicxx.arch`:

```
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias $(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip
```

(continues on next page)

(continued from previous page)

```
CC =icc
COPTS = -O2 $(FLOATOPTS)
```

File Linux-x86_64-mpi-smp-mpicxx.arch:

```
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-smp-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias $(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC =icc
COPTS = -O2 $(FLOATOPTS)
```

File Linux-x86_64-ofi-icc.arch:

```
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias $(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC =icc
COPTS = -O2 $(FLOATOPTS)
```

File Linux-x86_64-ofi-smp-icc.arch:

```
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-smp-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias $(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC =icc
COPTS = -O2 $(FLOATOPTS)
```

To compile NAMD, the corresponding building folder must be created via the config command. The following commands will create 4 folders for the corresponding versions of charm++ that we will use:

```
./config Linux-x86_64-mpi-mpicxx --charm-arch mpi-linux-x86_64-ifort-mpicxx
./config Linux-x86_64-mpi-smp-mpicxx --charm-arch mpi-linux-x86_64-ifort-smp-mpicxx
./config Linux-x86_64-ofi-icc --charm-arch ofi-linux-x86_64-ifort-icc
./config Linux-x86_64-ofi-smp-icc --charm-arch ofi-linux-x86_64-ifort-smp-icc
```

Now we can go inside each folder and compile the code with `make`. To speed up the compilation, 12 execution lines will be used:

```
cd Linux-x86_64-mpi-mpicxx
make -j12
cd ..
cd Linux-x86_64-mpi-smp-mpicxx
make -j12
cd ..
cd Linux-x86_64-ofi-icc
make -j12
cd ..
cd Linux-x86_64-ofi-smp-icc
make -j12
cd ..
```

At the end of those compilations we will have 4 versions of the command `namd2`. However, due to a bug on Intel's `opa-psm2` the NAMD binaries will return an error when executed. The error looks similar to this:

```
hfi_userinit: mmap of status page (dabbad0008030000) failed: Operation not permitted
```

For the particular case of Thorny, executing NAMD will return (MPI version):

```
trcis001.hpc.wvu.edu.26685hfi_userinit: mmap of status page (dabbad00080b0000) failed:  
↳ Operation not permitted
trcis001.hpc.wvu.edu.26685hfp_gen1_context_open: hfi_userinit: failed, trying again (1/3)
trcis001.hpc.wvu.edu.26685hfi_userinit: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26685hfp_gen1_context_open: hfi_userinit: failed, trying again (2/3)
trcis001.hpc.wvu.edu.26685hfi_userinit: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26685hfp_gen1_context_open: hfi_userinit: failed, trying again (3/3)
trcis001.hpc.wvu.edu.26685hfi_userinit: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26685PSM2 can't open hfi unit: -1 (err=23)
Abort(1615759) on node 0 (rank 0 in comm 0): Fatal error in PMPI_Init_thread: Other MPI  
↳ error, error stack:
MPIR_Init_thread(703).....
MPI_D_Init(923).....
MPI_D_OFI_mpi_init_hook(1211):
create_endpoint(1892)....: OFI endpoint open failed (ofi_init.c:1892:create_
↳ endpoint:Invalid argument)
```

Or (OFI version):

```
Charm++>ofi> provider: psm2
Charm++>ofi> control progress: 2
Charm++>ofi> data progress: 2
Charm++>ofi> maximum inject message size: 64
Charm++>ofi> eager maximum message size: 65536 (maximum header size: 40)
Charm++>ofi> cq entries count: 8
```

(continues on next page)

(continued from previous page)

```

Charm++>ofi> use inject: 1
Charm++>ofi> maximum rma size: 4294967295
Charm++>ofi> mr mode: 0x1
Charm++>ofi> use memory pool: 0
trcis001.hpc.wvu.edu.26858hfi_userinit: mmap of status page (dabbad00080b0000) failed: ↵
  ↵Operation not permitted
trcis001.hpc.wvu.edu.26858hfp_gen1_context_open: hfi_userinit: failed, trying again (1/3)
trcis001.hpc.wvu.edu.26858hfp_gen1_context_open: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26858hfp_gen1_context_open: hfi_userinit: failed, trying again (2/3)
trcis001.hpc.wvu.edu.26858hfp_gen1_context_open: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26858hfp_gen1_context_open: hfi_userinit: failed, trying again (3/3)
trcis001.hpc.wvu.edu.26858hfi_userinit: assign_context command failed: Invalid argument
trcis001.hpc.wvu.edu.26858PSM2 can't open hfi unit: -1 (err=23)
----- Partition 0 Processor 0 Exiting: Called CmiAbort -----
Reason: OFI::LrtsInit::fi_endpoint error
[0] Stack Traceback:
[0:0] namd2 0x1126347 CmiAbortHelper(char const*, char const*, char const*, int, int)
[0:1] namd2 0x11262e7 CmiAbort
[0:2] namd2 0x1125088 LrtsInit(int*, char***, int*, int*)
[0:3] namd2 0x112664a ConverseInit
[0:4] namd2 0x68e302 BackEnd::init(int, char**)
[0:5] namd2 0x68332c main
[0:6] libc.so.6 0x7fbe439b53d5 __libc_start_main
[0:7] namd2 0x5d9ab9

```

The issue is related to the execute bit being set in the GNU_STACK of the ELF headers in a binary. That in turn attempts to map the memory region with both the read and execute bits enabled, rather than just the read bit as PSM2 is requesting. As described in this post:

<https://stackoverflow.com/questions/32730643/why-in-mmap-prot-read-equals-prot-exec>

And the solution was posted here:

<https://github.com/intel/opa-psm2/issues/29>

One can inspect a binary for this setting using readelf:

```
readelf --program-headers ./namd2
```

The output from that command will show this for the GNU_STACK:

GNU_STACK	0x0000000000000000 0x0000000000000000 0x0000000000000000	0x0000000000000000 0x0000000000000000 RWE 10
-----------	--	--

This issue can be fixed over the binaries already created by executing:

```
execstack -c ./namd2
```

From the NAMD source folder the following command will fix that for the 4 binaries:

```
execstack -c Linux-x86_64-* /namd2
```

6.9.3 Quick test NAMD2 for Alanin

Now we can start testing the 4 binaries of namd2. NAMD offers a very small case for testing on `src/alanin`. Execute NAMD on each folder to test the binary. Notice that for the SMP binaries a couple of extra arguments are needed:

```
cd Linux-x86_64-mpi-mpicxx
./charmrun ++local +p2 ./namd2 src/alanin
cd ..
cd Linux-x86_64-ofi-icc
./charmrun ++local +p2 ./namd2 src/alanin
cd ..
cd Linux-x86_64-mpi-smp-mpicxx
./charmrun ++local +p2 ./namd2 src/alanin ++ppn2 +setcpuaffinity
cd ..
cd Linux-x86_64-ofi-smp-icc
./charmrun ++local +p2 ./namd2 src/alanin ++ppn2 +setcpuaffinity
cd ..
```

The MPI-based non-SMP binary is executed as follows:

```
$ ./charmrun ++local +p2 ./namd2 src/alanin

Running on 2 processors: ./namd2 src/alanin
charmrun> /bin/setarch x86_64 -R mpirun -np 2 ./namd2 src/alanin
Charm++> Running on MPI version: 3.1
Charm++> level of thread support used: MPI_THREAD_SINGLE (desired: MPI_THREAD_SINGLE)
Charm++> Running in non-SMP mode: 2 processes (PEs)
Charm++> Using recursive bisection (scheme 3) for topology aware partitions
Converse/Charm++ Commit ID: v6.10.2-0-g7bf00fa-namd-charm-6.10.2-build-2020-Aug-05-556
CharmLB> Load balancer assumes all CPUs are same.
Charm++> Running on 1 hosts (2 sockets x 12 cores x 2 PUs = 48-way SMP)
Charm++> cpu topology info is gathered in 0.001 seconds.
Info: NAMD Git-2021-10-05 for Linux-x86_64-MPI
...
```

The OFI non-SMP binaries can be tested in a similar way:

```
$ ./charmrun ++local +p2 ./namd2 src/alanin

Running on 2 processors: ./namd2 src/alanin
charmrun> /bin/setarch x86_64 -R mpirun -np 2 ./namd2 src/alanin
Charm++>ofi> provider: psm2
Charm++>ofi> control progress: 2
Charm++>ofi> data progress: 2
Charm++>ofi> maximum inject message size: 64
Charm++>ofi> eager maximum message size: 65536 (maximum header size: 40)
Charm++>ofi> cq entries count: 8
Charm++>ofi> use inject: 1
Charm++>ofi> maximum rma size: 4294963200
Charm++>ofi> mr mode: 0x1
Charm++>ofi> use memory pool: 0
Charm++>ofi> use request cache: 0
Charm++>ofi> number of pre-allocated recvs: 8
Charm++>ofi> exchanging addresses over OFI
```

(continues on next page)

(continued from previous page)

```

Charm++> Running in non-SMP mode: 2 processes (PEs)
Charm++> Using recursive bisection (scheme 3) for topology aware partitions
Converse/Charm++ Commit ID: v6.10.2-0-g7bf00fa-namd-charm-6.10.2-build-2020-Aug-05-556
CharmLB> Load balancer assumes all CPUs are same.
Charm++> Running on 1 hosts (2 sockets x 12 cores x 2 PUs = 48-way SMP)
Charm++> cpu topology info is gathered in 0.001 seconds.
Info: NAMD Git-2021-10-05 for Linux-x86_64-ofi
...

```

The SMP binaries are special in the arguments needed to run. The binary at Linux-x86_64-mpi-smp-mpicxx needs at least an extra argument `++ppn`:

```

$ ./charmrun ++local +p2 ./namd2 src/alanin ++ppn2

Running on 1 processors: ./namd2 src/alanin ++ppn2
charmrun> /bin/setarch x86_64 -R mpirun -np 1 ./namd2 src/alanin ++ppn2
Charm++> Running on MPI version: 3.1
Charm++> level of thread support used: MPI_THREAD_FUNNELED (desired: MPI_THREAD_FUNNELED)
Charm++> Running in SMP mode: 1 processes, 2 worker threads (PEs) + 1 comm threads per_
↪process, 2 PEs total
Charm++> The comm. thread both sends and receives messages
Charm++> Using recursive bisection (scheme 3) for topology aware partitions
Converse/Charm++ Commit ID: v6.10.2-0-g7bf00fa-namd-charm-6.10.2-build-2020-Aug-05-556
Charm++ communication thread will sleep due to single-process run.
CharmLB> Load balancer assumes all CPUs are same.
Charm++> Running on 1 hosts (2 sockets x 12 cores x 2 PUs = 48-way SMP)
Charm++> cpu topology info is gathered in 0.001 seconds.
Info: NAMD Git-2021-10-05 for Linux-x86_64-MPI-smp
...

```

The OFI SMP binaries Linux-x86_64-ofi-smp-icc needs `+setcpuaffinity` because at least one thread for communication and that will oversubscribe the number of worker processes plus one communication thread:

```

$ ./charmrun ++local +p2 ./namd2 src/alanin ++ppn2 +setcpuaffinity

Running on 1 processors: ./namd2 src/alanin ++ppn2 +setcpuaffinity
charmrun> /bin/setarch x86_64 -R mpirun -np 1 ./namd2 src/alanin ++ppn2_
↪+setcpuaffinity
Charm++>ofi> provider: psm2
Charm++>ofi> control progress: 2
Charm++>ofi> data progress: 2
Charm++>ofi> maximum inject message size: 64
Charm++>ofi> eager maximum message size: 65536 (maximum header size: 40)
Charm++>ofi> cq entries count: 8
Charm++>ofi> use inject: 1
Charm++>ofi> maximum rma size: 4294963200
Charm++>ofi> mr mode: 0x1
Charm++>ofi> use memory pool: 0
Charm++>ofi> use request cache: 0
Charm++>ofi> number of pre-allocated recv: 8
Charm++>ofi> exchanging addresses over OFI
Charm++> Running in SMP mode: 1 processes, 2 worker threads (PEs) + 1 comm threads per_
↪process, 2 PEs total

```

(continues on next page)

(continued from previous page)

```

Charm++> The comm. thread both sends and receives messages
Charm++> Using recursive bisection (scheme 3) for topology aware partitions
Converse/Charm++ Commit ID: v6.10.2-0-g7bf00fa-namd-charm-6.10.2-build-2020-Aug-05-556
Charm++ communication thread will sleep due to single-process run.
CharmLB> Load balancer assumes all CPUs are same.
Charm++> cpu affinity enabled.
Charm++> Running on 1 hosts (2 sockets x 12 cores x 2 PUs = 48-way SMP)
Charm++> cpu topology info is gathered in 0.013 seconds.
Info: NAMD Git-2021-10-05 for Linux-x86_64-ofi-smp
...

```

The extra argument is needed as multiple PEs get assigned to same core. Setting +setcpuaffinity will prevent that.

You should not pay much attention to timings for this case. The purpose of the executions above is to proof than NAMD works at least for a simple execution. The memory used start showing important changes between the 4 binaries:

```

11:52:06-build@trcis001:/shared/src/NAMD_Git-2021-10-05_Source/Linux-x86_64-mpi-mpicxx$ WallClock: 0.018382 CPUTime: 0.018382 Memory: 4145.171875 MB

11:52:20-build@trcis001:/shared/src/NAMD_Git-2021-10-05_Source/Linux-x86_64-ofi-icc$ WallClock: 0.019521 CPUTime: 0.016802 Memory: 318.988281 MB

11:52:11-build@trcis001:/shared/src/NAMD_Git-2021-10-05_Source/Linux-x86_64-mpi-smp- mpicxx$ WallClock: 0.019347 CPUTime: 0.015666 Memory: 2610.667969 MB

11:52:01-build@trcis001:/shared/src/NAMD_Git-2021-10-05_Source/Linux-x86_64-ofi-smp-icc$ WallClock: 0.064102 CPUTime: 0.028695 Memory: 458.476562 MB

```

The OFI binaries use less memory than the MPI versions. The SMP versions use less memory than the non-SMP versions but the difference is lower compared with the OFI vs MPI binaries.

6.9.4 Script summarizing compilation of NAMD

The next script execute all steps above:

```

#!/bin/bash

VERSION=2021-10-06

if [ ! -f NAMD_Git-${VERSION}_Source.tar.gz ]
then
wget https://www.ks.uiuc.edu/Research/namd/cvs/download/741376/NAMD_Git-${VERSION}_ Source.tar.gz
fi

if [ ! -d NAMD_Git-${VERSION}_Source ]
then
tar -zxvf NAMD_Git-${VERSION}_Source.tar.gz
fi

cd NAMD_Git-${VERSION}_Source/

```

(continues on next page)

(continued from previous page)

```

if [ ! -d charm-6.10.2 ]
then
    tar -xvf charm-6.10.2.tar
fi
cd charm-6.10.2

MPICXX=mpiicpc ./build charm++ mpi-linux-x86_64 mpicxx ifort -j12 --with-production
MPICXX=mpiicpc ./build charm++ mpi-linux-x86_64 mpicxx ifort smp -j12 --with-production
./build charm++ ofi-linux-x86_64 icc ifort -j12 --with-production
./build charm++ ofi-linux-x86_64 icc ifort smp -j12 --with-production

cd mpi-linux-x86_64-ifort-mpicxx/tests/charm++
make clean && make && cd megatest && make && make test
cd ../../..
cd mpi-linux-x86_64-ifort-smp-mpicxx/tests/charm++
make clean && make && cd megatest && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-smp-icc/tests/charm++
make clean && make && cd megatest && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-icc/tests/charm++
make clean && make && cd megatest && make && make test
cd ../../..
cd ofi-linux-x86_64-ifort-smp-icc/tests/charm++
make clean && make && cd megatest && make && make test
cd ../../..

cd ..

wget http://www.ks.uiuc.edu/Research/namd/libraries/fftw-linux-x86_64.tar.gz
tar xzf fftw-linux-x86_64.tar.gz
mv linux-x86_64 fftw
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64.tar.gz
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64-threaded.tar.gz
tar xzf tcl8.5.9-linux-x86_64.tar.gz
tar xzf tcl8.5.9-linux-x86_64-threaded.tar.gz
mv tcl8.5.9-linux-x86_64 tcl
mv tcl8.5.9-linux-x86_64-threaded tcl-threaded

ln -s charm-6.10.2 charm

cat << EOF > arch/Linux-x86_64-mpi-mpicxx.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias $(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

```

(continues on next page)

(continued from previous page)

```

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

cat << EOF > arch/Linux-x86_64-mpi-smp-mpicxx.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = mpi-linux-x86_64-ifort-smp-mpicxx

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

cat << EOF > arch/Linux-x86_64-ofi-icc.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

cat << EOF > arch/Linux-x86_64-ofi-smp-icc.arch
NAMD_ARCH = Linux-x86_64
CHARMARCH = ofi-linux-x86_64-ifort-smp-icc

FLOATOPTS = -ip -axSKYLAKE-AVX512,CASCADELAKE -qopenmp-simd

CXX = icpc -std=c++11
CXXOPTS = -O2 \$(FLOATOPTS)
CXXNOALIASOPTS = -O2 -fno-alias \$(FLOATOPTS)
CXXCOLVAROPTS = -O2 -ip

CC = icc
COPTS = -O2 \$(FLOATOPTS)
EOF

./config Linux-x86_64-mpi-mpicxx --charm-arch mpi-linux-x86_64-ifort-mpicxx
./config Linux-x86_64-mpi-smp-mpicxx --charm-arch mpi-linux-x86_64-ifort-smp-mpicxx
./config Linux-x86_64-ofi-icc --charm-arch ofi-linux-x86_64-ifort-icc

```

(continues on next page)

(continued from previous page)

```

./config Linux-x86_64-ofi-smp-icc --charm-arch ofi-linux-x86_64-ifort-smp-icc

cd Linux-x86_64-mpi-mpicxx
make -j12
cd ..
cd Linux-x86_64-mpi-smp-mpicxx
make -j12
cd ..
cd Linux-x86_64-ofi-icc
make -j12
cd ..
cd Linux-x86_64-ofi-smp-icc
make -j12
cd ..

execstack -c Linux-x86_64-*/namd2

cd Linux-x86_64-mpi-mpicxx
./charmrun ++local +p2 ./namd2 src/alanin
make release
cd ..

cd Linux-x86_64-mpi-smp-mpicxx
./charmrun ++local +p2 ./namd2 src/alanin ++ppn2
make release
cd ..

cd Linux-x86_64-ofi-icc
./charmrun ++local +p2 ./namd2 src/alanin
make release
cd ..

cd Linux-x86_64-ofi-smp-icc
./charmrun ++local +p2 ./namd2 src/alanin ++ppn2 +setcpuaffinity
make release
cd ..

```

6.9.5 Benchmarking NAMD2

NAMD has a case often used for Benchmarking. Still small but we can start extracting some performance figures. ApoA1 benchmark (92,224 atoms, periodic; 2fs timestep with rigid bonds, 12A cutoff with PME every 2 steps):

Download the code with:

```

wget http://www.ks.uiuc.edu/Research/namd/utilities/apollo1.tar.gz
tar xzf apollo1.tar.gz

```

Once you have untar the package. Edit the input file and change the line for the output. You can do that from the command line with:

```
cd apoa1  
cp apoa1.namd apoa1.namd_BKP  
cat apoa1.namd_BKP | sed 's/\/usr//g' > apoa1.namd
```

We start with a simple execution using 12 cores. Notice that the first time you execute NAMD it will compute the FFT optimization and that could take a several seconds. With 12 cores the simulation last for around a minute:

```
./Linux-x86_64-mpi-mpicxx/charmrun +p12 ./Linux-x86_64-mpi-mpicxx/namd2 apoa1.namd  
./Linux-x86_64-mpi-mpicxx/charmrun +p12 ./Linux-x86_64-mpi-mpicxx/namd2 apoa1.namd
```

At the end of the second run the timing was:

```
WallClock: 32.377525 CPUTime: 32.377525 Memory: 2932.089844 MB  
[Partition 0] [Node 0] End of program
```

The second version with MPI and SMP is like this:

```
./Linux-x86_64-mpi-smp-mpicxx/charmrun +p12 ./Linux-x86_64-mpi-smp-mpicxx/namd2 apoa1.  
↳namd ++ppn2  
./Linux-x86_64-mpi-smp-mpicxx/charmrun +p12 ./Linux-x86_64-mpi-smp-mpicxx/namd2 apoa1.  
↳namd ++ppn2
```

The timing for this version is similar:

```
WallClock: 29.577475 CPUTime: 29.438684 Memory: 2853.781250 MB  
[Partition 0] [Node 0] End of program
```

The OFI versions run like this:

```
./Linux-x86_64-ofi-icc/charmrun +p12 ./Linux-x86_64-ofi-icc/namd2 apoa1.namd  
./Linux-x86_64-ofi-icc/charmrun +p12 ./Linux-x86_64-ofi-icc/namd2 apoa1.namd
```

With timings for the second run:

```
WallClock: 33.552193 CPUTime: 33.414692 Memory: 662.109375 MB  
[Partition 0] [Node 0] End of program
```

The final binary is OFI with SMP enabled:

```
./Linux-x86_64-ofi-smp-icc/charmrun +p12 ./Linux-x86_64-ofi-smp-icc/namd2 apoa1.namd  
↳++ppn2  
./Linux-x86_64-ofi-smp-icc/charmrun +p12 ./Linux-x86_64-ofi-smp-icc/namd2 apoa1.namd  
↳++ppn2
```

With timings:

```
WallClock: 34.350666 CPUTime: 34.264492 Memory: 641.882812 MB  
[Partition 0] [Node 0] End of program
```

At this point all four binaries perform very similarly. However, this execution was done on the head node, where several user and system processes could be taking CPU time, making any claim about performance misleading.

Our next step is to move the execution to an isolated compute node where the time could be more accurate.

To do this lets request an interactive execution on an isolated node:

```
qsub -I -n -l nodes=1:ppn=40
```

Once you log into the compute node, load clean your modules and load the Intel Compilers 2021:

```
module purge
module load compiler/2021.2.0 mpi/2021.2.0 mkl/2021.2.0
```

The following script can be used to execute 4 versions of NAMD under the same conditions multiple times to gather a more precise timing. The first execution will be larger due to NAMD computing the FFT parameter optimization. The script could be called `run_apoa1.sh`:

```
#!/bin/bash

for j in 2 4 8 10 20 40
do

for i in 0 1 2 3
do

echo Linux-x86_64-mpi-mpicxx ${j} ${i}
./Linux-x86_64-mpi-mpicxx/charmrn +p$j \
    ./Linux-x86_64-mpi-mpicxx/namd2 apoal.namd > mpi_${j}_${i}.dat
echo Linux-x86_64-mpi-smp-mpicxx ${j} ${i}
./Linux-x86_64-mpi-smp-mpicxx/charmrn +p$j \
    ./Linux-x86_64-mpi-smp-mpicxx/namd2 apoal.namd ++ppn2 > mpi_smp_${j}_${i}.dat
echo Linux-x86_64-ofi-icc ${j} ${i}
./Linux-x86_64-ofi-icc/charmrn +p$j \
    ./Linux-x86_64-ofi-icc/namd2 apoal.namd > ofi_${j}_${i}.dat
echo Linux-x86_64-ofi-smp-icc ${j} ${i}
./Linux-x86_64-ofi-smp-icc/charmrn +p$j \
    ./Linux-x86_64-ofi-smp-icc/namd2 apoal.namd ++ppn2 +setcpuaffinity > ofi_smp_${j}_$i.dat

done
done
```

The script can be executed like this:

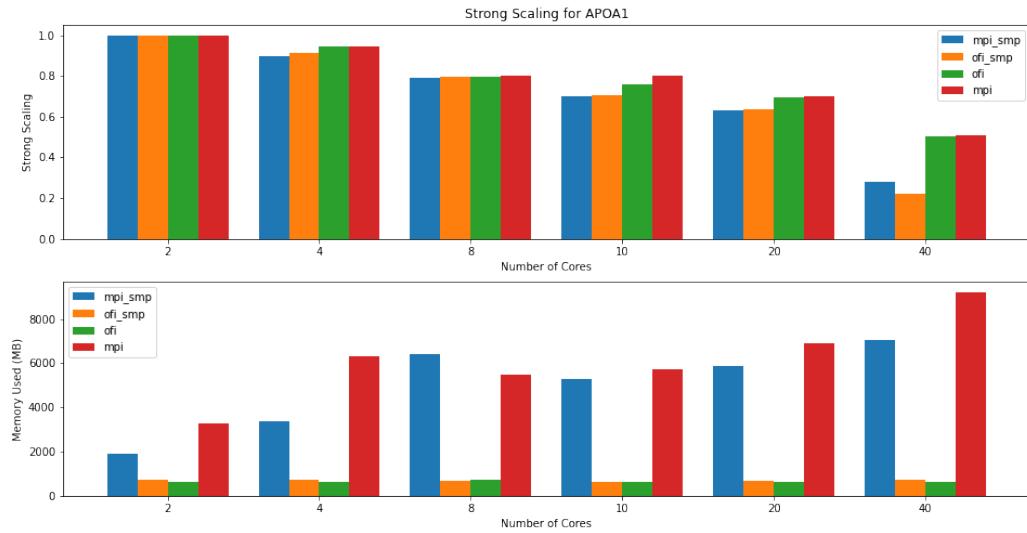
```
./run_apoa1.sh
```

From the figure above we can see that the MPI versions consume more memory as we use more cores and the memory usage is larger compared with the OFI versions. For computing the strong scaling we use the timing for 2 cores as reference. The equation for strong scaling is $t1/(tn^N)$ where $t1$ is the runtime for 1 core, tn is the time when N cores are used. Notice that there is a small advantage for the non-SMP versions (MPI and OFI) and with a high penalty for the 40 core case with scaling under 25%.

More significant for measuring the performance of NAMD for large systems comes from the STMV benchmark (1,066,628 atoms, periodic; 2fs timestep with rigid bonds, 12A cutoff with PME every 2 steps)

Download the input for the STMV benchmark, untar and uncompress the package and move into the folder:

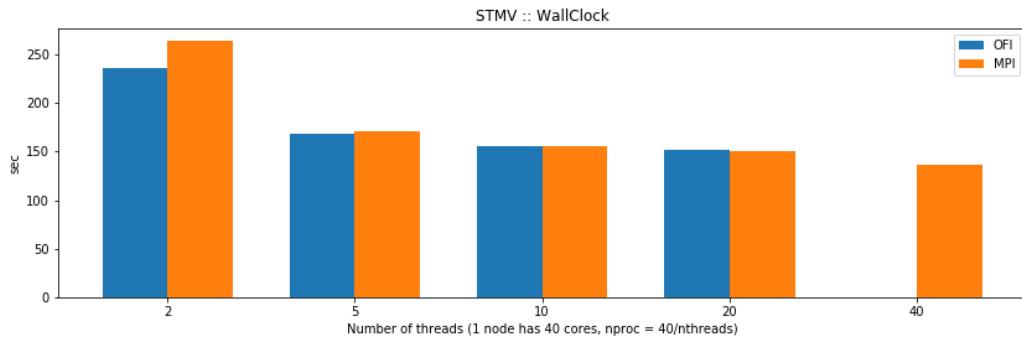
```
wget https://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz
tar -zxvf stmv.tar.gz
cd stmv
```



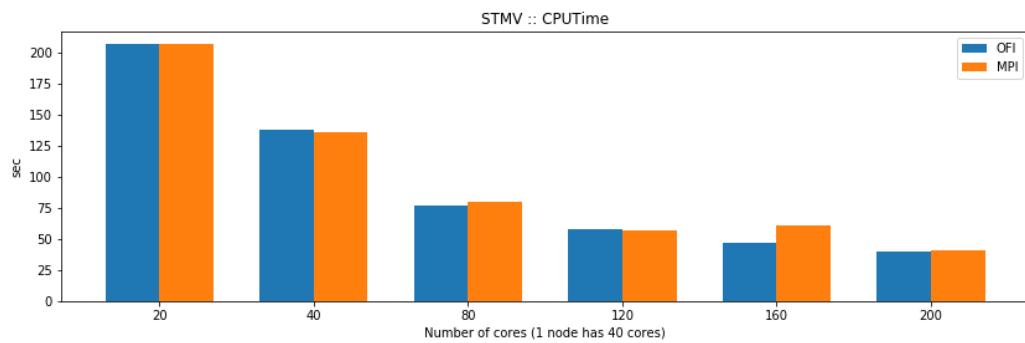
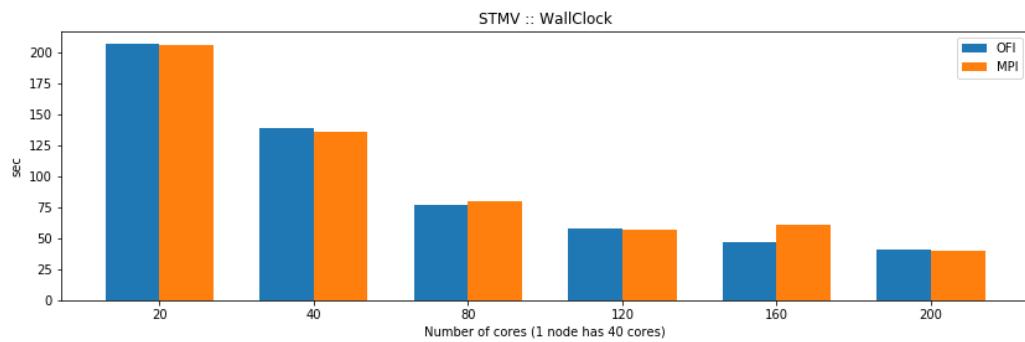
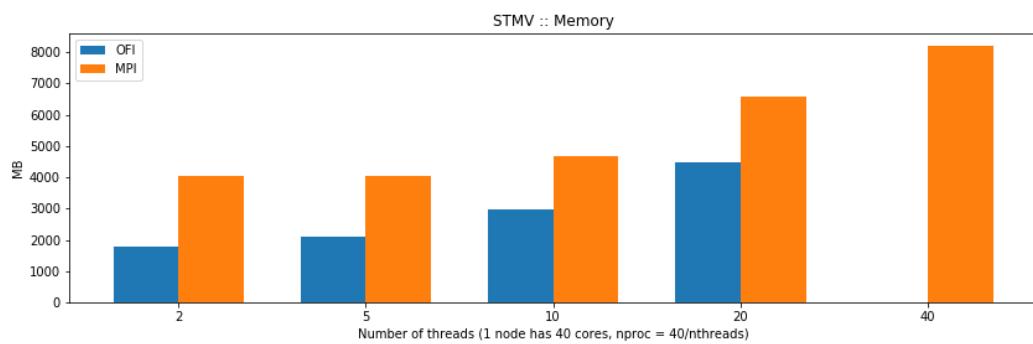
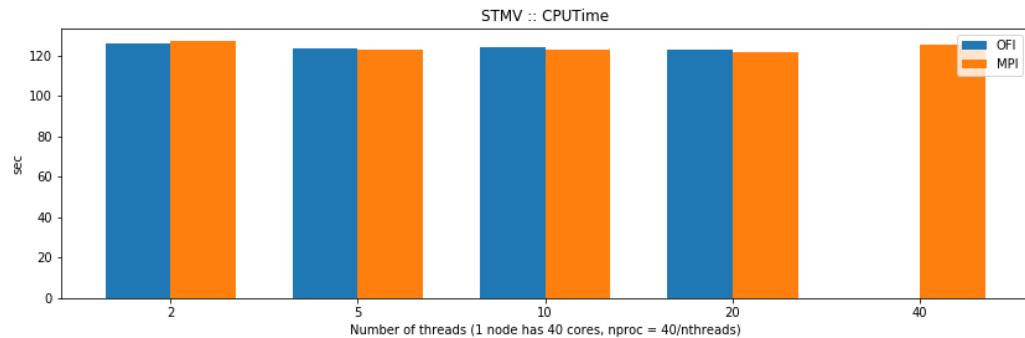
The STMV execution takes longer so a submission script is better suited for the task. Our next set of tests will explore the best performance that we can get using all the cores on a single node. There are several options for the SMP case either adding more worker threads (+pN) or adding more PEs per logical node (++ppn N).

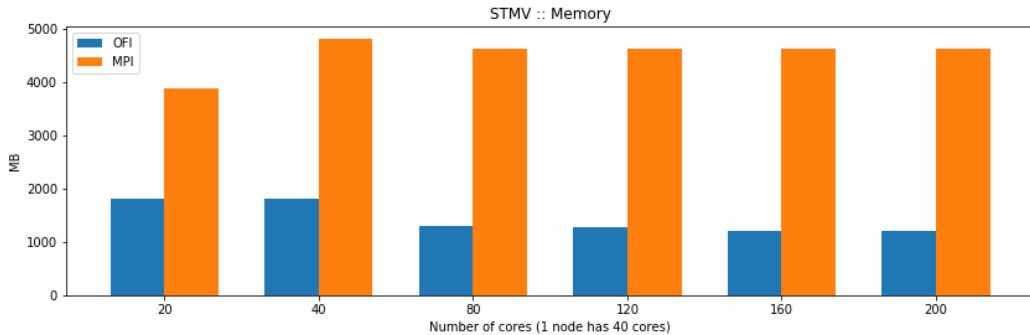
The first set of benchmark uses the SMP builds. Each node has 40 cores so there are several ways of balance the number of process and the number of threads on each process. The benchmarks below uses the 40 cores under a different number of computational threads and the complementary number of processes.

The results show that the MPI and OFI versions are similar in CPU time but differ in wall time. That could be explained by the OFI being more efficient network I/O improving the overall time the computer needs to perform the task. From the point of view of memory usage the OFI version shows considerably better memory usage.



For the case of single thread NAMD builds the benchmarks show that the OFI and MPI flavors behave similar in terms of performance with a small advantage for OFI due to a better communication between nodes compared to the MPI build. The big advantage comes from the memory usage, the OFI build uses less than half memory compared to MPI build.





6.9.6 Conclusions

We have explored the performance of NAMD using two flavors of network communication both for single thread and SMP build. The benchmarks using STMV (1 million atoms) show a small advantage for OFI in most cases. However, it differentiates notably on its memory usage, something that could be critical for systems with large numbers of atoms.

6.10 Density Functional Theory

6.10.1 ABINIT

6.10.2 ABINIT 9.6.2 on Spruce Knob using GCC 10.3

There are 4 builds of GCC 10.3 using various implementations of MPI

Using OpenMPI 3.1.6

The modules used:

```
$> module load lang/gcc/10.3.0 libs/openblas/0.3.19_gcc103 atomistic/wannier90/3.1.0_
˓→gcc103 \
libs/fftw/3.3.10_gcc103 parallel/openmpi/3.1.6_gcc103
```

Create the folder for building inside the sources:

```
/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316
```

The AC9 file is written below, the last lines are only added after the fallbacks were compiled. Comment those 6 final lines before executing `../configure`:

```
prefix="/shared/software/atomistic/abinit/9.6.2_gcc103_ompi316"
with_debug_flavor="verbose"
with_optim_flavor="standard"

FCFLAGS_EXTRA="-fallow-argument-mismatch"
#FCFLAGS_EXTRA="-heap-arrays 1 -xAVX"
#FC_LIBS="-lstdc++ -ldl"
#CPP="icc -E"
```

(continues on next page)

(continued from previous page)

```
LINALG_LIBS="-L${MD_OPENBLAS}/lib -lopenblas -lpthread"
WANNIER90_LIBS="-L${MD_WANNIER90}/lib -lwannier"
with_libxml2="/shared/software/lang/gcc/10.3.0"
enable_openmp="no"

with_fft_flavor="fftw3"
FFTW3_FCFLAGS="-I${MD_FFTW}/include"
FFTW3_LIBS="-L${MD_FFTW}/lib -lfftw3 -lfftw3f"

CC=mpicc
CXX=mpicxx
FC=mpif90

with_libxc=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_fb/gnu/
→10.3/libxc/4.3.4
with_hdf5=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_fb/gnu/10.
→3/hdf5/1.10.6
with_netcdf=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_fb/gnu/
→10.3/netcdf4/4.6.3
with_netcdf_fortran=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_
→fb/gnu/10.3/netcdf4_fortran/4.5.2
with_xm1f90=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_fb/gnu/
→10.3/xm1f90/1.5.3.1
with_libpsml=/gpfs/shared/src/abinit-9.6.2/build_gcc103_ompi316/fallbacks/install_fb/gnu/
→10.3/libpsml/1.1.7
```

Execute:

```
$> ./configure
```

After the configure is executed a folder **fallbacks**, move to that folder and execute:

```
$> cd fallbacks
$> FCFLAGS=-fallow-argument-mismatch ./build-abinit-fallbacks.sh
```

After building the fallbacks the lines can be integrated to the file **srih0001.ac9**.

ABINIT 9.6.2 on Thorny using Intel 2021.4

This is the canonical configuration DFT codes for Thorny module **dft_intel21**:

```
module load dft_intel21
```

The ac9 file for this build:

```
prefix="/shared/software/atomistic/abinit/9.6.2_intel21_impi21"

with_debug_flavor="verbose"
with_optim_flavor="standard"

FCFLAGS_EXTRA="-g -traceback -heap-arrays 1 -xSKYLAKE-AVX512"
```

(continues on next page)

(continued from previous page)

```

FC_LIBS="-lstdc++ -ldl"
CPP="icc -E"

WANNIER90_LIBS="-L${MD_WANNIER90}/lib -lwannier"
with_libxml2="/shared/software/lang/gcc/9.3.0"

CC=mpiicc
CXX=mpiicpc
FC=mpiifort

with_libxc=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/2021.4/
↳ libxc/4.3.4
with_hdf5=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/2021.4/hdf5/
↳ 1.10.6
with_netcdf=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/2021.4/
↳ netcdf4/4.6.3
with_netcdf_fortran=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/
↳ 2021.4/netcdf4_fortran/4.5.2
with_xm1f90=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/2021.4/
↳ xm1f90/1.5.3.1
with_libpsml=/shared/src/abinit-9.6.2/build_intel21/fallbacks/install_fb/intel/2021.4/
↳ libpsml/1.1.7

```

Run make and make install:

```
$> make
$> make install
```

The results of the testsuite:

Suite	failed	passed	succeeded	skipped	disabled	run_etime	tot_etime
atompaw	0	0	0	2	0	0.00	0.02
bigdft	0	0	0	19	0	0.00	0.05
bigdft_paral	0	0	0	4	0	0.00	0.00
built-in	0	0	6	1	0	19.79	19.82
etsf_io	0	0	8	0	0	27.45	27.69
fast	0	0	11	0	0	43.82	44.49
gpu	0	0	0	7	0	0.00	0.01
libxc	1	8	26	0	0	457.34	459.65
mpio	0	0	13	4	0	134.70	140.90
paral	1	8	39	84	0	591.17	595.17
psml	0	2	12	0	0	90.26	90.91
seq	0	0	0	18	0	0.00	0.02
tutomultibinit	0	2	4	0	0	163.75	165.02
tutoparal	0	1	0	27	0	46.55	46.97
tutoplugs	0	4	0	0	0	52.90	53.11
tutorespfn	1	8	20	2	0	922.51	926.44
tutorial	2	10	47	0	0	2210.86	2213.49
unitary	0	0	18	20	0	75.31	75.71
v1	0	1	73	0	0	259.79	262.73
v2	1	14	64	0	0	260.72	263.70
v3	2	12	64	0	0	333.02	337.28

(continues on next page)

(continued from previous page)

v4	0	10	51	0	0	242.36	245.63
v5	3	12	58	0	0	690.92	696.28
v6	0	8	53	0	0	422.96	426.97
v67mbpt	0	9	16	0	0	185.33	187.75
v7	1	17	47	0	0	706.63	712.59
v8	0	17	50	2	0	620.94	625.90
v9	0	16	41	4	0	501.88	504.46
vdwxc	0	0	1	0	0	7.03	7.05
wannier90	2	5	1	0	0	32.28	32.53

Completed in 2400.30 [s]. Average time for test=10.10 [s], stdev=45.74 [s]

Summary: failed=14, succeeded=723, passed=164, skipped=194, disabled=0

ABINIT 9.4.2 on Spruce using Intel 2019

This is the canonical configuration for Spruce module dft_intel19:

```
module load dft_intel19
```

The list of modules loaded are:

```
$> module list
Currently Loaded Modulefiles:
 1) lang/gcc/9.3.0           6) libs/fftw/3.3.8_intel19          11) atomistic/
  ↵elk/7.2.42_intel19        7) atomistic/abinit/9.4.2_intel19    12) atomistic/
  2) lang/intel/2019          8) atomistic/vasp/6.2.1_intel19    13) dft_
  ↵siesta/4.0.2_intel19      9) atomistic/octopus/11.0_intel19
  3) lang/python/cpython_3.9.6_gcc93 10) atomistic/espresso/6.8_intel19
```

The building folder is:

```
/gpfs/shared/src/abinit-9.4.2/build_intel19
```

Create a ac9 file for autoconfigure. The file must have the same name as the headnode with extension .ac9 (For Spruce it will be srih0001.ac9):

```
$ cat srih0001.ac9
prefix="/shared/software/atomistic/abinit/9.4.2_intel19_impi19"

with_debug_flavor="verbose"
with_optim_flavor="standard"

FCFLAGS_EXTRA="-heap-arrays 1 -axAVX,CORE-AVX2,CORE-AVX-I"
FC_LIBS="-lstdc++ -ldl"
CPP="icc -E"

CC=mpiicc
CXX=mpiicpc
FC=mpiifort
```

(continues on next page)

(continued from previous page)

```
enable_mpi_io="no"

#with_libxc=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->libxc/4.3.4
#with_hdf5=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->hdf5/1.10.6
#with_netcdf=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->netcdf4/4.6.3
#with_netcdf_fortran=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/
#->intel/19.0/netcdf4_fortran/4.5.2
#with_xmlf90=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->xmlf90/1.5.3.1
#with_libpsml=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.
#->0/libpsml/1.1.7
```

The last 6 lines are commented as they will be used after compiling the fallbacks. Run a first configure with this ac9:

```
.../configure
```

This first configure will prepare the folders for building the fallbacks:

```
$> cd fallbacks
$> CC=mpiicc CXX=mpiicpc FC=mpiifort ./build-abinit-fallbacks.sh
```

After compiling the fallbacks, move one folder up and remove the comments to the last six lines of the ac9 file enabling the fallbacks for the next configure:

```
$> cd ..
$> tail -n 8 srih0001.ac9

with_libxc=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->libxc/4.3.4
with_hdf5=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->hdf5/1.10.6
with_netcdf=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->netcdf4/4.6.3
with_netcdf_fortran=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/
#->intel/19.0/netcdf4_fortran/4.5.2
with_xmlf90=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.0/
#->xmlf90/1.5.3.1
with_libpsml=/gpfs/shared/src/abinit-9.4.2/build_intel19/fallbacks/install_fb/intel/19.
#->0/libpsml/1.1.7
```

Now proceed to configure again:

```
$> CC=mpiicc CXX=mpiicpc FC=mpiifort .../configure
```

Execute make with an appropriate number of compilation threads:

```
$> make -j16
$> make install
```

To run the testsuite go to the tests folder and execute:

```
$> cd tests
$> python3 ../../tests/runtests.py -j4 -n4
```

On Spruce the results of the testsuite are:

Suite	failed	passed	succeeded	skipped	disabled	run_etime	tot_etime
atompaw	0	0	0	2	0	0.00	0.00
bigdft	0	0	0	19	0	0.00	0.03
bigdft_paral	0	0	0	4	0	0.00	0.01
built-in	0	0	5	2	0	24.60	24.61
etsf_io	0	0	8	0	0	24.32	24.91
fast	0	0	11	0	0	45.27	46.23
gpu	0	0	0	7	0	0.00	0.01
libxc	0	9	26	0	0	417.28	419.47
mpio	0	0	0	17	0	0.00	0.02
paral	11	8	26	76	0	399.02	402.03
psml	0	2	12	0	0	79.79	80.39
seq	0	0	0	18	0	0.00	0.02
tutomultibinit	0	0	3	3	0	29.94	31.10
tutoparal	0	1	0	27	0	33.91	34.53
tutoplugs	0	0	0	4	0	0.00	0.00
tutorespfn	1	9	19	2	0	987.28	991.43
tutorial	4	7	48	0	0	573.78	576.74
unitary	0	0	18	20	0	97.45	97.74
v1	0	0	74	0	0	254.83	257.99
v2	0	12	67	0	0	280.23	283.74
v3	0	12	66	0	0	414.27	420.35
v4	0	10	51	0	0	302.68	306.88
v5	2	14	57	0	0	857.25	864.74
v6	0	8	53	0	0	528.13	533.83
v67mbpt	0	9	16	0	0	251.67	254.97
v7	1	15	49	0	0	929.57	936.62
v8	0	17	50	4	0	1193.84	1199.95
v9	0	15	34	2	0	952.27	956.56
vdwxc	0	0	0	1	0	0.00	0.00
wannier90	0	0	0	8	0	0.00	0.01

Completed in 2381.46 [s]. Average time for test=10.09 [s], stdev=22.42 [s]
 Summary: failed=19, succeeded=693, passed=148, skipped=216, disabled=0

Execution completed.

Results in HTML format are available in Test_suite/suite_report.html

Parallel version with GCC 9.3 and MPICH 3.4.1

Abinit 9.4.1 was compiled with the following modules:

```
module load lang/gcc/9.3.0
module load lang/python/cpython_3.9.4_gcc93
module load parallel/mpich/3.4.1_gcc93
module load libs/openblas/0.3.10_gcc93
module load libs/libxc/4.3.4_gcc93
module load libs/xmlf90/1.5.4_gcc93
module load libs/libpsml/1.1.7_gcc93
module load libs/openblas/0.3.10_gcc93
module load libs/hdf5/1.12.0_gcc93
module load libs/netcdf/4.7.4_gcc93
module load libs/netcdf/fortran-4.5.3_gcc93
module load libs/fftw/3.3.9_gcc93
```

ABINIT uses a configure file, a template can be found at doc/build/config-template.ac9. The lines to be changed from the template are:

```
prefix="/shared/software/atomistic/abinit/9.4.1_gcc93_mplic341"
with_mpi="${MD_MPICH}"
LINALG_LIBS="-L${MD_OPENBLAS}/lib -lopenblas -lpthread "
with_fft_flavor="fftw3"
FFTW3_LIBS="-L${MD_FFTW} -lfftw3 -lfftw3f"
with_libxc=${MD_LIBXC}
with_libxml2=${MD_GCC}
with_hdf5=${MD_HDF5}"
NETCDF_FCFLAGS="-I${MD_NETCDF}/include"
NETCDF_LIBS="-L${MD_NETCDF}/lib -lnetcdf"
NETCDF_FORTRAN_FCFLAGS="-I${MD_NETCDF_FORTRAN}/include"
NETCDF_FORTRAN_LIBS="-L${MD_NETCDF_FORTRAN}/lib -lnetcdff"
LIBPSML_FCFLAGS="-I${MD_LIBPSML}/include"
LIBPSML_LIBS="-L${MD_LIBPSML}/lib -lpsml"
with_xmlf90=${MD_XMLF90}"
```

These lines will use environment variables declared on the corresponding modules loaded above. The configure ac9 is:

```
build_gcc93_mplic341.ac9
```

To configure ABINIT is customary to create a build folder, ABINIT was build inside the folder build_gcc93_mplic341:

```
mkdir build_gcc93_mplic341
cd build_gcc93_mplic341
```

Execute the configure with:

```
../configure --with-config-file=../build_gcc93_mplic341.ac9
```

The resulting summary of the configurations for building ABINIT are these:

Core build parameters

```

* C compiler      : gnu version 9.3
* Fortran compiler : gnu version 9.3
* architecture    : intel xeon (64 bits)
* debugging       : basic
* optimizations   : standard

* OpenMP enabled   : no (collapse: ignored)
* MPI     enabled   : yes (flavor: auto)
* MPI     in-place   : no
* MPI-IO enabled   : yes
* GPU     enabled   : no (flavor: none)

* LibXML2 enabled   : yes
* LibPSML enabled   : yes
* XMLF90  enabled   : yes
* HDF5 enabled      : yes (MPI support: no)
* NetCDF enabled    : yes (MPI support: no)
* NetCDF-F enabled  : yes (MPI support: no)

* FFT flavor       : fftw3 (libs: user-defined)
* LINALG flavor    : netlib (libs: auto-detected)
* SCALAPACK enabled : no
* ELPA enabled      : no

* FCFLAGS          : -g -ffree-line-length-none -I/shared/software/libs/netcdf-c/4.
→ 7.4_gcc93/include -I/shared/software/libs/netcdf-fortran/4.5.3_gcc93/include -I/
→ shared/software/libs/xmlf90/1.5.4_
gcc93/include -I/shared/software/libs/libpsml/1.1.7_gcc93/include
* CPATH            : /shared/software/libs/fftw/3.3.9_gcc93/include:/shared/software/
→ libs/netcdf-fortran/4.5.3_gcc93/include:/shared/software/libs/netcdf-c/4.7.4_gcc93/
→ include:/shared/software/libs/hdf
5/1.12.0_gcc93/include:/shared/software/libs/libpsml/1.1.7_gcc93/include:/shared/
→ software/libs/xmlf90/1.5.4_gcc93/include:/shared/software/libs/libxc/4.3.4_gcc93/
→ include:/shared/software/libs/openblas/0.3
.10_gcc9.3.0/include:/shared/software/parallel/mpich/3.4.1_gcc93/include:/shared/
→ software/lang/python/3.9.4_gcc93/include:/shared/software/lang/gcc/9.3.0/include

* Build workflow    : monolith

0 deprecated options have been used:.

Configuration complete.
You may now type "make" to build Abinit.
(or "make -j<n>", where <n> is the number of available processors)

```

ABINIT can now be build with:

```
make -j12
```

Running the testsuite produces these results:

Suite	failed	passed	succeeded	skipped	disabled	run_etime	tot_etime
atompaw	0	0	0	2	0	0.00	0.00
bigdft	0	0	0	19	0	0.00	0.01
bigdft_paral	0	0	0	4	0	0.00	0.00
built-in	0	0	5	2	0	18.92	18.93
etsf_io	0	0	8	0	0	71.88	72.11
fast	0	1	10	0	0	114.94	115.72
gpu	0	0	0	7	0	0.00	0.00
libxc	1	7	27	0	0	1217.96	1220.14
mpio	1	0	12	4	0	2298.48	2306.73
paral	1	11	33	76	0	6497.65	6502.18
psml	0	2	12	0	0	536.08	536.88
seq	0	0	0	18	0	0.00	0.01
tutomultibinit	0	0	6	0	0	248.18	250.11
tutoparal	0	0	1	26	0	154.05	154.50
tutoplugs	0	0	0	4	0	0.00	0.00
tutorespfn	1	8	20	2	0	4046.58	4050.13
tutorial	2	10	47	0	0	1655.70	1659.39
unitary	0	1	17	20	0	107.07	107.41
v1	0	1	73	0	0	529.11	532.30
v2	0	10	69	0	0	601.61	606.15
v3	0	14	64	0	0	597.51	602.62
v4	0	12	49	0	0	559.48	563.87
v5	2	12	59	0	0	2705.48	2712.52
v6	0	7	54	0	0	1491.29	1495.96
v67mbpt	1	9	15	0	0	645.63	648.78
v7	1	14	50	0	0	2800.01	2806.79
v8	0	17	52	2	0	3690.55	3696.26
v9	0	9	42	0	0	1196.94	1200.38
vdwxc	0	0	0	1	0	0.00	0.00
wannier90	0	0	0	8	0	0.00	0.00

Completed in 3760.01 [s]. Average time for test=36.12 [s], stdev=97.31 [s]
Summary: failed=10, succeeded=725, passed=145, skipped=195, disabled=0

CUDA Version with GCC 9.3, MPICH 3.4.1 and CUDA 11.1

Similar to the parallel version above with the addition of this module:

parallel/cuda/11.1

The configue file was:

```
prefix="/shared/software/atomistic/abinit/9.4.1_gcc93_mpich341_gpu"
with_mpi="${MD_MPICH}"
with_gpu="/usr/local/cuda"
with_gpu_flavor="cuda-double"
GPU_CPPFLAGS="-I/usr/local/cuda/include"
GPU_CFLAGS="-I/usr/local/cuda/include"
GPU_CXXFLAGS="-std=c++"
GPU_FCFLAGS="-I/usr/local/cuda/include"
GPU_LDFLAGS="-L/usr/local/cuda/lib64 -lcublas -lcufft -lcudart -lstdc++"
```

(continues on next page)

(continued from previous page)

```

GPU_LIBS="-L/usr/local/cuda/lib64 -lcublas -lcufft -lcudart -lstdc++"
LINALG_LIBS="-L${MD_OPENBLAS}/lib -lopenblas -lpthread "
with_fft_flavor="fftw3"
FFTW3_LIBS="-L${MD_FFTW} -lfftw3 -lfftw3f"
with_libxc=${MD_LIBXC}
with_libxml2="${MD_GCC}"
with_hdf5="${MD_HDF5}"
NETCDF_FCFLAGS="-I${MD_NETCDF}/include"
NETCDF_LIBS="-L${MD_NETCDF}/lib -lnetcdf"
NETCDF_FORTRAN_FCFLAGS="-I${MD_NETCDF_FORTRAN}/include"
NETCDF_FORTRAN_LIBS="-L${MD_NETCDF_FORTRAN}/lib -lnetcdff"
LIBPSML_FCFLAGS="-I${MD_LIBPSML}/include"
LIBPSML_LIBS="-L${MD_LIBPSML}/lib -lpsml"
with_xmlf90="${MD_XMLF90}"

```

The code must be compiled from a compute node with GPUs as the CUDA toolkit is only present there.

Abinit 9.6.2 on Spruce with GCC 10.3 and OpenMPI 3.1.6

Modules loaded::

```
$ $> module load lang/gcc/10.3.0 lang/python/cpython_3.10.2_gcc103 libs/openblas/0.3.19_gcc103 atomistic/wannier90/3.1.0_gcc103 parallel/openmpi/3.1.6_gcc103 libs/fftw/3.3.10_gcc103
```

Test Suite:

Suite	failed	passed	succeeded	skipped	disabled	run_etime	tot_etime
atompaw	0	0	0	2	0	0.00	0.00
bigdft	0	0	0	19	0	0.00	0.01
bigdft_paral	0	0	0	4	0	0.00	0.00
built-in	0	0	6	1	0	26.13	26.15
etsf_io	0	0	8	0	0	112.21	112.56
fast	0	1	10	0	0	182.93	183.29
gpu	0	0	0	7	0	0.00	0.00
libxc	0	13	22	0	0	1462.06	1464.53
mpio	1	1	11	4	0	3356.26	3372.92
paral	3	8	37	84	0	5626.88	5632.48
psml	0	2	12	0	0	234.75	235.43
seq	0	0	0	18	0	0.00	0.02
tutomultibinit	0	2	4	0	0	612.53	613.89
tutoparal	0	1	0	27	0	42.99	43.59
tutoplugs	0	4	0	0	0	275.10	275.27
tutorespfn	2	16	11	2	0	4887.03	4890.27
tutorial	2	14	43	0	0	2780.47	2783.86
unitary	0	0	18	20	0	182.90	183.27
v1	0	1	73	0	0	700.69	703.41
v2	1	16	62	0	0	742.17	745.30
v3	2	14	62	0	0	1144.97	1150.12
v4	0	14	47	0	0	1147.91	1152.00
v5	3	15	55	0	0	2396.42	2403.60
v6	0	11	50	0	0	1743.95	1748.69
v67mbpt	0	10	15	0	0	1025.92	1029.32

(continues on next page)

(continued from previous page)

	1	26	38	0	0	3385.66	3395.43
v7	1	26	38	0	0	3385.66	3395.43
v8	0	21	46	2	0	2436.99	2442.48
v9	0	27	30	4	0	1734.56	1738.00
vdwxc	0	0	1	0	0	40.25	40.29
wannier90	2	6	0	0	0	234.82	235.19

Completed in 9245.87 [s]. Average time for test=40.53 [s], stdev=106.36 [s]
 Summary: failed=17, succeeded=661, passed=223, skipped=194, disabled=0

6.10.3 Octopus

Octopus 11.3 on Thorny Flat using Intel 2021.4

Use the metamodule for DFT codes:

```
$> module purge
$> module load dft_intel21
```

Resulting in these modules being loaded:

Currently Loaded Modulefiles:

1) lang/gcc/9.3.0	15) libs/arpack-ng/3.8.0_intel21_impi21
2) tbb/latest	16) libs/xmlf90/1.5.5_intel21
3) compiler-rt/latest	17) libs/libpsml/1.1.10_intel21
4) compiler/latest	18) libs/gridxc/0.9.6_intel21
5) mpi/latest	19) atomistic/wannier90/3.1.0_intel21
6) mkl/latest	20) atomistic/abinit/9.6.2_intel21_impi21
7) lang/python/cpython_3.9.7_gcc93	21) atomistic/vasp/6.2.1_intel21
8) libs/openblas/0.3.17_intel21	22) atomistic/octopus/11.3_intel21_impi21
9) libs/libxc/5.1.7_intel21	23) atomistic/espresso/6.8_intel21
10) libs/hdf5/1.12.1_intel21	24) atomistic/elk/8.3.15_intel21
11) libs/hdf5/1.12.1_intel21_impi21	25) atomistic/siesta/4.1.5_intel21
12) libs/netcdf/4.8.1_intel21_impi21	26) atomistic/siesta/4.1.5_psml_intel21
13) libs/netcdf/fortran-4.5.3_intel21_impi21	27) dft_intel21
14) libs/fftw/3.3.9_intel21_impi21	

The building folder is:

```
/shared/src/octopus-11.3/build_intel21
```

Run the configure script:

```
MKL="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm
-lldl"

./configure --prefix=/shared/software/atomistic/octopus/11.3_intel21_impi21 \
--with-blas="${MKL}" \
--with-lapack="${MKL}" \
--enable-mpi \
--with-libxc-prefix=$MD_LIBXC \
--with-boost=/shared/software/libs/boost/1.78_gcc93 \
```

(continues on next page)

(continued from previous page)

```
--with-sparskit=/shared/software/libs/sparskit2/lib/libskit.a \
CC=mpiicc CXX=mpiicpc FC=mpiifort FCFLAGS="-g -traceback -heap-arrays 1 -O2 -fp-
model=precise"
```

Compile the code using 12 compilation threads:

```
$> make -j 12
```

Run the testsuite:

```
$> ulimit -s unlimited
$> make check
```

The results of the testsuite were:

```
Status: 8 failures
    Passed: 223 / 228
    Skipped: 1 / 228
    Failed: 4 / 228

    testfile                                # failed testcases
-----
periodic_systems/17-aluminium.test          4
periodic_systems/18-TiO2.test                2
lda_u/07-noncollinear.test                 3
functionals/18-mgga.test                   8

Total run-time of the testsuite: 00:10:11
```

Octopus 11.0 on Spruce using Intel 2019

This is the canonical configuration for Spruce module dft_intel19:

```
module load dft_intel19
```

The list of modules loaded are:

```
$> module list
Currently Loaded Modulefiles:
  1) lang/gcc/9.3.0           6) libs/fftw/3.3.8_intel19      11) atomistic/
  ↵elk/7.2.42_intel19        7) atomistic/abinit/9.4.2_intel19   12) atomistic/
  2) lang/intel/2019          8) atomistic/vasp/6.2.1_intel19    13) dft_
  ↵siesta/4.0.2_intel19      9) atomistic/octopus/11.0_intel19
  3) lang/python/cpython_3.9.6_gcc93 10) atomistic/espresso/6.8_intel19
```

The building folder is:

```
/shared/src/octopus-11.0/build_intel19
```

Execute this configure line:

```
$> ./configure --prefix=/shared/software/atomistic/octopus/11.0_intel19 \
--with-lapack="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core - \
-lmomp5 -lpthread -lm -ldl" \
--with-blas="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core - \
-lmomp5 -lpthread -lm -ldl" \
--with-blacs="${MKLROOT}/lib/intel64/libmkl_blacs_intelmpi_lp64.a" \
--with-scalapack="${MKLROOT}/lib/intel64/libmkl_scalapack_lp64.a CC=mpiicc CXX=mpiicpc \
-FC=mpiifort
```

Build the software and install:

```
$> make
$> make install
```

Running the tests with:

```
$> make check
```

The results were:

```
Status: 8 failures
Passed: 217 / 223
Skipped: 1 / 223
Failed: 5 / 223

testfile                                     # failed testcases
-----
maxwell/02-external-current.test            3
periodic_systems/06-h2o_pol_lr.test         2
linear_response/01-casida.test              16
pseudopotentials/14-carbon_dojo_psp8.test   1
functionals/18-mgga.test                     8
```

Total run-time of the testsuite: 02:16:08

```
make[3]: *** [Makefile:876: check-base] Error 5
make[3]: Leaving directory '/gpfs/shared/src/octopus-11.0/build_intel19/testsuite'
make[2]: *** [Makefile:865: check-wrapper] Error 2
make[2]: Leaving directory '/gpfs/shared/src/octopus-11.0/build_intel19/testsuite'
make[1]: *** [Makefile:853: check] Error 2
make[1]: Leaving directory '/gpfs/shared/src/octopus-11.0/build_intel19/testsuite'
make: *** [Makefile:545: check-recursive] Error 1
```

Octopus 10.4 with GCC 9.3

Octopus is a Real Space DFT code. This instructions show how to compile Octopus 10.4 (latest version by 2021.04.19). This is the parallel version compiled with GCC 9.3

The modules loaded for compilation were:

```
module load lang/gcc/9.3.0 libs/libxc/4.3.4_gcc93 \
libs/hdf5/1.12.0_gcc93 \
libs/netcdf/4.7.4_gcc93 \
libs/netcdf/fortran-4.5.3_gcc93 \
libs/openblas/0.3.10_gcc93 \
libs/fftw/3.3.9_gcc93 \
parallel/mpich/3.4.1_gcc93
```

The sources can be downloaded from the developers and uncompressed with:

```
wget https://octopus-code.org/download/10.4/octopus-10.4.tar.gz
tar -zxvf octopus-10.4.tar.gz
```

It is customary to compile codes on a separate folder from the sources. The foler build_gcc93 is created inside the sources for that purpose:

```
cd octopus-10.4
mkdir build_gcc93_mpich341
cd build_gcc93_mpich341
```

The configure line was:

```
../configure --prefix=/shared/software/atomistic/octopus/10.4_gcc93_mpich341 \
--with-libxc-prefix=${MD_LIBXC} --with-blas="-L${MD_OPENBLAS} -lopenblas" \
--with-fftw-prefix=${MD_FFTW} --with-netcdf-prefix=${MD_NETCDF_FORTRAN} \
--with-mpi=${MD_MPICH} --enable-mpi
```

On Thorny Flat the results from the testsuite were:

```
*****
Passed: 184 / 200
Skipped: 16 / 200

Everything seems to be OK

Total run-time of the testsuite: 00:20:42
```

6.10.4 Quantum Espresso

Quantum Espresso 6.8 on Spruce using Intel 2019

Download the sources:

```
wget https://github.com/QEF/q-e/releases/download/qe-6.8/qe-6.8-ReleasePack.tgz
```

This is the canonical configuration for Spruce module dft_intel19:

```
module load dft_intel19
```

The list of modules loaded are:

```
$> module list
Currently Loaded Modulefiles:
 1) lang/gcc/9.3.0
 2) lang/intel/2019
 3) lang/python/cpython_3.9.7_gcc93
 4) libs/libxc/5.1.5_intel19
 5) libs/hdf5/1.12.1_intel19
 6) libs/fftw/3.3.8_intel19
 7) libs/arpack-ng/3.8.0_intel19
 8) libs/xm1f90/1.5.4_intel19
 9) libs/psml/1.1.10_intel19
10) libs/gridxc/0.9.6_intel19
11) atomistic/abinit/9.4.2_intel19
12) atomistic/vasp/6.2.1_intel19
13) atomistic/octopus/11.0_intel19
14) atomistic/espresso/6.8_intel19
15) atomistic/elk/7.2.42_intel19
16) atomistic/siesta/4.1.5_intel19
17) atomistic/siesta/4.1.5_psml_intel19
18) dft_intel19
```

Uncompress the sources:

```
$> tar -zvxf qe-6.8-ReleasePack.tgz
$> cd qe-6.8
```

Configure the sources to use HDF5, libXC and FFTW 3:

```
$> ./configure CC=mpiicc CXX=mpiicpc FC=mpiifort --with-hdf5 --with-libxc --enable-parallel \
--prefix=/shared/software/atomistic/qe/6.8_intel19 LDFLAGS="-L${MD_FFTW}/lib -lfftw3"
```

Build the binaries:

```
$> make
$> make install
$> make check
```

6.10.5 Siesta

Siesta is a electronic structure code using linear scaling algorithms. The version compiled was 4.0.2. The code was compiled with Intel Compilers 2018 and 2019

To compile the code a arch.make needs to be created. The contents of the file are:

```
SIESTA_ARCH=intel-mpi

FC=mpiifort
FFLAGS=-g -xHost -O3 -prec-div -prec-sqrt -fp-model precise -qopt-prefetch -fPIC -m64

DUMMY_FOX=--enable-dummy
FFLAGS_DEBUG=-g -O2 -debug full -traceback -C
LDFLAGS= -static-intel -static-libgcc
RANLIB=ranlib
FC_SERIAL=ifort
FPPFLAGS_CDF=

MPI_INTERFACE=libmpi_f90.a
```

(continues on next page)

(continued from previous page)

```

MKL_INCLUDE=-I$(MKLROOT)/include
MPI_LIBS=-L$(I_MPI_ROOT)/intel64/lib -lmpi
MKL_LIBS=$(MKLROOT)/lib/intel64
MPI_INCLUDE=-I$(I_MPI_ROOT)/intel64/include
INCFLAGS=$(MPI_INCLUDE) $(MKL_INCLUDE)

FPPFLAGS_MPI=-DMPI -DMPI_TIMING -DFC_HAVE_FLUSH -DFC_HAVE_ABORT -DSIESTA__NO_MRRR

NETCDF_LIBS=
NETCDF_INTERFACE=

LIBS=-mkl=cluster $(MPI_LIBS) -qopenmp -lpthread -lstdc++ -ldl

SYS=nag
FPPFLAGS= $(FPPFLAGS_CDF) $(FPPFLAGS_MPI)

atom.o: atom.F
    $(FC) -c $(FFLAGS_DEBUG) $(INCFLAGS) $(FPPFLAGS) $(FPPFLAGS_fixed_F) $<
state_analysis.o: state_analysis.F
    $(FC) -c $(FFLAGS_DEBUG) $(INCFLAGS) $(FPPFLAGS) $(FPPFLAGS_fixed_F) $<

.F.o:
    $(FC) -c $(FFLAGS) $(INCFLAGS) $(FPPFLAGS) $<
.f.o:
    $(FC) -c $(FFLAGS) $(INCFLAGS) $<
.F90.o:
    $(FC) -c $(FFLAGS) $(INCFLAGS) $(FPPFLAGS) $<
.f90.o:
    $(FC) -c $(FFLAGS) $(INCFLAGS) $<

```

6.10.6 CASTEP

CASTEP is a leading code for calculating the properties of materials from first principles. Using density functional theory, it can simulate a wide range of properties of materials including energetics, structure at the atomic level, vibrational properties, electronic response properties etc. In particular it has a wide range of spectroscopic features that link directly to experiment, such as infra-red and Raman spectroscopies, NMR, and core level spectra.

CASTEP can only be compiled with Intel 2018 due to a bug on Intel 2019 MPI implementation. The code was compiled on both clusters with Intel 2018.

Modules used:

```
$> module purge
$> module load lang/python/intelpython_2.7.16 lang/intel/2018
```

Compilation line:

```
$> make ARCH=linux_x86_64_ifort18 COMMS_ARCH=mpi SUBARCH=mpi FFT=mkl MATHLIBS=mkl110 \
INSTALL_DIR=/shared/software/atomistic/castep/19.11-mpi_intel18 \
FFTLIBDIR=${MKLROOT} MATHLIBDIR=${MKLROOT} -j 8
```

A run of a test suite on both clusters passes all tests.

On Spruce:

```
$ make ARCH=linux_x86_64_ifort18 COMMS_ARCH=mpi SUBARCH=mpi FFT=mkl MATHLIBS=mkl10_
↪INSTALL_DIR=/shared/software/atomistic/castep/19.11-mpi_intel18 \
FFTLIBDIR=${MKLROOT} MATHLIBDIR=${MKLROOT} -j 8 check

Makefile:595: GNU make version 3.82 or later is recommended: proceeding with Make 3.81
Some modules may be compiled at unnecessarily low optimisation level

make -C "Test" ARCH=linux_x86_64_ifort18--mpi check-simple
make[1]: Entering directory `/gpfs/shared/src/CASTEP-19.11/Test'
rm -f */*/*.{castep,dfpt_wvfn,fd_wvfn,wvfn.*,*_.err}
./bin/testcode.py -q --processors=4 --total-processors=16 -e /gpfs/shared/src/CASTEP-
↪19.11/obj/linux_x86_64_ifort18--mpi/castep.mpi -c simple
.....
↪.....
.....
↪.....
.....
↪.....
.....
[464/464]
make[1]: Leaving directory `/gpfs/shared/src/CASTEP-19.11/Test'
```

On Thorny:

```
$ make ARCH=linux_x86_64_ifort18 COMMS_ARCH=mpi SUBARCH=mpi FFT=mkl MATHLIBS=mkl10_
↪INSTALL_DIR=/shared/software/atomistic/castep/19.11-mpi_intel18 \
FFTLIBDIR=${MKLROOT} MATHLIBDIR=${MKLROOT} -j 8 check
make -C "Test" ARCH=linux_x86_64_ifort18--mpi check-simple
make[1]: Entering directory `/gpfs20/shared/src/CASTEP-19.11/Test'
rm -f */*/*.{castep,dfpt_wvfn,fd_wvfn,wvfn.*,*_.err}
./bin/testcode.py -q --processors=4 --total-processors=48 -e /gpfs20/shared/src/
↪CASTEP-19.11/obj/linux_x86_64_ifort18--mpi/castep.mpi -c simple
.....
↪.....
.....
↪.....
.....
[464/464]
make[1]: Leaving directory `/gpfs20/shared/src/CASTEP-19.11/Test'
```

6.10.7 VASP

VASP 6.2.1 on Thorny Flat with Intel 2021.4

The Vienna Ab initio Simulation Package (VASP) is a computer program for atomic scale materials modelling, e.g. electronic structure calculations and quantum-mechanical molecular dynamics, from first principles.

VASP 6.2.1 was compiled with Intel 2021.4 on Thorny Flat. There are two builds of VASP, one compiled with a MKL running the routines sequential mode (no multithreading) and another build with OpenMP enabled and MKL running in multithreaded mode.

VASP is a proprietary code that require a license to legally run the code. The downloaded file is called `vasp.6.2.1.tar.gz` that uncompress into a folder `vasp.6.2.1`.

Before compiling VASP, you need to edit the file `makefile.include` for the sequential version:

```
# Precompiler options
CPP_OPTIONS= -DHOST=\"LinuxIFC\" \
              -DMPI -DMPI_BLOCK=8000 \
              -DCACHE_SIZE=4000 \
              -DscaLAPACK \
              -Dvasp6 \
              -Duse_bse_te \
              -Dtbdyn \
              -Dfock_dblbuf

CPP      = fpp -f_com=no -free -w0 $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FC       = mpiifort
FCL     = mpiifort

FREE    = -free -names lowercase

FFLAGS   = -assume byterecl -w -traceback -static-libstdc++ -static-libgcc -heap-
           ↵ arrays 1 -xSKYLAKE-AVX512
OFLAG    = -O2 -g3
OFLAG_IN = $(OFLAG)
DEBUG    = -O0 -g3

MKL_PATH = $(MKLROOT)/lib/intel64
BLAS     = -qmkl=sequential -static-intel
LAPACK   =
BLACS    = -lmkl_blacs_intelmpi_lp64
SCALAPACK = $(MKL_PATH)/libmkl_scalapack_lp64.a $(BLACS)

OBJECTS  = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o

INCS     =-I$(MKLROOT)/include/fftw

LLIBS    = $(SCALAPACK) $(LAPACK) $(BLAS)

OBJECTS_01 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_02 += fft3dlib.o

# For what used to be vasp.5.lib
CPP_LIB  = $(CPP)
FC_LIB   = $(FC)
CC_LIB   = icc
CFLAGS_LIB = -O -static-libgcc -xSKYLAKE-AVX512
FFLAGS_LIB = -O1 -static-libgcc -xSKYLAKE-AVX512
FREE_LIB  = $(FREE)

OBJECTS_LIB= linpack_double.o getshmem.o

# For the parser library
```

(continues on next page)

(continued from previous page)

```
CXX_PARS = icpc
LLIBS += -lstdc++ -static-libstdc++ -xSKYLAKE-AVX512

# Normally no need to change this
SRCDIR = ../../src
BINDIR = ../../bin
```

Running tests:

```
$> make test
```

VASP 6.2.1 on Spruce using Intel 2019

```
# Precompiler options
CPP_OPTIONS= -DHOST=\"LinuxIFC\" \
    -DMPI -DMPI_BLOCK=8000 \
    -Duse_collective \
    -DCACHE_SIZE=4000 \
    -DscaLAPACK \
    -Dvasp6 \
    -Duse_bse_te \
    -Dtbdyn \
    -Dfock_dbdbuf

CPP      = fpp -f_com=no -free -w0 $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FC       = mpiifort
FCL      = mpiifort

FREE     = -free -names lowercase

FFLAGS   = -assume byterecl -w -axSANDYBRIDGE,IVYBRIDGE,HASWELL -static-libgcc -
    ↵traceback -g
OFLAG    = -O2
OFLAG_IN = $(OFLAG)
DEBUG    = -O0

MKL_PATH = $(MKLROOT)/lib/intel64
BLAS     = ${MKLROOT}/lib/intel64/libmkl_core.a
LAPACK   = ${MKLROOT}/lib/intel64/libmkl_intel_lp64.a ${MKLROOT}/lib/intel64/libmkl_-
    ↵sequential.a
BLACS    = ${MKLROOT}/lib/intel64/libmkl_blacs_intelmpi_lp64.a
SCALAPACK = ${MKLROOT}/lib/intel64/libmkl_scalapack_lp64.a

OBJECTS  = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o

INCS     = -I${MKLROOT}/include -I$(MKLROOT)/include/fftw

LLIBS    = $(SCALAPACK) -Wl,--start-group $(LAPACK) $(BLAS) $(BLACS) -Wl,--end-group -
    ↵lpthread -lm -ldl
```

(continues on next page)

(continued from previous page)

```

OBJECTS_01 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_02 += fft3dlib.o

# For what used to be vasp.5.lib
CPP_LIB      = $(CPP)
FC_LIB       = $(FC)
CC_LIB       = icc
CFLAGS_LIB   = -O -static-libgcc -axSANDYBRIDGE,IVYBRIDGE,HASWELL
FFLAGS_LIB   = -O2 -static-libgcc -axSANDYBRIDGE,IVYBRIDGE,HASWELL
FREE_LIB     = $(FREE)

OBJECTS_LIB= linpack_double.o getshmem.o

# For the parser library
CXX_PARS    = icpc
LLIBS       += -lstdc++ -static-libstdc++ -static-libgcc -static-intel

# Normally no need to change this
SRCDIR      = ../../src
BINDIR      = ../../bin

```

The version that runs MKL with multithreading and enables OpenMP is like this:

```

# Precompiler options
CPP_OPTIONS= -DHOST=\"LinuxIFC\" \
              -DMPI -DMPI_BLOCK=8000 \
              -Duse_collective \
              -DCACHE_SIZE=4000 \
              -DscaLAPACK \
              -Dvasp6 \
              -Duse_bse_te \
              -Dtbdyn \
              -Dfock_dbdbuf \
              -D_OPENMP

CPP        = fpp -f_com=no -free -w0 $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FC         = mpiifort
FCL        = mpiifort

FREE       = -free -names lowercase

FFLAGS     = -assume byterecl -w -axSANDYBRIDGE,IVYBRIDGE,HASWELL -static-intel -static-
            ↪libgcc -traceback -g -qopenmp
OFLAG      = -O2
OFLAG_IN   = $(OFLAG)
DEBUG      = -O0

MKL_PATH   = $(MKLROOT)/lib/intel64
BLAS       = ${MKLROOT}/lib/intel64/libmkl_core.a
LAPACK     = ${MKLROOT}/lib/intel64/libmkl_intel_lp64.a ${MKLROOT}/lib/intel64/libmkl_
            ↪intel_thread.a

```

(continues on next page)

(continued from previous page)

```

BLACS      = ${MKLROOT}/lib/intel64/libmkl_blacs_intelmpi_lp64.a
SCALAPACK  = ${MKLROOT}/lib/intel64/libmkl_scalapack_lp64.a

OBJECTS    = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o

INCS       = -I${MKLROOT}/include -I${MKLROOT}/include/fftw

LLIBS       = $(SCALAPACK) -Wl,--start-group $(LAPACK) $(BLAS) $(BLACS) -Wl,--end-group -
             -liomp5 -lpthread -lm -ldl

OBJECTS_01 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_02 += fft3dlib.o

# For what used to be vasp.5.lib
CPP_LIB    = $(CPP)
FC_LIB     = $(FC)
CC_LIB     = icc
CFLAGS_LIB = -O -axSANDYBRIDGE,IVYBRIDGE,HASWELL -static-libgcc
FFLAGS_LIB = -O2 -axSANDYBRIDGE,IVYBRIDGE,HASWELL -static-libgcc
FREE_LIB   = $(FREE)

OBJECTS_LIB= linpack_double.o getshmem.o

# For the parser library
CXX_PARS   = icpc
LLIBS      += -lstdc++ -static-libstdc++ -static-libgcc -static-intel

# Normally no need to change this
SRCDIR    = ../../src
BINDIR    = ../../bin

```

The only module needed to compile VASP is:

```

module purge
module load lang/intel/2019

```

VASP includes a testsuite and running it produces this final results:

```

=====
SUMMARY:
=====

The following tests failed, please check the output file manually:
bulk_SiO2LOPTICS bulk_SiO2LOPTICSnosym bulk_SiO2LOPTICSRPR
bulk_SiO2LPEAD bulk_SiO2LPEADnosym bulk_SiO2LPEADRPR
C_2x2x2CORECON C_2x2x2CORECONRPR

```

VASP 6.2.1 on Spruce using Intel 2019

This is the canonical configuration for Spruce module dft_intel19:

```
module load dft_intel19
```

The list of modules loaded are:

```
$> module list
Currently Loaded Modulefiles:
 1) lang/gcc/9.3.0           6) libs/fftw/3.3.8_intel19      11) atomistic/
  ↵elk/7.2.42_intel19        7) atomistic/abinit/9.4.2_intel19   12) atomistic/
  2) lang/intel/2019          8) atomistic/vasp/6.2.1_intel19    13) dft_
  ↵siesta/4.0.2_intel19      9) atomistic/octopus/11.0_intel19
  3) lang/python/cpython_3.9.6_gcc93 10) atomistic/espresso/6.8_intel19
```

In case of having a previous build, erase the folder:

```
rm -rf build/*
```

Create a file `makefile.include` with the contents as follows:

```
# Precompiler options
CPP_OPTIONS= -DHOST=\"LinuxIFC\" \
              -DMPI -DMPI_BLOCK=8000 \
              -DCACHE_SIZE=4000 \
              -DscaLAPACK \
              -Dvasp6 \
              -Duse_bse_te \
              -Dtbdyn \
              -Dfock_dbdbuf

CPP      = fpp -f_com=no -free -w0 $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FC       = mpiifort
FCL     = mpiifort

FREE     = -free -names lowercase

FFLAGS   = -assume byterecl -w -traceback -static-libstdc++ -static-libgcc -heap-
  ↵arrays 1 -axAVX,CORE-AVX2,CORE-AVX-I
OFLAG    = -O2 -g3
OFLAG_IN = $(OFLAG)
DEBUG    = -O0 -g3

MKL_PATH = $(MKLROOT)/lib/intel64
BLAS     = -mkl=sequential -static-intel
LAPACK   =
BLACS    = -lmkl_blacs_intelmpi_lp64
SCALAPACK = $(MKL_PATH)/libmkl_scalapack_lp64.a $(BLACS)
```

(continues on next page)

(continued from previous page)

```

OBJECTS      = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o
INCS        = -I$(MKLROOT)/include/fftw
LLIBS        = $(SCALAPACK) $(LAPACK) $(BLAS)

OBJECTS_01 += fftw3d.o fftmpi.o fftmpiw.o
OBJECTS_02 += fft3dlib.o

# For what used to be vaspx.5.lib
CPP_LIB     = $(CPP)
FC_LIB      = $(FC)
CC_LIB      = icc
CFLAGS_LIB = -O -static-libgcc -axAVX,CORE-AVX2,CORE-AVX-I
FFLAGS_LIB = -O1 -static-libgcc -axAVX,CORE-AVX2,CORE-AVX-I
FREE_LIB    = $(FREE)

OBJECTS_LIB= linpack_double.o getshmem.o

# For the parser library
CXX_PARS   = icpc
LLIBS      += -lstdc++ -static-libstdc++ -axAVX,CORE-AVX2,CORE-AVX-I

# Normally no need to change this
SRCDIR     = ../../src
BINDIR     = ../../bin

```

Execute make, do not try to use multiple compilations threads as this fails.

make

Installation is manual and consists of copying the 3 binaries to the folder that will be added to the \$PATH:

```
$> rsync -av bin/ /shared/software/atomistic/vasp/6.2.1_intel19/bin/
```

Testsuite can be run by going into testsuite and running:

```
$> cd testsuite
$> ./runtest
```

6.10.8 ELK

ELK 8.3.22 on Spruce

Modules used:

```
$> module load lang/gcc/10.3.0 libs/openblas/0.3.19_gcc103 lang/python/cpython_3.10.2_-
  ↲gcc103 \
atomistic/wannier90/3.1.0_gcc103 libs/fftw/3.3.10_gcc103 parallel/openmpi/4.0.7_gcc103 \
libs/libxc/5.2.2_gcc103
```

The file `make.inc`:

```

MAKE = make
F90 = gfortran
F90_OPTS = -O3 -ffast-math -funroll-loops -fopenmp -fallow-argument-mismatch
F77 = gfortran
F77_OPTS = -O3 -ffast-math -funroll-loops -fopenmp -fallow-argument-mismatch
AR = ar
LIB_SYS =
# LAPACK and BLAS libraries
LIB_LPK = ${MD_OPENBLAS}/lib/libopenblas.a
LIB_FFT = fftlib.a
SRC_OMP =

#-----
SRC_MPI = mpi_stub.f90
# To enable MPI parallelism the MPI version of the Fortran compiler must be
# used. This is usually mpif90 or mpiifort. Uncomment the following lines and
# run 'make clean' followed by 'make'.
F90 = mpif90
F77 = mpif90
SRC_MPI =
#-----

#-----
SRC_MKL = mkl_stub.f90
# To enable MKL parallelism, link with the MKL library then uncomment the
# following line and run 'make clean' followed by 'make'.
#SRC_MKL =
#-----


#-----
SRC_OBLAS = blas_stub.f90
# To enable OpenBLAS parallelism, link with the OpenBLAS library then uncomment
# the following line and run 'make clean' followed by 'make'.
SRC_OBLAS = ${MD_OPENBLAS}/lib/libopenblas.a
#-----


#-----
SRC_BLIS = blis_stub.f90
# To enable BLIS parallelism, link with the BLIS library then uncomment the
# following line and run 'make clean' followed by 'make'.
#SRC_BLIS =
#-----


#-----
SRC_libxc = libxcifc_stub.f90
# To enable Libxc first download and compile version 5.x of the library. Next
# copy the files libxcf90.a and libxc.a to the elk/src directory. Then uncomment
# the following lines and run 'make clean' followed by 'make'.
LIB_libxc = ${MD_LIBXC}/lib/libxcf90.a ${MD_LIBXC}/lib/libxc.a
SRC_libxc = libxcf90.f90 libxcifc.f90
#-----


#-----

```

(continues on next page)

(continued from previous page)

```

SRC_FFT = zfftifc.f90
# To use a different FFT library, copy the relevant library or include files to
# the elk/src directory (eg. mkl_dfti.f90), uncomment the appropriate lines
# below and run 'make clean' followed by 'make'.
SRC_FFT = zfftifc_fftw.f90
#LIB_FFT = libfftw3.a
LIB_FFT = ${MD_FFTW}/lib/libfftw3.a
#SRC_FFT = mkl_dfti.f90 zfftifc_mkl.f90
#-----

#-----
SRC_W90S = w90_stub.f90
# To enable the Wannier90 library copy libwannier.a to the elk/src directory.
# Then uncomment the following lines and run 'make clean' followed by 'make'.
SRC_W90S =
LIB_W90 = ${MD_WANNIER90}/lib/libwannier.a
#-----
```

Execute make to compile the codes:

```
$> make
```

Installation folder is manually created and the binaries are copied there:

```

$> mkdir -p /shared/software/atomistic/elk/8.3.22_gcc103/bin
$> rsync -av src/eos/eos src/spacegroup/spacegroup src/protex src/elk src/rmspaces src/
  vimelk /shared/software/atomistic/elk/8.3.22_gcc103/bin
$> rsync -av species /shared/software/atomistic/elk/8.3.22_gcc103
```

6.11 Big Data

6.11.1 What is Hadoop

Hadoop is an open-source software package that utilizes large scale data processing by using the Hadoop Distributed File System (HDFS) to provide fast bandwidth of large data across clusters. Using the Hadoop interface, it is possible to split large data across both nodes and processors to increase data processing time. Hadoop's advantage is processing large data files (gigabytes to terabyte range), in a non-POSIX compliant filesystem to increase throughput performance. Hadoop is currently installed on the Spruce Knob cluster.

6.11.2 Using Hadoop on the cluster

Running Hadoop jobs are very similar to [running other jobs on the cluster](#). You will run all of your commands within a PBS job script. However, before you run Hadoop commands, you need to start the Hadoop datanode. To do this, you need to load a few module files and run a few start-up scripts. In addition a specific PBS directive (PBS -W) needs to be specified so Hadoop runs properly:

```

#PBS -W x=rmatchpolicy:exactnode

module load compilers/java/jre1.8.0
```

(continues on next page)

(continued from previous page)

```
module load libraries/DFS/myhadoop
source $MH_HOME/etc/spruce_default.conf
myhadoop-configure.sh
start-all.sh
```

Below these lines, you can place your intended hadoop commands. At the end of your script, you need to stop the hadoop datanode:

```
stop-all.sh
myhadoop-cleanup.sh
```

Both of these commands will end the Hadoop backend. Hadoop commands after the cleanup scripts will result in errors. Here is an example Hadoop script that will run on 3 nodes with 1 processor a node. The example is a wordcount program that is distributed with Hadoop that counts the presence of all words in small sample of Shakespeare playwrights taken from Project Gutenberg.

```
#!/bin/bash

#PBS -q standby
#PBS -N hadoop_job
#PBS -l nodes=3:ppn=1
#PBS -o hadoop_run.out
#PBS -e hadoop_run.err
#PBS -W x=nmatchpolicy:exactnode
#PBS -V

module load compilers/java/jre1.8.0
module load libraries/DFS/myhadoop

# Configure myHadoop environment
source $MH_HOME/etc/spruce_default.conf
myhadoop-configure.sh

# Start Hadoop Cluster
start-all.sh

##### Run your jobs here
echo "Run some test Hadoop jobs"
hadoop dfs -mkdir Data
hadoop dfs -copyFromLocal $HADOOP_HOME/data/gutenberg Data
hadoop dfs -ls Data/gutenberg
hadoop jar /shared/software/myhadoop/hadoop-1.2.1/hadoop-examples-1.2.1.jar wordcount_
↳ Data/gutenberg Outputs
hadoop dfs -ls Outputs
echo

##### Stop the Hadoop cluster
stop-all.sh
```

(continues on next page)

(continued from previous page)

```
#### Cleanup myHadoop configurations  
myhadoop-cleanup.sh  
  
exit 0
```

6.11.3 R Interface with Hadoop - Rhipe

Running large datasets within R is difficult since R loads all data into memory. However, using Hadoop with R can overcome that limitation. To use the R package Rhipe, after setting up the hadoop datanode, you can load the two modules for the dynamically shared R binary and the Rhipe R package:

```
module load compilers/R/3.1.0  
module load libraries/R/Rhipe
```

Note: Rhipe must be loaded after the R module - otherwise you will get a pre-requisite error.

6.11.4 Further Information

Useful Webpages

- Apache Hadoop
- Myhadoop Sourceforge
- Rhipe

Useful Books

Hadoop: The Definitive Guide. Tom White. ISBN: 978-1-4493-1152-0

Parallel R. Q.E. McCallum and S. Watson. ISBN: 978-1-4493-0992-3

6.12 Python 3.9.7

Instructions to compile Python on both clusters are very similar. We will follow generic instructions for any Python 3.9.x version. The latest version at the time of writing this document (August 30, 2021) is 3.9.7. A few particularities related to architecture will be mentioned in due place.

6.12.1 Modules

We will compile Python 3.9 using GCC 9.3 which is a relatively recent version compared to those provided by RedHat 6.x and 7.x (4.4 and 4.7 respectively).

6.12.2 Download

Download the sources on the canonical location for sources /shared/src. Uncompress and move into the folder:

```
$> wget https://www.python.org/ftp/python/3.9.7/Python-3.9.7.tgz
$> tar -zxvf Python-3.9.7.tgz
$> cd Python-3.9.7
```

6.12.3 Configure

Edit the file `setup.py` from the base source dir. Add a line pointing to the include folder of the corresponding GCC version:

```
def detect_sqlite(self):
    # The sqlite interface
    sqlite_setup_debug = False    # verbose debug prints from this script?

    # We hunt for #define SQLITE_VERSION "n.n.n"
    # We need to find >= sqlite version 3.3.9, for sqlite3_prepare_v2
    sqlite_incdir = sqlite_libdir = None
    sqlite_inc_paths = [ '/shared/software/lang/gcc/9.3.0/include',
                         '/usr/include',
                         '/usr/include/sqlite',
                         '/usr/include/sqlite3',
                         '/usr/local/include',
                         '/usr/local/include/sqlite',
                         '/usr/local/include/sqlite3',
                         ]
```

Configure Python 3.9.x enabling optimizations, shared libraries, and setting the prefix for installation:

```
$> ./configure --enable-shared --enable-optimizations \
--prefix=/shared/software/lang/python/3.9.7_gcc93 \
--with-tcltk-libs="-L/shared/software/lang/gcc/9.3.0/lib -ltcl8.6 -ltk8.6" \
--with-tcltk-includes="-I/shared/software/lang/gcc/9.3.0/include" \
--with-lto LDFLAGS="-L${MD_GCC}/lib"
```

Compile and install:

```
$> make
$> make install
```

The test suite:

```
$> make test
```

The results:

```
Ran 3 tests in 0.010s

FAILED (failures=2, skipped=1)
test test_pathlib failed
0:02:28 load avg: 1.97 Re-running test_zipfile in verbose mode (matching: test_add_file_
after_2107)
```

(continues on next page)

(continued from previous page)

```

test_add_file_after_2107 (test.test_zipfile.StoredTestsWithSourceFile) ... ERROR
=====
ERROR: test_add_file_after_2107 (test.test_zipfile.StoredTestsWithSourceFile)
-----
Traceback (most recent call last):
  File "/gpfs20/shared/src/Python-3.9.7/Lib/test/test_zipfile.py", line 618, in test_add_
-> file_after_2107
    os.utime(TESTFN, (ts, ts))
OSError: [Errno 75] Value too large for defined data type
-----
Ran 1 test in 0.002s

FAILED (errors=1)
test test_zipfile failed
2 tests failed again:
  test_pathlib test_zipfile

== Tests result: FAILURE then FAILURE ==

409 tests OK.

2 tests failed:
  test_pathlib test_zipfile

14 tests skipped:
  test_devpoll test_gdb test_ioctl test_kqueue test_msilib
  test_ossaudiodev test_startfile test_tix test_tk test_ttk_guionly
  test_winconsoleio test_winreg test_winsound test_zipfile64

Total duration: 2 min 28 sec
Tests result: FAILURE then FAILURE
make: *** [Makefile:1209: test] Error 2

```

6.12.4 Numpy and Scipy

Download the latest versions of Numpy and Scipy:

```

wget https://github.com/numpy/numpy/releases/download/v1.21.2/numpy-1.21.2.tar.gz
wget https://github.com/scipy/scipy/releases/download/v1.7.1/scipy-1.7.1.tar.gz

```

On each folder, create a file `site.cfg` with the following lines:

```

[openblas]
libraries = openblas
library_dirs = /shared/software/libs/openblas/0.3.13_gcc93/lib
include_dirs = /shared/software/libs/openblas/0.3.13_gcc93/include
runtime_library_dirs = /shared/software/libs/openblas/0.3.13_gcc93/lib

```

Load the module:

```
module load libs/openblas/0.3.13_gcc93
```

Install dependencies:

```
python3 -m pip install pybind11 Cython
```

Install numpy with:

```
python3 -m pip install -v --compile -v .
```

Similarly inside Scipy folder:

```
python3 -m pip install -v --compile -v .
```

Install a set:

```
python3 -m pip install pandas pyyaml networkx seaborn matplotlib ipython jupyter
    ↪ jupyterlab scikit-learn
```

6.13 Matlab N-D multithreaded matrix operations (MMX)

MMX is a Matlab plugin for N-D multithreaded matrix operations. The File Exchange page for it is:

<https://www.mathworks.com/matlabcentral/fileexchange/37515-mm>?ue

The source code is hosted on Github:

<https://github.com/yuvaltassa/mm>

6.13.1 Download the code

The original code can be downloaded with the command:

```
git clone https://github.com/yuvaltassa/mm.git
```

The original instructions for installing MMX does not work in general. MKL path is hardcoded and the name of the static MKL library is wrong. These instructions were created to help others to compile this Matlab plugin on Thorny Flat. That means that the names of environment modules and path for Intel MKL are specific for this cluster. However, the instructions are general enough so other people can compile this code on other computers, regardless of having superusers privileges or not.

We start by getting access to Matlab 2021a, the latest version installed on Thorny. Thorny Flat runs RHEL 7.x and the version of GCC is old (4.8), we have GCC 9.3 as the stable compiler on this cluster:

```
module load lang/gcc/9.3.0 matlab/2021a
```

Other Linux machines could have recent versions of GCC supported by Matlab see this webpage to check which compilers are accepted.

<https://www.mathworks.com/support/requirements/supported-compilers.html>

Recent versions of Matlab support GCC up to 9.x

Download the code with the git command mentioned above:

```
git clone https://github.com/yuvaltassa/mmx.git
```

Go to the folder `mmx/src`:

```
cd mmx/src/
```

Enter in Matlab:

```
matlab
```

Execute the command `build_mmx()` (This command will fail):

```
>> build_mmx()

Compilation of 'mmx_mkl_single' failed with error:
ls: cannot access *small_mkl_ilp64.*: No such file or directory
=====

Trying to compile 'mmx_mkl_multi', using
CXXFLAGS="\$CXXFLAGS -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 ", -
-lmwblas, -lmwlapack, -lpthread, -DUNIX_SYSTEM, -DUSE_BLAS,
Compilation of 'mmx_mkl_multi' failed with error:
Access denied; check that you have permissions to access '/gpfs20/shared/software/matlab/
r2021a/extern/lib/glnxa64'.
=====

Trying to compile 'mmx_naive', using
-lpthread, -DUNIX_SYSTEM,
Compilation of 'mmx_naive' failed with error:
Access denied; check that you have permissions to access '/gpfs20/shared/software/matlab/
r2021a/extern/lib/glnxa64'.
>>
```

6.13.2 Edit the building script

Edit the file `build_mmx.m`, adjust several lines of code to make it work. The differences are:

```
$ diff -u build_mmx.m /shared/software/matlab/r2021a/extern/lib/glnxa64/build_mmx.m
--- build_mmx.m 2022-01-10 10:22:56.969414000 -0500
+++ /shared/software/matlab/r2021a/extern/lib/glnxa64/build_mmx.m      2022-01-09.
-          10:19:34.160261000 -0500
@@ -126,7 +126,7 @@
              link      = {'mwblas','mwlapack','pthread'};
              define   = {'UNIX_SYSTEM','USE_BLAS'};
              case 'mmx_mkl_single'
-              root = '/opt/intel/mkl';
+              root = '/shared/software/intel/2021.4/mkl/2021.4.0';
                  inc_dir = [root '/include'];
                  if strcmp(arch,'glnx86')
                      link_dir = [matlabroot '/extern/lib/glnx86'];
@@ -134,7 +134,7 @@
              define   = {'UNIX_SYSTEM', 'USE_BLAS', 'MKL_32'};
              else
```

(continues on next page)

(continued from previous page)

```

link_dir = [matlabroot '/extern/lib/glnxa64'];
-
link      = {'small_mkl_ilp64','pthread'};
+
link      = {'single_mkl_ilp64','pthread'};
define   = {'UNIX_SYSTEM', 'USE_BLAS', 'MKL_ILP64'};
end
end

```

6.13.3 Build a custom MKL library

Build the custom MKL library, first load module for Intel MKL:

```
module load mkl
```

Choose a folder to build the MKL custom library, it could be the same mmx/src folder and we will use it here Concatenate the list of Linear Algebra routines BLAS and LAPACK from MKL:

```
cat ${MKLROOT}/tools/builderblas_example_list > blas_lapack_list
cat ${MKLROOT}/tools/builderlapack_example_list >> blas_lapack_list
```

Copy the makefile from the MKL builder folder:

```
cp ${MKLROOT}/tools/builder/makefile .
```

And build the library from it:

```
make libintel64 interface=ilp64 export=blas_lapack_list name=libsingle_mkl_ilp64
  ↵threading=sequential
```

Verify that you have reach this step correctly by checking for the file libsingl_mkl_ilp64.so

6.13.4 Compile MMX

Now we can compile mmx as a Matlab MEX file

The output is extensive but illustrative of the 3 compilations of MMX that are done. build_mmx will try to compile from the most efficient to the simplest version. The first time it succeeds compiles the “official” mmx MEX library

This is the complete output in verbose mode:

```
$ matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

                                     < M A T L A B (R) >
Copyright 1984-2021 The MathWorks, Inc.
                                         R2021a (9.10.0.1602886) 64-bit.
                                         February 17, 2021

To get started, type doc.
For product information, visit www.mathworks.com.
```

(continues on next page)

(continued from previous page)

```

>> build_mmxx(verbose=true)
Error using build_mmxx
Too many input arguments.

>> build_mmxx(true)

=====
Trying to compile 'mmxx_mkl_single', using
-v, CXXFLAGS="\$CXXFLAGS -I/shared/software/intel/2021.4/mkl/2021.4.0/include ", LDFLAGS=
-> "\$LDFLAGS -L/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 ", -lsingle_mkl_
-> ilp64, -lpthread, -DUNIX_SYSTEM, -DUSE_BLAS, -DMKL_ILP64,
Verbose mode is on.
No MEX options file identified; looking for an implicit selection.
... Looking for compiler 'g++' ...
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Executing command 'g++ -print-file-name=libstdc++.so' ...Yes ('/shared/software/lang/
-> gcc/9.3.0/lib64/libstdc++.so').
... Executing command 'g++ -dumpversion' ...Yes ('9.3.0').
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Looking for folder '/shared/software/lang/gcc/9.3.0' ...Yes.
... Executing command 'g++ -dumpmachine' ...Yes ('x86_64-pc-linux-gnu').
Found installed compiler 'g++'.
Set INCLUDE = /shared/software/lang/gcc/9.3.0/lib/gcc/x86_64-pc-linux-gnu/9.3.0/include;/
-> shared/software/lang/gcc/9.3.0/include/c++/9.3.0;/shared/software/lang/gcc/9.3.0/
-> include/c++/9.3.0/x86_64-pc-linux-gnu;/shared/software/lang/gcc/9.3.0/include/c++/9.3.
-> 0/backward;/shared/software/lang/gcc/9.3.0/include
Options file details
-----
Compiler location: /shared/software/lang/gcc/9.3.0/bin/g++
Options file: /gpfs20/shared/software/matlab/r2021a/bin/glnxa64/mexopts/g++_glnxa64.
-> xml
CMDLINE2 : /shared/software/lang/gcc/9.3.0/bin/g++ \pthread -Wl,--no-undefined -L/
-> gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-
-> script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
-> exportsmexfileversion.map" /tmp/mex_59078790081513512_46067/mmxx.o /tmp/mex_
-> 59078790081513512_46067/cpp_mexapi_version.o -lsingle_mkl_ilp64 -lpthread -
-> lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/
-> glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/
-> shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
-> r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmex -lm -lmat -o /gpfs20/users/
-> vh0010/mmxx/src/mmxx_mkl_single.mexa64
CXX : /shared/software/lang/gcc/9.3.0/bin/g++
DEFINES : -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -
-> DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE
MATLABMEX : -DMATLAB_MEX_FILE
CFLAGS : -fexceptions -fPIC -fno-omit-frame-pointer -pthread
CXXFLAGS : \-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/shared/
-> software/intel/2021.4/mkl/2021.4.0/include
INCLUDE : -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/
-> software/matlab/r2021a/simulink/include"
CXXOPTIMFLAGS : -O2 -fwrapv -DNDEBUG

```

(continues on next page)

(continued from previous page)

```

CXXDEBUGFLAGS : -g
LDXX : /shared/software/lang/gcc/9.3.0/bin/g++
LDFLAGS : \-pthread -Wl,--no-undefined -L/gpfs20/shared/software/matlab/r2021a/
↳extern/lib/glnxa64
LDTYPE : -shared
LINKEXPORT : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/
↳glnxa64/mexFunction.map"
LINKEXPORTVER : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/
↳lib/glnxa64/c_exports_mexfileversion.map"
LINKLIBS : -lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/-
↳gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
↳r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/
↳glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray_
↳-lmx -lmex -lm -lmat
LDOPTIMFLAGS : -O
LDDEBUGFLAGS : -g
MWCPLIB : "/gpfs20/shared/software/matlab/r2021a/sys/os/glnxa64/orig/libstdc++.so.6"
OBJEXT : .o
LDEXT : .mexa64
SETENV : CC="gcc"
          CXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          CFLAGS="-fexceptions -fPIC -fno-omit-frame-pointer -pthread -DUNIX_
↳SYSTEM -DUSE_BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_
↳GNU_SOURCE -DMATLAB_MEX_FILE "
          CXFLAGS="\-fexceptions -fPIC -fno-omit-frame-pointer -pthread -
          std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -DUNIX_SYSTEM -DUSE_
          BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -
          DMATLAB_MEX_FILE "
          COPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CXXOPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CDEBUGFLAGS="-g"
          CXDEBUGFLAGS="-g"
          MW_GLIBC_SHIM="$MW_GLIBC_SHIM"
          LD="gcc"
          LDXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          LDFLAGS="\-pthread -Wl,--no-undefined -L/gpfs20/shared/software/matlab/
          r2021a/extern/lib/glnxa64 -shared -lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-
          needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/
          shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/
          matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/
          glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -Wl,--version-script,"/gpfs20/shared/
          software/matlab/r2021a/extern/lib/glnxa64/mexFunction.map"
          LDDEBUGFLAGS="-g"
GCC : /shared/software/lang/gcc/9.3.0/bin/g++
CPPLIB_DIR : /shared/software/lang/gcc/9.3.0/lib64/libstdc++.so
VER : 9.3.0
GCCDIR : /shared/software/lang/gcc/9.3.0
GCC_TARGET : x86_64-pc-linux-gnu
MATLABROOT : /gpfs20/shared/software/matlab/r2021a
ARCH : glnxa64
SRC : "/gpfs20/users/vh0010/mmx/src/mmx.cpp";"/gpfs20/shared/software/matlab/r2021a/
↳extern/version/cpp_mexapi_version.cpp"

```

(continues on next page)

(continued from previous page)

```

OBJ : /tmp/mex_59078790081513512_46067/mmx.o;/tmp/mex_59078790081513512_46067/cpp_-
↳ mexapi_version.o
OBJS : /tmp/mex_59078790081513512_46067/mmx.o /tmp/mex_59078790081513512_46067/cpp_-
↳ mexapi_version.o
SRCROOT : /gpfs20/users/vh0010/mmx/src/mmx
DEF : /tmp/mex_59078790081513512_46067/mmx_mkl_single.def
EXP : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_single.exp"
LIB : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_single.lib"
EXE : /gpfs20/users/vh0010/mmx/src/mmx_mkl_single.mexa64
ILK : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_single.ilk"
MANIFEST : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_single.mexa64.manifest"
TEMPNAME : /gpfs20/users/vh0010/mmx/src/mmx_mkl_single
EXEDIR : /gpfs20/users/vh0010/mmx/src/
EXENAME : mmx_mkl_single
OPTIM : -O2 -fwrapv -DNDEBUG
LINKOPTIM : -O
CMDLINE1_0 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS - -
↳ DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_-
↳ MEX_FILE -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/-
↳ software/matlab/r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer - -
↳ pthread -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv - -
↳ DNDEBUG "/gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078790081513512_46067/mmx.-
↳ o
CMDLINE1_1 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS - -
↳ DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_-
↳ MEX_FILE -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/-
↳ software/matlab/r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer - -
↳ pthread -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv - -
↳ DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" - -
↳ o /tmp/mex_59078790081513512_46067/cpp_mexapi_version.o
-----
Building with 'g++'.
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 - -
↳ DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
↳ gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
↳ r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread - -
↳ std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
↳ gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078790081513512_46067/mmx.o
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 - -
↳ DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
↳ gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
↳ r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread - -
↳ std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
↳ gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/
↳ mex_59078790081513512_46067/cpp_mexapi_version.o
/shared/software/lang/gcc/9.3.0/bin/g++ \pthread -Wl,--no-undefined -L/gpfs20/shared/
↳ software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-script,"/gpfs20/
↳ shared/software/matlab/r2021a/extern/lib/glnxa64/c_exports_mexfileversion.map" /tmp/mex_-
↳ 59078790081513512_46067/mmx.o /tmp/mex_59078790081513512_46067/cpp_mexapi_version.o - -
↳ lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/
↳ software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/
↳ shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx (continues on next page)
↳ lmat -o /gpfs20/users/vh0010/mmx/src/mmx_mkl_single.mexa64

```

(continued from previous page)

```

Recompile embedded version with '-DMATLAB_MEXCMD_RELEASE=R2017b'
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -
-DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
r2021a/simulink/include" \ -fexceptions -fPIC -fno-omit-frame-pointer -pthread -
-std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/
mex_59078790081513512_46067/cpp_mexapi_version.o -DMATLAB_MEXCMD_RELEASE=R2017b
/shared/software/lang/gcc/9.3.0/bin/g++ \ -pthread -Wl,--no-undefined -L/gpfs20/shared/
software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-script,"/gpfs20/
shared/software/matlab/r2021a/extern/lib/glnxa64/c_exportsmexfileversion.map" /tmp/mex_
59078790081513512_46067/mmx.o /tmp/mex_59078790081513512_46067/cpp_mexapi_version.o -
-lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/
software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64
-Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/
shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -
-lmat -o /gpfs20/users/vh0010/mmx/src/mmx_mkl_single.mexa64
MEX completed successfully.
Compilation of 'mmx_mkl_single' succeeded.
Compiling again to 'mmx' target using 'mmx_mkl_single' build.
Verbose mode is on.
... Looking for compiler 'g++' ...
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Executing command 'g++ -print-file-name=libstdc++.so' ...Yes ('/shared/software/lang/
gcc/9.3.0/lib64/libstdc++.so').
... Executing command 'g++ -dumpversion' ...Yes ('9.3.0').
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Looking for folder '/shared/software/lang/gcc/9.3.0' ...Yes.
... Executing command 'g++ -dumpmachine' ...Yes ('x86_64-pc-linux-gnu').
Found installed compiler 'g++'.
Set INCLUDE = /shared/software/lang/gcc/9.3.0/lib/gcc/x86_64-pc-linux-gnu/9.3.0/include;/
shared/software/lang/gcc/9.3.0/include/c++/9.3.0;/shared/software/lang/gcc/9.3.0/
include/c++/9.3.0/x86_64-pc-linux-gnu;/shared/software/lang/gcc/9.3.0/include/c++/9.3.
0/backward;/shared/software/lang/gcc/9.3.0/include
Options file details
-----
    Compiler location: /shared/software/lang/gcc/9.3.0/bin/g++
    Options file: /gpfs20/shared/software/matlab/r2021a/bin/glnxa64/mexopts/g++_glnxa64.
    xml
    CMDLINE2 : /shared/software/lang/gcc/9.3.0/bin/g++ \ -pthread -Wl,--no-undefined -L/
gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-
script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
exportsmexfileversion.map" /tmp/mex_59078796776035188_46067/mmx.o /tmp/mex_
59078796776035188_46067/cpp_mexapi_version.o -lsingle_mkl_ilp64 -lpthread -
lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/
glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/
shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lmat -o /gpfs20/users/
vh0010/mmx/src/mmx.mexa64
    CXX : /shared/software/lang/gcc/9.3.0/bin/g++
    DEFINES : -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -
DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE

```

(continues on next page)

(continued from previous page)

```

MATLABMEX : -DMATLAB_MEX_FILE
CFLAGS : -fexceptions -fPIC -fno-omit-frame-pointer -pthread
CXXFLAGS : \-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/shared/
software/intel/2021.4/mkl/2021.4.0/include
INCLUDE : -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/
software/matlab/r2021a/simulink/include"
CXXOPTIMFLAGS : -O2 -fwrapv -DNDEBUG
CXXDEBUGFLAGS : -g
LDXX : /shared/software/lang/gcc/9.3.0/bin/g++
LDFLAGS : \-pthread -Wl,--no-undefined -L/gpfs20/shared/software/matlab/r2021a/
extern/lib/glnxa64
LDTYPE : -shared
LINKEXPORT : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/
glnxa64/mexFunction.map"
LINKEXPORTVER : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/
lib/glnxa64/c_exports_mexfileversion.map"
LINKLIBS : -lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/-
gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/
glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray_
-lmx -lmex -lm -lmat
LDOPTIMFLAGS : -O2
LDDEBUGFLAGS : -g
MWCPLIB : "/gpfs20/shared/software/matlab/r2021a/sys/os/glnxa64/orig/libstdc++.so.6"
OBJEXT : .o
LDEXT : .mexa64
SETENV : CC="gcc"
          CXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          CFLAGS="-fexceptions -fPIC -fno-omit-frame-pointer -pthread -DUNIX_
SYSTEM -DUSE_BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_
GNU_SOURCE -DMATLAB_MEX_FILE "
          CXXFLAGS="\-fexceptions -fPIC -fno-omit-frame-pointer -pthread -
std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -DUNIX_SYSTEM -DUSE_
BLAS -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -
DMATLAB_MEX_FILE "
          COPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CXXOPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CDEBUGFLAGS="-g"
          CXXDEBUGFLAGS="-g"
          MW_GLIBC_SHIM="$MW_GLIBC_SHIM"
          LD="gcc"
          LDXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          LDFLAGS="\-pthread -Wl,--no-undefined -L/gpfs20/shared/software/matlab/
r2021a/extern/lib/glnxa64 -shared -lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-
needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/
shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/
matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/
glnxa64" -lMatlabdataArray -lmex -lm -lmat -Wl,--version-script,"/gpfs20/shared/
software/matlab/r2021a/extern/lib/glnxa64/mexFunction.map"""
          LDDEBUGFLAGS="-g"
GCC : /shared/software/lang/gcc/9.3.0/bin/g++
CPPLIB_DIR : /shared/software/lang/gcc/9.3.0/lib64/libstdc++.so

```

(continues on next page)

(continued from previous page)

```

VER : 9.3.0
GCCDIR : /shared/software/lang/gcc/9.3.0
GCC_TARGET : x86_64-pc-linux-gnu
MATLABROOT : /gpfs20/shared/software/matlab/r2021a
ARCH : glnxa64
SRC : "/gpfs20/users/vh0010/mmx/src/mmx.cpp";"/gpfs20/shared/software/matlab/r2021a/
↳ extern/version/cpp_mexapi_version.cpp"
OBJ : /tmp/mex_59078796776035188_46067/mmx.o;/tmp/mex_59078796776035188_46067/cpp_
↳ mexapi_version.o
OBJS : /tmp/mex_59078796776035188_46067/mmx.o /tmp/mex_59078796776035188_46067/cpp_
↳ mexapi_version.o
SRCROOT : /gpfs20/users/vh0010/mmx/src/mmx
DEF : /tmp/mex_59078796776035188_46067/mmx.def
EXP : "/gpfs20/users/vh0010/mmx/src/mmx.exp"
LIB : "/gpfs20/users/vh0010/mmx/src/mmx.lib"
EXE : /gpfs20/users/vh0010/mmx/src/mmx.mexa64
ILK : "/gpfs20/users/vh0010/mmx/src/mmx.ilk"
MANIFEST : "/gpfs20/users/vh0010/mmx/src/mmx.mexa64.manifest"
TEMPNAME : /gpfs20/users/vh0010/mmx/src/mmx
EXEDIR : /gpfs20/users/vh0010/mmx/src/
EXENAME : mmx
OPTIM : -O2 -fwrapv -DNDEBUG
LINKOPTIM : -O
CMDLINE1_0 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -
↳ -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_
↳ _MEX_FILE -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/
↳ software/matlab/r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -
↳ -pthread -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -
↳ -DNDEBUG "/gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078796776035188_46067/mmx.
↳ o
CMDLINE1_1 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -
↳ -DMKL_ILP64 -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_
↳ _MEX_FILE -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/
↳ software/matlab/r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -
↳ -pthread -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -
↳ -DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -
↳ o /tmp/mex_59078796776035188_46067/cpp_mexapi_version.o
-----
Building with 'g++'.
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -
↳ -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
↳ gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
↳ r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -
↳ -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
↳ gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078796776035188_46067/mmx.o
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -
↳ -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
↳ gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
↳ r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -
↳ -std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
↳ gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/
↳ mex_59078796776035188_46067/cpp_mexapi_version.o

```

(continues on next page)

(continued from previous page)

```

/shared/software/lang/gcc/9.3.0/bin/g++ \-pthread -Wl,--no-undefined -L/gpfs20/shared/
↳ software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-script,"/gpfs20/
↳ shared/software/matlab/r2021a/extern/lib/glnxa64/c_exports_mexfileversion.map" /tmp/mex_
↳ 59078796776035188_46067/mmx.o /tmp/mex_59078796776035188_46067/cpp_mexapi_version.o -
↳ lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/
↳ software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64
↳ " -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/
↳ shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -
↳ lmat -o /gpfs20/users/vh0010/mmx/src/mmx.mexa64
Recompile embedded version with '-DMATLAB_MEXCMD_RELEASE=R2017b'
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMKL_ILP64 -
↳ DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
↳ gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
↳ r2021a/simulink/include" \-fexceptions -fPIC -fno-omit-frame-pointer -pthread -
↳ std=c++11 -I/shared/software/intel/2021.4/mkl/2021.4.0/include -O2 -fwrapv -DNDEBUG "/
↳ gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/
↳ mex_59078796776035188_46067/cpp_mexapi_version.o -DMATLAB_MEXCMD_RELEASE=R2017b
/shared/software/lang/gcc/9.3.0/bin/g++ \-pthread -Wl,--no-undefined -L/gpfs20/shared/
↳ software/matlab/r2021a/extern/lib/glnxa64 -shared -O -Wl,--version-script,"/gpfs20/
↳ shared/software/matlab/r2021a/extern/lib/glnxa64/c_exports_mexfileversion.map" /tmp/mex_
↳ 59078796776035188_46067/mmx.o /tmp/mex_59078796776035188_46067/cpp_mexapi_version.o -
↳ lsingle_mkl_ilp64 -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/
↳ software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64
↳ " -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/
↳ shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -
↳ lmat -o /gpfs20/users/vh0010/mmx/src/mmx.mexa64
MEX completed successfully.
=====
```

```

Trying to compile 'mmx_mkl_multi', using
-v, CXXFLAGS="\$CXXFLAGS -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 ", -
↳ lmwblas, -lmwlapack, -lpthread, -DUNIX_SYSTEM, -DUSE_BLAS,
Verbose mode is on.
... Looking for compiler 'g++' ...
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Executing command 'g++ -print-file-name=libstdc++.so' ...Yes ('/shared/software/lang/
↳ gcc/9.3.0/lib64/libstdc++.so').
... Executing command 'g++ -dumpversion' ...Yes ('9.3.0').
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Looking for folder '/shared/software/lang/gcc/9.3.0' ...Yes.
... Executing command 'g++ -dumpmachine' ...Yes ('x86_64-pc-linux-gnu').
Found installed compiler 'g++'.
Set INCLUDE = /shared/software/lang/gcc/9.3.0/lib/gcc/x86_64-pc-linux-gnu/9.3.0/include;/
↳ shared/software/lang/gcc/9.3.0/include/c++/9.3.0;/shared/software/lang/gcc/9.3.0/
↳ include/c++/9.3.0/x86_64-pc-linux-gnu;/shared/software/lang/gcc/9.3.0/include/c++/9.3.
↳ 0/backward;/shared/software/lang/gcc/9.3.0/include
Options file details
-----
```

```

Compiler location: /shared/software/lang/gcc/9.3.0/bin/g++
Options file: /gpfs20/shared/software/matlab/r2021a/bin/glnxa64/mexopts/g++_glnxa64.
↳ xml
CMDLINE2 : /shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -
↳ shared -O -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/
↳ glnxa64/c_exports_mexfileversion.map" /tmp/mex_59078799484017262_46067/mmx.o /tmp/mex_
↳ 59078799484017262_46067/cpp_mexapi_version.o -lmwblas -lmwlapack -lpthread -
↳ lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/
292 ↳ glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -Wl,-rpath-link,/gpfs20/
↳ shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
↳ r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -o /gpfs20/users/
↳ vh0010/mmx/src/mmx_mkl_multi.mexa64
-----
```

(continued from previous page)

```

CXX : /shared/software/lang/gcc/9.3.0/bin/g++
DEFINES : -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE
MATLABMEX : -DMATLAB_MEX_FILE
CFLAGS : -fexceptions -fPIC -fno-omit-frame-pointer -pthread
CXXFLAGS : \-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64
INCLUDE : -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/simulink/include"
CXXOPTIMFLAGS : -O2 -fwrapv -DNDEBUG
CXXDEBUGFLAGS : -g
LDXX : /shared/software/lang/gcc/9.3.0/bin/g++
LDFLAGS : -pthread -Wl,--no-undefined
LDTYPE : -shared
LINKEXPORT : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/mexFunction.map"
LINKEXPORTVER : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_exports_mexfileversion.map"
LINKLIBS : -lmwblas -lmwlapack -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat
LDOPTIMFLAGS : -O2
LDDEBUGFLAGS : -g
MWCPLIB : "/gpfs20/shared/software/matlab/r2021a/sys/os/glnxa64/orig/libstdc++.so.6"
OBJEXT : .o
LDEXT : .mexa64
SETENV : CC="gcc"
          CXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          CFLAGS="-fexceptions -fPIC -fno-omit-frame-pointer -pthread -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE"
          CXXFLAGS="\-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE"
          COPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CXXOPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
          CDEBUGFLAGS="-g"
          CXXDEBUGFLAGS="-g"
          MW_GLIBC_SHIM="$MW_GLIBC_SHIM"
          LD="gcc"
          LDXX="/shared/software/lang/gcc/9.3.0/bin/g++"
          LDFLAGS="-pthread -Wl,--no-undefined -shared -lmwblas -lmwlapack -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -Wl,-version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/mexFunction.map"
          LDDEBUGFLAGS="-g"

```

(continues on next page)

(continued from previous page)

```

GCC : /shared/software/lang/gcc/9.3.0/bin/g++
CPPLIB_DIR : /shared/software/lang/gcc/9.3.0/lib64/libstdc++.so
VER : 9.3.0
GCCDIR : /shared/software/lang/gcc/9.3.0
GCC_TARGET : x86_64-pc-linux-gnu
MATLABROOT : /gpfs20/shared/software/matlab/r2021a
ARCH : glnxa64
SRC : "/gpfs20/users/vh0010/mmx/src/mmx.cpp";"/gpfs20/shared/software/matlab/r2021a/
`-extern/version/cpp_mexapi_version.cpp"
OBJ : /tmp/mex_59078799484017262_46067/mmx.o;/tmp/mex_59078799484017262_46067/cpp-
`-mexapi_version.o
OBJS : /tmp/mex_59078799484017262_46067/mmx.o /tmp/mex_59078799484017262_46067/cpp-
`-mexapi_version.o
SRCROOT : /gpfs20/users/vh0010/mmx/src/mmx
DEF : /tmp/mex_59078799484017262_46067/mmx_mkl_multi.def
EXP : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_multi.exp"
LIB : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_multi.lib"
EXE : /gpfs20/users/vh0010/mmx/src/mmx_mkl_multi.mexa64
ILK : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_multi.ilk"
MANIFEST : "/gpfs20/users/vh0010/mmx/src/mmx_mkl_multi.mexa64.manifest"
TEMPNAME : /gpfs20/users/vh0010/mmx/src/mmx_mkl_multi
EXEDIR : /gpfs20/users/vh0010/mmx/src/
EXENAME : mmx_mkl_multi
OPTIM : -O2 -fwrapv -DNDEBUG
LINKOPTIM : -O
CMDLINE1_0 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -
`-DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
`-gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
`-r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -
`-std=c++11 -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -O2 -fwrapv -
`-DNDEBUG "/gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078799484017262_46067/mmx.
`-o
CMDLINE1_1 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -
`-DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/
`-gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/
`-r2021a/simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -
`-std=c++11 -I/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -O2 -fwrapv -
`-DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -
`-o /tmp/mex_59078799484017262_46067/cpp_mexapi_version.o
-----
Building with 'g++'.
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_
`-RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
`-software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
`-simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/
`-gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -O2 -fwrapv -DNDEBUG "/gpfs20/
`-users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078799484017262_46067/mmx.o
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_
`-RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
`-software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
`-simulink/include" \fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/
`-gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -O2 -fwrapv -DNDEBUG "/gpfs20/
`-shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/mex_59078799484017262_46067/cpp_mexapi_version.o
`-59078799484017262_46067/cpp_mexapi_version.o

```

(continued from previous page)

```

/shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -shared -O -Wl,--
→version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
→exportsmexfileversion.map" /tmp/mex_59078799484017262_46067/mmxx.o /tmp/mex_
→59078799484017262_46067/cpp_mexapi_version.o -lmwblas -lmwlapack -lpthread -
→lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/
→glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/
→shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
→r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -o /gpfs20/users/
→vh0010/mmxx/src/mmxx_mkl_multi.mexa64

Recompile embedded version with '-DMATLAB_MEXCMD_RELEASE=R2017b'
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DUSE_BLAS -DMATLAB_DEFAULT_-
→RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
→software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
→simulink/include" \-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -I/
→gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64 -O2 -fwrapv -DNDEBUG "/gpfs20/
→shared/software/matlab/r2021a/extern/version/cpp_mexapi_version.cpp" -o /tmp/mex_
→59078799484017262_46067/cpp_mexapi_version.o -DMATLAB_MEXCMD_RELEASE=R2017b

/shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -shared -O -Wl,--
→version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
→exportsmexfileversion.map" /tmp/mex_59078799484017262_46067/mmxx.o /tmp/mex_
→59078799484017262_46067/cpp_mexapi_version.o -lmwblas -lmwlapack -lpthread -
→lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/
→glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/
→shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/
→r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -o /gpfs20/users/
→vh0010/mmxx/src/mmxx_mkl_multi.mexa64

MEX completed successfully.
Compilation of 'mmxx_mkl_multi' succeeded.
=====

Trying to compile 'mmxx_naive', using
-v, -lpthread, -DUNIX_SYSTEM,
Verbose mode is on.
... Looking for compiler 'g++' ...
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Executing command 'g++ -print-file-name=libstdc++.so' ...Yes ('/shared/software/lang/
→gcc/9.3.0/lib64/libstdc++.so').
... Executing command 'g++ -dumpversion' ...Yes ('9.3.0').
... Executing command 'which g++' ...Yes ('/shared/software/lang/gcc/9.3.0/bin/g++').
... Looking for folder '/shared/software/lang/gcc/9.3.0' ...Yes.
... Executing command 'g++ -dumpmachine' ...Yes ('x86_64-pc-linux-gnu').
Found installed compiler 'g++'.
Set INCLUDE = /shared/software/lang/gcc/9.3.0/lib/gcc/x86_64-pc-linux-gnu/9.3.0/include;/
→shared/software/lang/gcc/9.3.0/include/c++/9.3.0;/shared/software/lang/gcc/9.3.0/
→include/c++/9.3.0/x86_64-pc-linux-gnu;/shared/software/lang/gcc/9.3.0/include/c++/9.3.
→0/backward;/shared/software/lang/gcc/9.3.0/include
Options file details
-----

Compiler location: /shared/software/lang/gcc/9.3.0/bin/g++
Options file: /gpfs20/shared/software/matlab/r2021a/bin/glnxa64/mexopts/g++_glnxa64.
→xml
CMDLINE2 : /shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -
→shared -O -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/
→glnxa64/c_exportsmexfileversion.map" /tmp/mex_59078801684538554_46067/mmxx.o /tmp/mex_
→59078801684538554_46067/cpp_mexapi_version.o -lpthread -lstdc++ -Wl,--as-needed -
→Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/
→software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/
→r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64" -
→-lMatlabdataArray -lmx -lmex -lm -lmat -o /gpfs20/users/vh0010/mmxx/src/mmxx_naive.mexa64

```

(continued from previous page)

```

CXX : /shared/software/lang/gcc/9.3.0/bin/g++
DEFINES : -DUNIX_SYSTEM -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_
FILE
SOURCE -DMATLAB_MEX_FILE
MATLABMEX : -DMATLAB_MEX_FILE
CFLAGS : -fexceptions -fPIC -fno-omit-frame-pointer -pthread
CXXFLAGS : -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11
INCLUDE : -I"/gpfs20/shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/
software/matlab/r2021a/simulink/include"
CXXOPTIMFLAGS : -O2 -fwrapv -DNDEBUG
CXXDEBUGFLAGS : -g
LDXX : /shared/software/lang/gcc/9.3.0/bin/g++
LDFLAGS : -pthread -Wl,--no-undefined
LDTYPE : -shared
LINKEXPORT : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/
glnxa64/mexFunction.map"
LINKEXPORTVER : -Wl,--version-script,"/gpfs20/shared/software/matlab/r2021a/extern/
lib/glnxa64/c_exports_mexfileversion.map"
LINKLIBS : -lpthread -lstdc++ -Wl,--as-needed -Wl,-rpath-link,/gpfs20/shared/
software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/bin/glnxa64
-Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/
shared/software/matlab/r2021a/extern/bin/glnxa64" -lMatlabdataArray -lmx -lmex -lm -
lmat
LDOPTIMFLAGS : -O
LDDEBUGFLAGS : -g
MWCPLIB : "/gpfs20/shared/software/matlab/r2021a/sys/os/glnxa64/orig/libstdc++.so.6"
OBJEXT : .o
LDEXT : .mexa64
SETENV : CC="gcc"
CXX="/shared/software/lang/gcc/9.3.0/bin/g++"
CFLAGS="-fexceptions -fPIC -fno-omit-frame-pointer -pthread -DUNIX_
SYSTEM -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_
FILE"
CXXFLAGS="-fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11_
-DUNIX_SYSTEM -DMATLAB_DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_
MEX_FILE"
COPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
CXXOPTIMFLAGS="-O2 -fwrapv -DNDEBUG"
CDEBUGFLAGS="-g"
CXXDEBUGFLAGS="-g"
MW_GLIBC_SHIM="$MW_GLIBC_SHIM"
LD="gcc"
LDXX="/shared/software/lang/gcc/9.3.0/bin/g++"
LDFLAGS="-pthread -Wl,--no-undefined -shared -lpthread -lstdc++ -Wl,--
as-needed -Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/
shared/software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/
matlab/r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/
glnxa64" -lMatlabdataArray -lmx -lmex -lm -lmat -Wl,--version-script,"/gpfs20/shared/
software/matlab/r2021a/extern/lib/glnxa64/mexFunction.map"
LDDEBUGFLAGS="-g"
GCC : /shared/software/lang/gcc/9.3.0/bin/g++
CPPLIB_DIR : /shared/software/lang/gcc/9.3.0/lib64/libstdc++.so
VER : 9.3.0

```

(continues on next page)

(continued from previous page)

```

GCCDIR : /shared/software/lang/gcc/9.3.0
GCC_TARGET : x86_64-pc-linux-gnu
MATLABROOT : /gpfs20/shared/software/matlab/r2021a
ARCH : glnxa64
SRC : "/gpfs20/users/vh0010/mmx/src/mmx.cpp";"/gpfs20/shared/software/matlab/r2021a/
↳ extern/version/cpp_mexapi_version.cpp"
OBJ : /tmp/mex_59078801684538554_46067/mmx.o;/tmp/mex_59078801684538554_46067/cpp-
↳ mexapi_version.o
OBJS : /tmp/mex_59078801684538554_46067/mmx.o /tmp/mex_59078801684538554_46067/cpp-
↳ mexapi_version.o
SRCROOT : /gpfs20/users/vh0010/mmx/src/mmx
DEF : /tmp/mex_59078801684538554_46067/mmx_naive.def
EXP : "/gpfs20/users/vh0010/mmx/src/mmx_naive.exp"
LIB : "/gpfs20/users/vh0010/mmx/src/mmx_naive.lib"
EXE : /gpfs20/users/vh0010/mmx/src/mmx_naive.mexa64
ILK : "/gpfs20/users/vh0010/mmx/src/mmx_naive.ilk"
MANIFEST : "/gpfs20/users/vh0010/mmx/src/mmx_naive.mexa64.manifest"
TEMPNAME : /gpfs20/users/vh0010/mmx/src/mmx_naive
EXEDIR : /gpfs20/users/vh0010/mmx/src/
EXENAME : mmx_naive
OPTIM : -O2 -fwrapv -DNDEBUG
LINKOPTIM : -O
CMDLINE1_0 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DMATLAB_
↳ DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/
↳ shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
↳ simulink/include" -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -O2 -
↳ fwrapv -DNDEBUG "/gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078801684538554_
↳ 46067/mmx.o
CMDLINE1_1 : /shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DMATLAB_
↳ DEFAULT_RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/
↳ shared/software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
↳ simulink/include" -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -O2 -
↳ fwrapv -DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi-
↳ version.cpp" -o /tmp/mex_59078801684538554_46067/cpp_mexapi_version.o
-----
Building with 'g++'.
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DMATLAB_DEFAULT_
↳ RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
↳ software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
↳ simulink/include" -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -O2 -
↳ fwrapv -DNDEBUG "/gpfs20/users/vh0010/mmx/src/mmx.cpp" -o /tmp/mex_59078801684538554_
↳ 46067/mmx.o
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DMATLAB_DEFAULT_
↳ RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
↳ software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
↳ simulink/include" -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -O2 -
↳ fwrapv -DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi-
↳ version.cpp" -o /tmp/mex_59078801684538554_46067/cpp_mexapi_version.o
/shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -shared -O -Wl,--
↳ version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
↳ exportsmexfileversion.map" /tmp/mex_59078801684538554_46067/mmx.o /tmp/mex_
↳ 59078801684538554_46067/cpp_mexapi_version.o -lpthread -lstdc++ -Wl,--as-needed -
↳ Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/
↳ software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/
↳ r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64"
6.13 Matlab N-D multithreaded matrix operations (MMX) 297
↳ 1MatlabDataArray.lmx.lmx -Imat -O7gpfs20/users/vh0010/mmx/src/mmx_naive.mexa64

```

(continued from previous page)

```

Recompile embedded version with '-DMATLAB_MEXCMD_RELEASE=R2017b'
/shared/software/lang/gcc/9.3.0/bin/g++ -c -DUNIX_SYSTEM -DMATLAB_DEFAULT_
-RELEASE=R2017b -DUSE_MEX_CMD -D_GNU_SOURCE -DMATLAB_MEX_FILE -I"/gpfs20/shared/
software/matlab/r2021a/extern/include" -I"/gpfs20/shared/software/matlab/r2021a/
simulink/include" -fexceptions -fPIC -fno-omit-frame-pointer -pthread -std=c++11 -O2 -
-fwrapv -DNDEBUG "/gpfs20/shared/software/matlab/r2021a/extern/version/cpp_mexapi_
version.cpp" -o /tmp/mex_59078801684538554_46067/cpp_mexapi_version.o -DMATLAB_MEXCMD_
RELEASE=R2017b
/shared/software/lang/gcc/9.3.0/bin/g++ -pthread -Wl,--no-undefined -shared -O -Wl,--
version-script,"/gpfs20/shared/software/matlab/r2021a/extern/lib/glnxa64/c_
exportsmexfileversion.map" /tmp/mex_59078801684538554_46067/mmxx.o /tmp/mex_
59078801684538554_46067/cpp_mexapi_version.o -lpthread -lstdc++ -Wl,--as-needed -
-Wl,-rpath-link,/gpfs20/shared/software/matlab/r2021a/bin/glnxa64 -L"/gpfs20/shared/
software/matlab/r2021a/bin/glnxa64" -Wl,-rpath-link,/gpfs20/shared/software/matlab/
r2021a/extern/bin/glnxa64 -L"/gpfs20/shared/software/matlab/r2021a/extern/bin/glnxa64"_
-lMatlabdataArray -lmx -lmex -lm -lmat -o /gpfs20/users/vh0010/mmxx/src/mmxx_naive.mexa64
MMEX completed successfully.
Compilation of 'mmxx_naive' succeeded.
>>

```

This is working because `libsingle_mkl_ilp64.so` on the MATLAB's `glnxa64` folder:

```
$ ls /shared/software/matlab/r2021a/extern/lib/glnxa64/libsingle_mkl_ilp64.so
/shared/software/matlab/r2021a/extern/lib/glnxa64/libsingle_mkl_ilp64.so
```

To compile without requiring special privileges, extra editing to the `build_mmxx` is needed. Basically redirecting the library paths to the location where `lib single_mkl_ilp64.so` is present. Edit `build_mmxx.m` to search for a library in the current location (-L.).

At the end of the compilation these files are created:

```

mmxx.mexa64           <-- This is equivalent to mmxx_mkl_single.mexa64
mmxx_mkl_multi.mexa64
mmxx_mkl_single.mexa64
mmxx_naive.mexa64

```

6.13.5 Teting the 3 compilations of MMX

Now lets test the library with some examples:

For using MMX, there is no need to be inside `mmxx/src` but we need to declare the folder as a path for searching MEX files:

This is one simple example:

```

$ matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

                                     < M A T L A B (R) >
Copyright 1984-2021 The MathWorks, Inc.
(glnxa64)                                         R2021a (9.10.0.1602886) 64-bit

```

(continues on next page)

(continued from previous page)

February 17, 2021

To get started, type doc.
 For product information, visit www.mathworks.com.

```
>> addpath("mmx/src")
>> A = randn(5,4,3,10,1);
>> B = randn(4,6,3,1,6);
>> C = randn(5,6,3,10,6);
>> C = mmx('mult', A, B)
```

The output will show the resulting C array. As we are using mmx() this is the most efficient compilation using MKL.

Now is time to try all the versions to check how they perform in extreme cases.

We will try a simple exercise with large matrices using the 3 versions of MMX that we compiled

mmx_naive

```
>> N = 1600
>> A = randn(N,N,3,10,1);
>> B = randn(N,N,3,1,4);
>> f_naive = @( ) mmx_naive('mult', A, B)
>> timeit(f_naive)
```

Using an interactive job and running on a SKYLAKE machine with 40 cores this is the average result:

```
22.5
```

mmx_mkl_multi

```
>> N = 1600
>> A = randn(N,N,3,10,1);
>> B = randn(N,N,3,1,4);
>> f_mkl_multi = @( ) mmx_mkl_multi('mult', A, B)
>> timeit(f_naive)
```

Average result:

0.95

mmx_mkl_single

```
>> N = 1600
>> A = randn(N,N,3,10,1);
>> B = randn(N,N,3,1,4);
>> f_mkl_single = @() mmx_mkl_single('mult', A, B)
>> timeit(f_naive)
```

Average result:

0.72

The best version of mmx is `mkl_single` which the C++ code for MMX compiled directly with MKL using the sequential libraries. For this case it performs better than the native MKL from Matlab or the naive version. See comments from `build_mmx`:

```
$ head -n 16 mmx/src/build_mmx.m
function build_mmx(verbose)
% BUILD_MMX - compiles mmx() for different platforms and provides help
%           regarding compilation.
%
% BUILD_MMX will try to compile, in this order, 3 different builds of mmx:
% mmx_mkl_single    - linked to Intel's single-threaded MKL library (usually fastest)
% mmx_mkl_multi     - linked to the multithreaded BLAS/LAPACK libraries that come
%                     with Matlab.
% mmx_naive         - does not link to anything, uses simple C-loops.
%
% The first time BUILD_MMX succeeds, it will compile again to 'mmx', so
% that the mex-file mmx should be the fastest possible build on your
% system.
%
% BUILD_MMX has been tested on Win32, Win64, OSX, Linux 64
```

6.13.6 Conclusions

In this page we show how to compile MMX, a C++ Matlab plugin (MEX file) that uses Intel MKL. The code was compiled without access to superuser privileges on a normal user folder.

MMX was compiled in 3 different flavors and a simple test example was used to demonstrate the performance of those 3 cases for large matrices.

6.14 Building Julia

Download the sources:

```
$> wget https://github.com/JuliaLang/julia/releases/download/v1.7.2/julia-1.7.2.tar.gz
```

uncompress the sources:

```
$> tar -zvxf julia-1.7.2.tar.gz
$> cd julia-1.7.2
```

edit `Make.inc`

compile Julia with:

```
$> make
```

install:

```
$> make install prefix=/shared/software/lang/julia/1.7.2
```

check the build:

```
$> make test
```

the results were:

Test Summary:	Pass	Broken	Total
Overall	38309168	352642	38661810
SUCCESS			

6.15 Tinker9

The Tinker molecular modeling software is a complete and general package for molecular mechanics and dynamics, with some special features for biopolymers. Tinker has the ability to use any of several common parameter sets, such as Amber (ff94, ff96, ff98, ff99, ff99SB), CHARMM (19, 22, 22/CMAP), Allinger MM (MM2-1991 and MM3-2000), OPLS (OPLS-UA, OPLS-AA), Merck Molecular Force Field (MMFF), Liam Dang's polarizable model, AMOEBA (2004, 2009, 2013, 2017, 2018) polarizable atomic multipole force fields, AMOEBA+ that adds charge penetration effects, and our new HIPPO (Hydrogen-like Interatomic Polarizable POTential) force field. Parameter sets for other force field models are under consideration for future releases.

Tinker9 is a complete rewrite and extension of the canonical Tinker software, currently Tinker8. Tinker9 is implemented as C++ code with OpenACC directives and CUDA kernels providing excellent performance on GPUs. At present, Tinker9 builds against the object library from Tinker8, and provides GPU versions of the Tinker ANALYZE, BAR, DYNAMIC, MINIMIZE and TESTGRAD programs. Existing Tinker file formats and force field parameter files are fully compatible with Tinker9, and nearly all Tinker8 keywords function identically in Tinker9. Over time we plan to port much or all of the remaining portions of Fortran Tinker8 to the C++ Tinker9 code base.

These instructions describe how tinker9 was compiled on Thorny Flat with GPU support for the particular GPU cards present on this cluster.

6.15.1 Clone the repository

Tinker9 sources are on Github and uses submodules. This command will download the latest version and the submodules:

```
$> git clone --recurse-submodules https://github.com/TinkerTools/tinker9.git
```

6.15.2 Create a build folder

It is a best practice to use a separate build folder to compile the code. The sources remain untouched and it is easy to just remove the content of the folder and recompile:

```
$> cd tinker9  
$> mkdir build  
$> cd build
```

6.15.3 Modules for compilation

Tinker9 uses cmake for building the software. The version of cmake on RHEL 7.x is usually too old so it is a good idea to load a more recent version. RHEL 7.x comes with cmake version 2.8.12.2 and the CMakeLists request the minimal version being 3.12.

Tinker9 is implemented as C++ and takes advantage of GPUs via OpenACC directives and CUDA programming. The natural compiler for a code that uses OpenACC is NVIDIA HPC suite as OpenACC is better implemented. For the CUDA side the latest version of the CUDA Toolkit at the time of writing this (April 2022) is CUDA 11.5. These are the modules loaded for compiling tinker9:

```
$> module load dev/cmake/3.21.1 dev/git/2.29.1 lang/nvidia/nvhpc/21.11 parallel/cuda/11.5
```

6.15.4 Configuring tinker9

Configuring a code with cmake will prepare the sources, and create the Makefiles for building the binaries using the compilers and libraries declared with the arguments presented to cmake. The cmake line used was this:

```
$> cmake -DCUDA_DIR=${CUDA_HOME} -DCOMPILER_CAPABILITY=61,75,80 \  
-DCMAKE_Fortran_COMPILER=gfortran -DCMAKE_BUILD_TYPE=release \  
-DCMAKE_INSTALL_PREFIX=/shared/software/atomistic/tinker9 \  
-DCMAKE_CUDA_COMPILER=`which nvcc` -Wno-dev ..
```

The CUDA compute capabilities can be found on [CUDA Website](#) For the particular case of Thorny Flat, we have 3 kinds of GPUs: P6000, RTX6000, and A100. The compute capabilities for these cards are:

GPU Model	Compute Capability
Quadro P6000	61
Quadro RTX 6000	75
NVIDIA A100	80

After configuring the code execute make with a reasonable number of compilation threads. On the head node of Thorny Flat we have 24 cores, using 12 compilation threads is reasonable and the code will compile relatively fast:

```
$> make -j12
```

6.15.5 Running the tests

To run the tests lauch and interactive job on a queue allowing GPUs:

```
$> qsub -I -l nodes=1:ppn=16:gpus=2 -q be_gpu
```

Go to the build folder and check the GPU present on the node:

```
$> cd /shared/src/tinker9/build
$> nvidia-smi
Fri Apr  1 12:35:39 2022
+-----+
| NVIDIA-SMI 495.29.05      Driver Version: 495.29.05      CUDA Version: 11.5      |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC  | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M.  |
| |          |          |           |           |          MIG M. |
+-----+
|  0  Quadro RTX 6000     Off  | 00000000:C1:00.0 Off  |                0 | | | |
| N/A   25C     P8    14W / 250W |    0MiB / 22698MiB |      0%  E. Process  |
| |          |          |           |           |          N/A |
+-----+
|  1  Quadro RTX 6000     Off  | 00000000:C2:00.0 Off  |                0 | | | |
| N/A   25C     P8    15W / 250W |    0MiB / 22698MiB |      0%  E. Process  |
| |          |          |           |           |          N/A |
+-----+
+-----+
| Processes:
| GPU  GI  CI      PID  Type  Process name          GPU Memory  |
|       ID  ID          |          Usage
+-----+
|  No running processes found
+-----+
```

Load the modules:

```
$> module load dev/cmake/3.21.1 dev/git/2.29.1 lang/nvidia/nvhpc/21.11 parallel/cuda/11.5
```

Execute the test suite:

```
$> make test
[ 69%] Built target tinkerObjF
[ 69%] Built target tinkerObjCpp
[ 69%] Built target tinkerFToCpp
[ 83%] Built target tinker9_cpp
[ 88%] Built target __t9_all_tests_o
[ 89%] Built target tinker9_f
[ 89%] Built target tinker9_version
[ 97%] Built target tinker9_acc
```

(continues on next page)

(continued from previous page)

```
[100%] Built target tinker9_cu
[100%] Built target all.tests
Filters: info
Primary GPU package : CUDA

GPU Device : Setting Device ID to 0 from GPU utilization
```

Program Information

Version:	1.0.0 GIT 1e34a417
Synchronized with Tinker commit:	5aa9948d
C++ compiler:	nvc++ 21.11.0
Size of real (bytes):	4
Size of mixed (bytes):	8
Using deterministic force:	true
Debug mode:	off
Platform:	CUDA and OpenACC
Primary GPU package:	CUDA
Latest CUDA supported by driver:	11.5
CUDA runtime version:	11.5
Thrust version:	1.13.1 patch 0
CUDA compiler:	nvcc 11.5.119
OpenACC compiler:	nvc++ 21.11.0
GPU detected:	2
GPU 0:	
PCI:	0000:C1:00.0
Name:	Quadro RTX 6000
Maximum compute capability:	7.5
Single double perf. ratio:	32
Compute mode:	exclusive process
Error-correcting code (ECC):	on
Clock rate (kHz):	1620000
Number of Multiprocessors:	72
Number of CUDA cores:	9216
Used/Total GPU memory:	0.79 % / 22.17 GB
GPU 1:	
PCI:	0000:C2:00.0
Name:	Quadro RTX 6000
Maximum compute capability:	7.5
Single double perf. ratio:	32
Compute mode:	exclusive process
Error-correcting code (ECC):	on
Clock rate (kHz):	1620000
Number of Multiprocessors:	72
Number of CUDA cores:	9216
Used/Total GPU memory:	0.79 % / 22.17 GB
<hr/>	
test cases:	1 1 passed
assertions:	- none -
Filters:	[ff],[util]

(continues on next page)

(continued from previous page)

```
....#####
....#####
..###                                     #####
.###          Tinker9 -- Software Tools for Molecular Design      #####
.##                                     ###
.##          Version 1.0.0-rc January 2021                      ##
.##                                     ##
.##          Copyright (c) Zhi Wang & the Ponder Lab             ##
.###          All Rights Reserved                                ##
..###                                     ##
....#####
....#####
....#####

Compiled at: 12:18:38 Apr 1 2022
Commit Date: Fri Apr 1 04:23:24 2022 -0500
Commit:       1e34a417

Primary GPU package : CUDA

GPU Device : Setting Device ID to 0 from GPU utilization

...
...
...

=====
All tests passed (65640 assertions in 58 test cases)

[100%] Built target test
```

6.16 Cuda

Installing new CUDA drivers on GPU compute nodes

```
$> systemctl stop nvidia-dcgm
$> rmmod nvidia_modeset nvidia_drm nvidia_uvm nvidia
$> sh /shared/src/cuda_11.7.0_515.43.04_linux.run --silent --driver
$> sh /shared/src/cuda_11.7.0_515.43.04_linux.run --silent --toolkit
```


DOMAIN SPECIFIC DETAILS

7.1 Engineering: ANSYS Products

7.1.1 Log in to Spruce

Enter on Spruce with X11 forwarding, at least on Linux it is with

```
ssh -X <USERNAME>@spruce.hpc.wvu.edu
```

On Windows you can MobaXTerm and the program will give you X11 forwarding too

You can test the X11 forwarding with:

```
xeyes
```

A small window with two eyes will pop up on your screen, execute CTRL-C to exit. The test will just prove that a window can be created on your client machine.

7.1.2 Create an interactive job

Ask for an interactive job with X11 Forwarding enabled, I am using “debug” queue, change accordingly

```
qsub -X -I -q debug
```

Wait around 2 minutes to get a compute node allocated for you. Different queues could have different response times before a compute node can be allocated to you. In general use “standby” on Spruce for Interactive jobs to run during a 4-hour timeframe.

7.1.3 Load the module for ANSYS Fluids or Structures

There are two modules corresponding to equal number of ANSYS Suites of packages

```
module load ansys/fluids_18.1
```

and

```
module load ansys/structures_18.1
```

You can load one or another or both, they use the same MPI Runtime Libraries so they are compatible to each other.

7.1.4 Executing the ANSYS package of your choice

Most commands will be directly accessible to you, you do not need to enter the full path

```
fluent
```

The fluent window opens and on the lower right corner you will see:

```
/shared/software/ansys/fluids_181/ansys_inc/v181/fluent/fluent18.1.0/bin/fluent -r18.1.0
→3d -alnAMD64 -path/shared/software/ansys/fluids_181/ansys_inc/v181/fluent -ssh -cx
→sllc0001.hpc.wvu.edu:41717:33691
Starting /shared/software/ansys/fluids_181/ansys_inc/v181/fluent/fluent18.1.0/lnAMD64/3d/
→fluent.18.1.0 -cx sllc0001.hpc.wvu.edu:41717:33691

Welcome to ANSYS Fluent Release 18.1

Copyright 2017 SAS IP, Inc. All Rights Reserved.
Unauthorized use, distribution or duplication is prohibited.
This product is subject to U.S. laws governing export and re-export.
For full Legal Notice, see documentation.

Build Time: Apr 11 2017 14:22:58 EDT Build Id: 10162
```

This is an academic version of ANSYS FLUENT. Usage of this product license is limited to the terms and conditions specified in your ANSYS license form, additional terms section.

```
Cleanup script file is /gpfs/home/gufranco/cleanup-fluent-sllc0001.hpc.wvu.edu-48370.sh
```

You can try also ANSYS/Forte

```
forte.sh
```

A window with 3 big buttons appear, click Simulate

The Full Window for Forte appears.

In the case of ANSYS Structures 18.1 all that you have to do after loading the module is to open the Workbench:

```
runwb2
```

7.2 Engineering: ANSYS/Forte

7.2.1 Overview

ANSYS Forte is the only CFD simulation package for internal combustion engines that incorporates proven ANSYS Chemkin-Pro solver technology – the gold standard for modeling and simulating gas phase and surface chemistry. Forte includes state-of-the-art automatic mesh generation (AMG), including solution adaptive mesh Refinement (SAM) and geometry-based adaptive mesh refinement (AMR). While legacy engine-combustion CFD simulations utilize chemistry solvers that are too slow to handle the chemistry details required for accurate predictions of ignition and emissions, Forte enables the use of multicomponent fuel models to combine with comprehensive spray dynamics – without sacrificing simulation time-to-solution.

7.2.2 Features

- Automatic mesh generation that eliminates weeks of effort typically spent on manual mesh preparation
- Embedded Chemkin-Pro solver technology that provides the computational speed required for predictive engine simulations
- True multicomponent fuel-vaporization models that enable a self-consistent representation of the physical spray and the kinetics for accurate prediction of fuel effects
- Advanced spray models that dramatically reduce grid and time-step dependency when compared to existing approaches
- The ability to track soot particle nucleation, growth, agglomeration and oxidation without a compute-time penalty to predict particle size and number

7.2.3 Tutorial

This tutorial will describe the steps needed to complete the execution of the first examples from the tutorial documentation.

Connect to Mountaineer

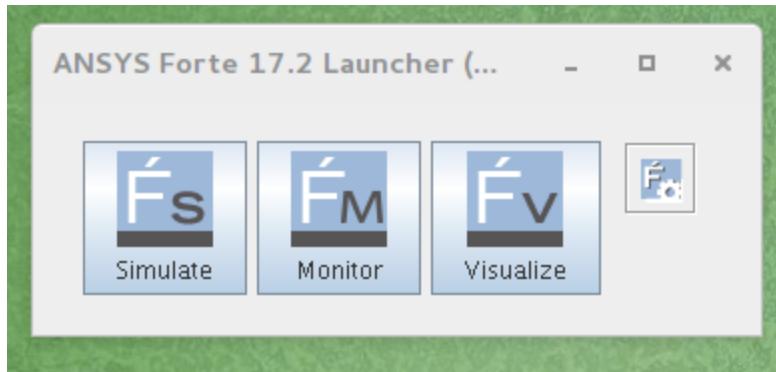
```
ssh -X username@mountaineer.hpc.wvu.edu
```

The `~-X` is necessary to get access to the graphical interface needed to prepare the simulation.

Load the Module

```
module load ansys/forte/17.2
```

Forte initial GUI



Execute the ANSYS/Forte GUI

First, we need to create a directory for the simulation we want to perform. Lets create for example a directory called 'forte' and inside create a subdirectory for our first simulation. The first simulation will be 'SectorMesh'

```
mkdir $HOME/forte
mkdir $HOME/forte/SectorMesh
```

The Official Tutorials for Forte are in '/software/ansys_central/Forte_Tutorials', we will copy Sandia_Engine_LTC_EarlyInj.ftsim to the directory we just create.

```
cp /software/ansys_central/Forte_Tutorials/Quick-start_SectorMesh/Sandia_Engine_LTC_
↳EarlyInj.ftsim $HOME/forte/SectorMesh
```

We will use this input for this tutorial, now, we can launch the GUI using the command:

```
forte.sh &
```

The '&' will free your terminal so you can enter more commands there. Otherwise you will need another terminal when we actually launch the simulation. Once you execute 'forte.sh' you should get a window with 3 buttons: Simulate, Monitor and Visualize.

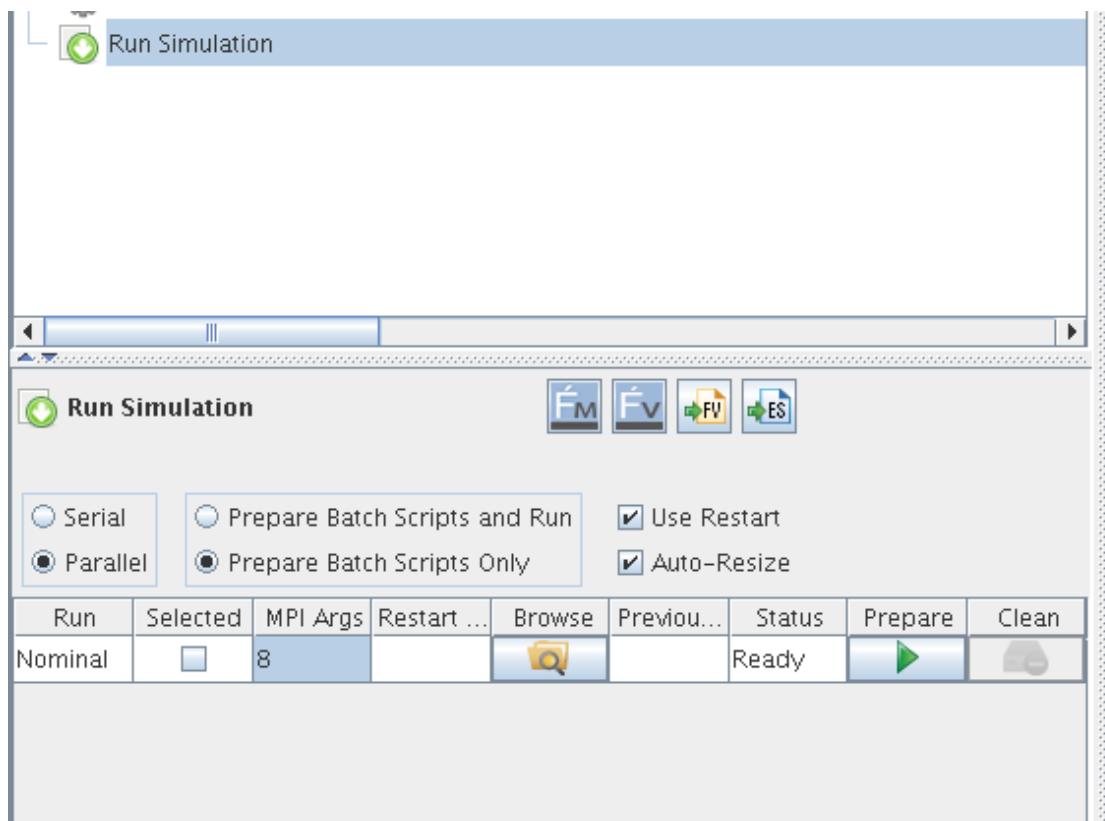
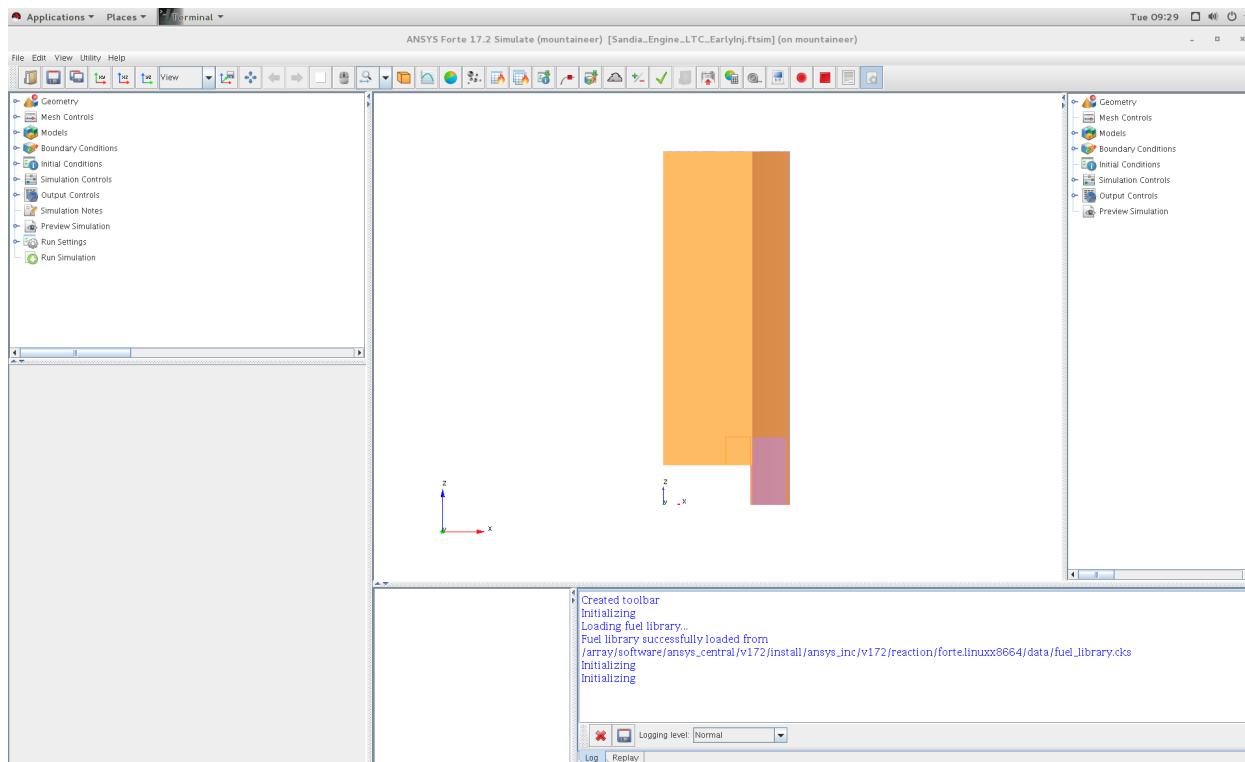
Simulate Window

Click on Simulate button and open the file `Sandia_Engine_LTC_EarlyInj.ftsim` You should get the 'Simulate' window. Click on Menu 'File - Open...' and search for the input file 'Sandia_Engine_LTC_EarlyInj.ftsim', in your directory 'forte/SectorMesh'. The main window will display the mesh, you get on the log panel messages such

```
Loading fuel library...
Fuel library successfully loaded from
/array/software/ansys_central/v172/install/ansys_inc/v172/reaction/forte.linuxx8664/
↳data/fuel_library.cks
Initializing
Initializing
```

Now we are ready to create the scripts and run the simulation on Mountaineer

Simulate Window



Prepare simulation files

Click on “Run Simulation” in the tree panel. The options for running the simulation appear in a panel down the tree panel. There are several options that we need to change. Select Parallel instead of Serial. Change to ‘Prepare Batch Scripts Only’. There is also a field under ‘MPI Args’, change that to the number of cores that you want use. 8 could be a reasonable number for this small simulation. Click the Green triangle under ‘Prepare’.

Now, we need to go back to the terminal and see the effects of preparing the simulation runs. A new directory is created called ‘Sandia_Engine_LTC_EarlyInj.analysis’. Inside that directory there are one file called ‘Sandia_Engine_LTC_EarlyInj.ftsim’ and a directory called ‘Nominal’ The contents of Nominal are:

```
chem.inp
chem.out
fuel_lib_chem.inp
fuel_lib_chem.out
fuel_lib_gas.asc
fuel_lib_xml.out
gas.asc
PREPARED
run_env.bat
run_env.sh
run_mpi.bat
run_mpi.sh
Sandia_Engine_LTC_EarlyInj.ftsim
therm.dat
xml.out
```

Now, we need to prepare the submission script for Mountaineer Go to ‘Nominal’ directory and create the script file

```
cd $HOME/forte/SectorMesh/Sandia_Engine_LTC_EarlyInj.analysis/Nominal
cat <<EOF >runjob.pbs
#!/bin/sh

#PBS -N FORTE
#PBS -l nodes=1:ppn=8,walltime=01:00:00
#PBS -m ae
##PBS -M username@mail.wvu.edu
#PBS -q hour

module load ansys/forte/17.2

cd $PBS_O_WORKDIR
sh run_mpi.sh
EOF
```

You should remove one # from ##PBS -M username@mail.wvu.edu and change the email address in order to receive notifications when job finish. We are requesting 1 hour (walltime=01:00:00), using one node (nodes=1) and creating 8 MPI processes (ppn=8). Those values should be adapted to the needs of your job. If you ask for more than one hour you must change also the queue where you are submitting the jobs (#PBS -q hour), consider for example queues ‘day’ and ‘week’ if your jobs need more time to complete.

Submit the job with

```
qsub runjob.pbs
```

The job is submitted to the queue, it will start running when enough resources are free to start executing your job. You can always monitor the status of your submission with

```
qstat -u username
```

Once your job simulation start execution, you can monitor the progress by looking at the MONITOR file, one way of doing that is using tail command.

```
tail -f MONITOR
```

You should see something like

7852	116.38	3.908E-02	5.000E-06	Maximum reached	953.80	330.53	0.4565	✉
↪ 9.79E+00	15729	190						
7853	116.42	3.909E-02	5.000E-06	Maximum reached	953.70	330.51	0.4563	✉
↪ 9.79E+00	15729	191						
7854	116.45	3.909E-02	5.000E-06	Maximum reached	953.60	330.49	0.4556	✉
↪ 9.78E+00	15729	194						
7855	116.49	3.910E-02	5.000E-06	Maximum reached	953.50	330.48	0.4557	✉
↪ 9.78E+00	15729	191						
7856	116.52	3.910E-02	5.000E-06	Maximum reached	953.41	330.46	0.4560	✉
↪ 9.78E+00	15729	190						
7857	116.56	3.911E-02	5.000E-06	Maximum reached	953.31	330.44	0.4557	✉
↪ 9.78E+00	15729	190						
7858	116.60	3.911E-02	5.000E-06	Maximum reached	953.21	330.42	0.4549	✉
↪ 9.77E+00	15729	188						
7859	116.63	3.912E-02	5.000E-06	Maximum reached	953.12	330.41	0.4552	✉
↪ 9.77E+00	15729	188						
Cycle# CA[deg] Time[s] Step[s] Step constraint					MaxT[K]	MinT[K]	MaxP[MPa]	✉
↪ MaxV[m/s]								
↪ #Cells #Clusters								
7860	116.67	3.912E-02	5.000E-06	Maximum reached	953.02	330.38	0.4550	✉
↪ 9.77E+00	15729	188						
7861	116.70	3.913E-02	5.000E-06	Maximum reached	952.92	330.37	0.4551	✉
↪ 9.76E+00	15729	188						
7862	116.74	3.913E-02	5.000E-06	Maximum reached	952.82	330.35	0.4542	✉
↪ 9.76E+00	15729	185						
7863	116.78	3.914E-02	5.000E-06	Maximum reached	952.72	330.33	0.4542	✉
↪ 9.76E+00	15729	187						
7864	116.81	3.914E-02	5.000E-06	Maximum reached	952.63	330.32	0.4544	✉
↪ 9.76E+00	15729	188						
7865	116.85	3.915E-02	5.000E-06	Maximum reached	952.53	330.29	0.4541	✉
↪ 9.75E+00	15729	186						
7866	116.88	3.915E-02	5.000E-06	Maximum reached	952.43	330.28	0.4535	✉
↪ 9.75E+00	15729	186						
7867	116.92	3.916E-02	5.000E-06	Maximum reached	952.33	330.26	0.4534	✉
↪ 9.75E+00	15729	188						
7868	116.96	3.916E-02	5.000E-06	Maximum reached	952.23	330.25	0.4534	✉
↪ 9.74E+00	15729	186						
7869	116.99	3.917E-02	5.000E-06	Maximum reached	952.13	330.23	0.4538	✉
↪ 9.74E+00	15729	187						
Cycle# CA[deg] Time[s] Step[s] Step constraint					MaxT[K]	MinT[K]	MaxP[MPa]	✉
↪ MaxV[m/s]								
↪ #Cells #Clusters								

(continues on next page)

(continued from previous page)

7870	117.03	3.917E-02	5.000E-06	Maximum reached	952.03	330.21	0.4535	✉
↪ 9.74E+00	15729	189						
7871	117.06	3.918E-02	5.000E-06	Maximum reached	951.93	330.19	0.4527	✉
↪ 9.74E+00	15729	192						
7872	117.10	3.918E-02	5.000E-06	Maximum reached	951.83	330.18	0.4528	✉
↪ 9.73E+00	15729	192						
7873	117.14	3.919E-02	5.000E-06	Maximum reached	951.73	330.16	0.4530	✉
↪ 9.73E+00	15729	192						

The screen will update when the execution advances. When simulation finishes you should get a summary of execution like

Engine Summary Data:		
Cylinder Index	1	
Power	9.84048	[kW]
IMEP	0.42126	[MPa]
Fuel Mass	5.35000E-002	[g/cyc]
Fuel Lower Heating Value	44.52000	[kJ/g]
Gross ISFC	195.72213	[g/kW-h]
ISFC(IVC->EVO)	223.19199	[g/kW-h]
Total Chemical Heat Release	2376.63829	[J]
Total Heat Release From P-V	1297.21755	[J]
Combustion Efficiency	0.99782	[]
Thermal Efficiency	0.41315	[]
Max Pressure	8.62842	[MPa]
Max Temperature	1262.21947	[K]
Max Pressure Rise Rate	0.84582	[MPa/deg]
CA @ 2% Heat Release	-14.97600	[deg ATDC]
CA @ 10% Heat Release	-8.99000	[deg ATDC]
CA @ 30% Heat Release	-5.98561	[deg ATDC]
CA @ 50% Heat Release	-5.98561	[deg ATDC]
CA @ 90% Heat Release	0.02813	[deg ATDC]
10%-90% Heat Release Duration	9.01813	[deg]
Soot @ EVO	3.66015E-004	[g/kg-f]
	7.16372E-005	[g/kW-h]
	8.72851E-006	[ppm]
EINOx @ EVO	3.39138E+000	[g/kg-f]
	6.63767E-001	[g/kW-h]
	2.09039E+001	[ppm]
CO @ EVO	2.17074E+001	[g/kg-f]
	4.24862E+000	[g/kW-h]
	2.19759E+002	[ppm]
UHC @ EVO	1.30878E+001	[g/kg-f]
	2.56157E+000	[g/kW-h]
C1-UHC @ EVO	2.59096E+002	[ppm]
VOC @ EVO	1.35866E+001	[g/kg-f]
	2.65920E+000	[g/kW-h]
C1-VOC @ EVO	2.63832E+002	[ppm]

(continues on next page)

(continued from previous page)

```
=====
Normal termination: Cycle# 8092, CrankAngle[deg] 125.00, Time[s] 4.0278E-02
CGNS solution file closed successfully
=====
```

```
=====
Summary of computational times:
=====
```

Total wall-clock time	1438.9 seconds (23 min, 58.9 sec)
Wall-clock time in chemistry	230.8 seconds (3 min, 50.8 sec)
Total CPU time	11511.0 seconds (3 h, 11 min, 51.0 sec)
CPU time in chemistry	1497.8 seconds (24 min, 57.8 sec)

```
=====
=====
```

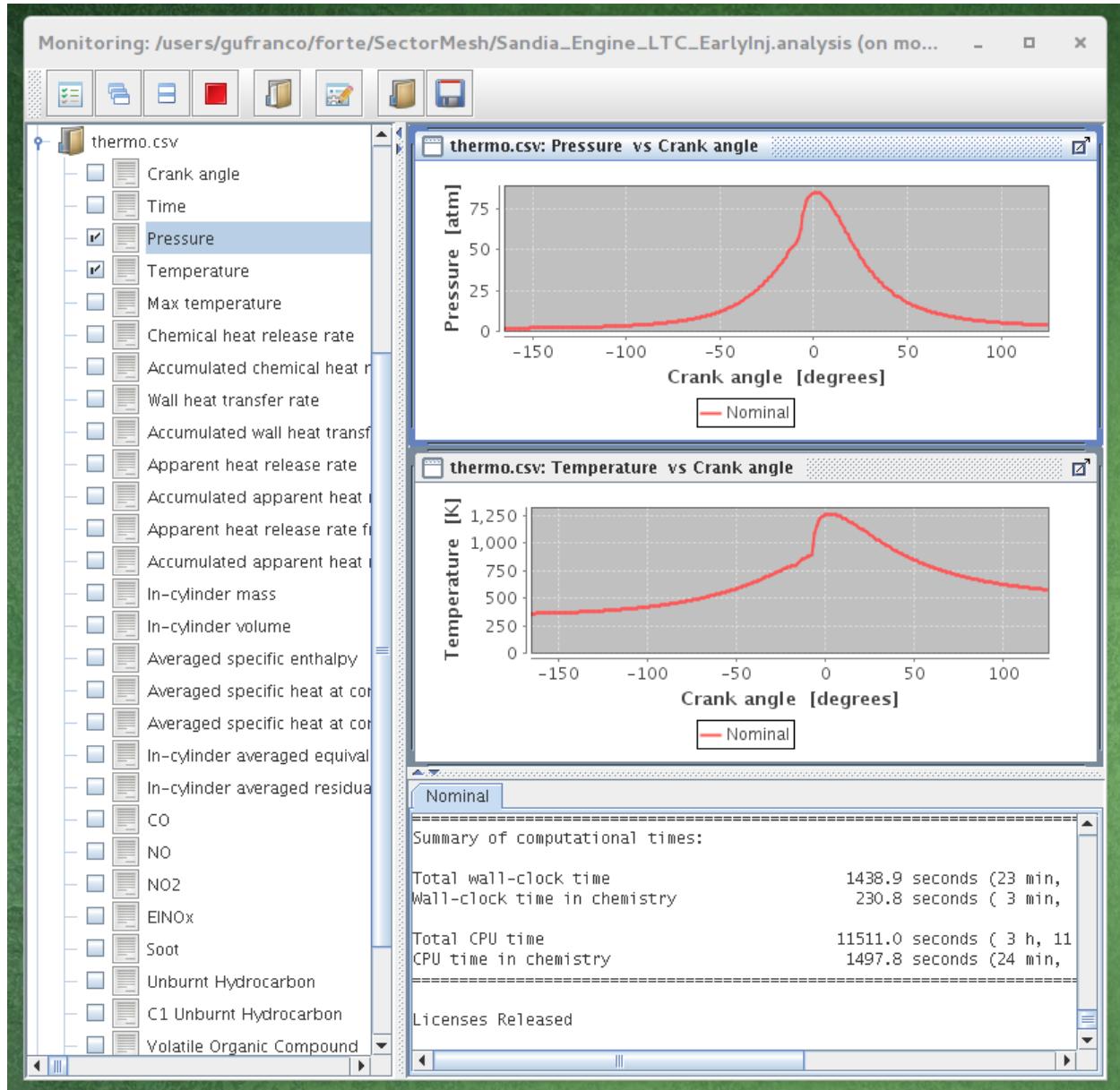
Licenses Released

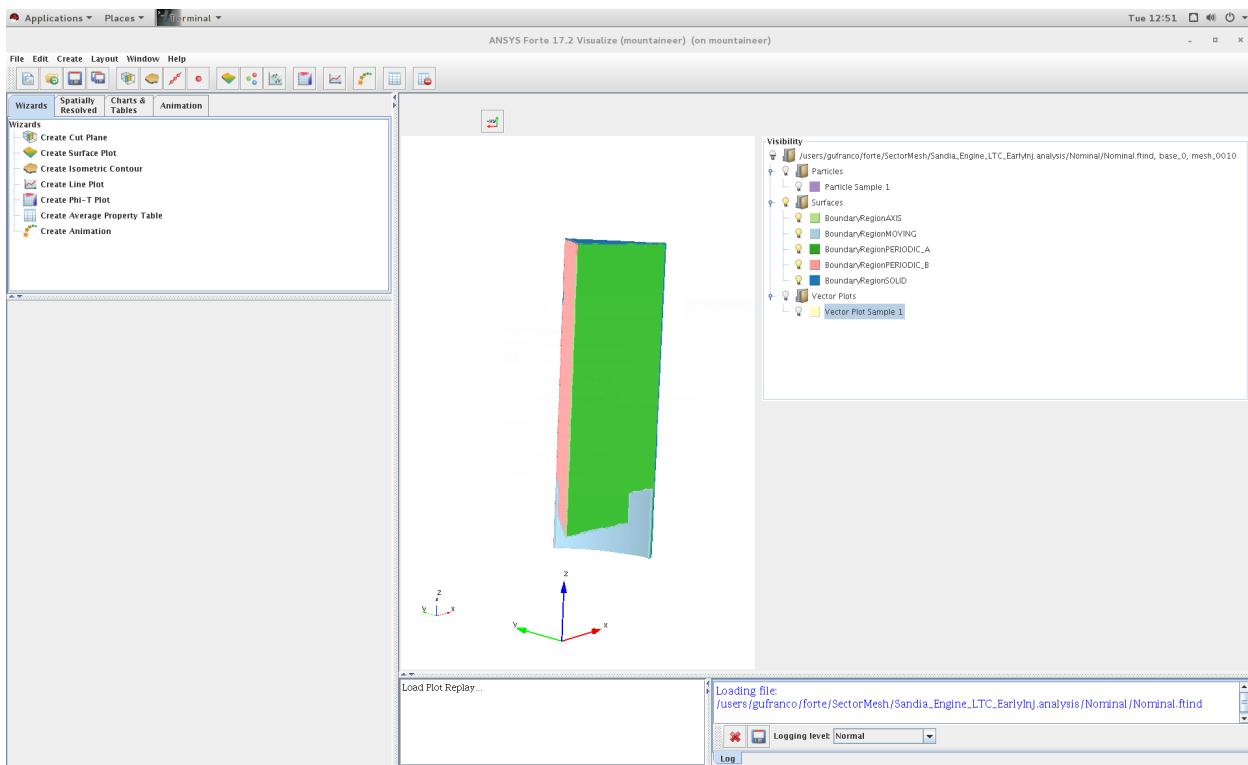
Many files are now on you Nominal directory.

chem_0	chemsolver.csv	fuel_library_diag_0	gas.asc
chem_0.out	COMPLETE	fuel_library_diag_1	massfraction.csv
chem_1	dynamic.csv	fuel_library_diag_2	memory_diagnostics.csv
chem_1.out	flow_diagnostics.csv	fuel_library_diag_3	molefraction.csv
chem_2	FORTE.e107840	fuel_library_diag_4	MONITOR
chem_2.out	FORTE.log	fuel_library_diag_5	nc14h30_fueltable.csv
chem_3	FORTE.o107840	fuel_library_diag_6	Nominal_CA_p_125.00_8092.ftrst
chem_3.out	fuelchem_0	fuel_library_diag_7	Nominal.ftavg
chem_4	fuelchem_0.out	fuel_lib_xml.out	Nominal.ftind
chem_4.out	fuelchem_1	fueltable.out	Nominal.ftres
chem_5	fuelchem_1.out	fueltran_0	run_env.bat
chem_5.out	fuelchem_2	fueltran_0.out	run_env.sh
chem_6	fuelchem_2.out	fueltran_1	runjob.pbs
chem_6.out	fuelchem_3	fueltran_1.out	runjob.pbs~
chem_7	fuelchem_3.out	fueltran_2	run_mpi.bat
chem_7.out	fuelchem_4	fueltran_2.out	run_mpi.sh
chemdiag_0	fuelchem_4.out	fueltran_3	Sandia_Engine_LTC_EarlyInj.ftsim
chemdiag_1	fuelchem_5	fueltran_3.out	speciesmass.csv
chemdiag_2	fuelchem_5.out	fueltran_4	spray.csv
chemdiag_3	fuelchem_6	fueltran_4.out	therm.dat
chemdiag_4	fuelchem_6.out	fueltran_5	thermo.csv
chemdiag_5	fuelchem_7	fueltran_5.out	wall_heat_transfer.csv
chemdiag_6	fuelchem_7.out	fueltran_6	whole_domain_parameters.csv
chemdiag_7	fuel_lib_chem.inp	fueltran_6.out	xml.out
chem.inp	fuel_lib_chem.out	fueltran_7	
chem.out	fuel_lib_gas.asc	fueltran_7.out	

The best way of continue from here is to copy those files back to your workstation for post-processing and analysis.

You can however continue on Mountaineer and start analysing the results of the simulation. On the initial window with three buttons, click on Monitor and open the project file. The image on the left, you can get see basic plots of temperature and pressure as a function of crank angle. By clicking on the Visualize button you get a big window where you can see visualizations of the fluids inside the crank.





asdsadsd

7.3 Engineering: OpenFOAM 10

On Thorny Flat OpenFOAM 10 is available via Singularity containers. There are two images on these locations:

```
/shared/containers/openfoam10-graphical-apps.sif
/shared/containers/openfoam10-paraview56.sif
```

7.3.1 Run OpenFOAM with an interactive job

We start by creating an interactive job, for the simple examples such as the tutorials, requesting 4 cores is more than enough:

```
$> qsub -I -l nodes=1:ppn=4
```

After a few minutes you get a message indicating that a compute node and corresponding cores have been assigned, you get connected to the compute node:

```
qsub: waiting for job 1660023.trcis002.hpc.wvu.edu to start
qsub: job 1660023.trcis002.hpc.wvu.edu ready

18:28:50-username@tcgcx300:~$
```

Lets work on the `$SCRATCH` folder and create a subfolder there:

```
$> cd $SCRATCH  
$> mkdir openfoam_tutorial  
$> cd openfoam_tutorial
```

Load the module for singularity and get a shell inside the container:

```
$> module load singularity  
$> singularity shell /shared/containers/openfoam10-paraview56.sif
```

Lets copy one of the examples in the tutorials:

```
Singularity> cp -r /opt/openfoam10/tutorials/incompressible/icoFoam/cavity .  
Singularity> cd cavity/  
Singularity> ls  
Allclean Allrun cavity cavityClipped cavityGrade
```

The tutorials include some scripts with easier access to the utilities in OpenFOAM:

```
Singularity> . $WM_PROJECT_DIR/bin/tools/RunFunctions
```

The mesh is prepared with the command:

```
Singularity> runApplication blockMesh  
Running blockMesh on /scratch/gufranco/openfoam_tutorial/cavity/cavity
```

And running the particular application from OpenFOAM:

```
Singularity> runApplication icoFoam  
Running icoFoam on /scratch/gufranco/openfoam_tutorial/cavity/cavity
```

This is a very simple example that takes less than a second to run. The result of the mesh preparation and simulation is a set of files in the current folder and log files:

```
Singularity> ls  
0 0.1 0.2 0.3 0.4 0.5 constant log.blockMesh log.icoFoam system
```

For example the ouput of the log for icoFoam is posted there (first and last lines):

```
===== |  
\\ / F ield | OpenFOAM: The Open Source CFD Toolbox  
\\ / O peration | Website: https://openfoam.org  
\\ / A nd | Version: 10  
\\/ M anipulation |  
\\-----*/  
Build : 10-0347120b3b9d  
Exec : icoFoam  
Date : Sep 15 2022  
Time : 21:19:29  
Host : "tcgcx300.hpc.wvu.edu"  
PID : 185550  
I/O : uncollated  
Case : /scratch/gufranco/openfoam_tutorial/cavity/cavity  
nProcs : 1  
fileModificationChecking : Monitoring run-time modified files using timeStampMaster  
(fileModificationSkew 10)
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```

time step continuity errors : sum local = 9.66354e-09, global = -1.28048e-19,
cumulative = 2.56141e-19
ExecutionTime = 0.064562 s ClockTime = 1 s

End

```

7.3.2 Using ParaView with OpenFoam

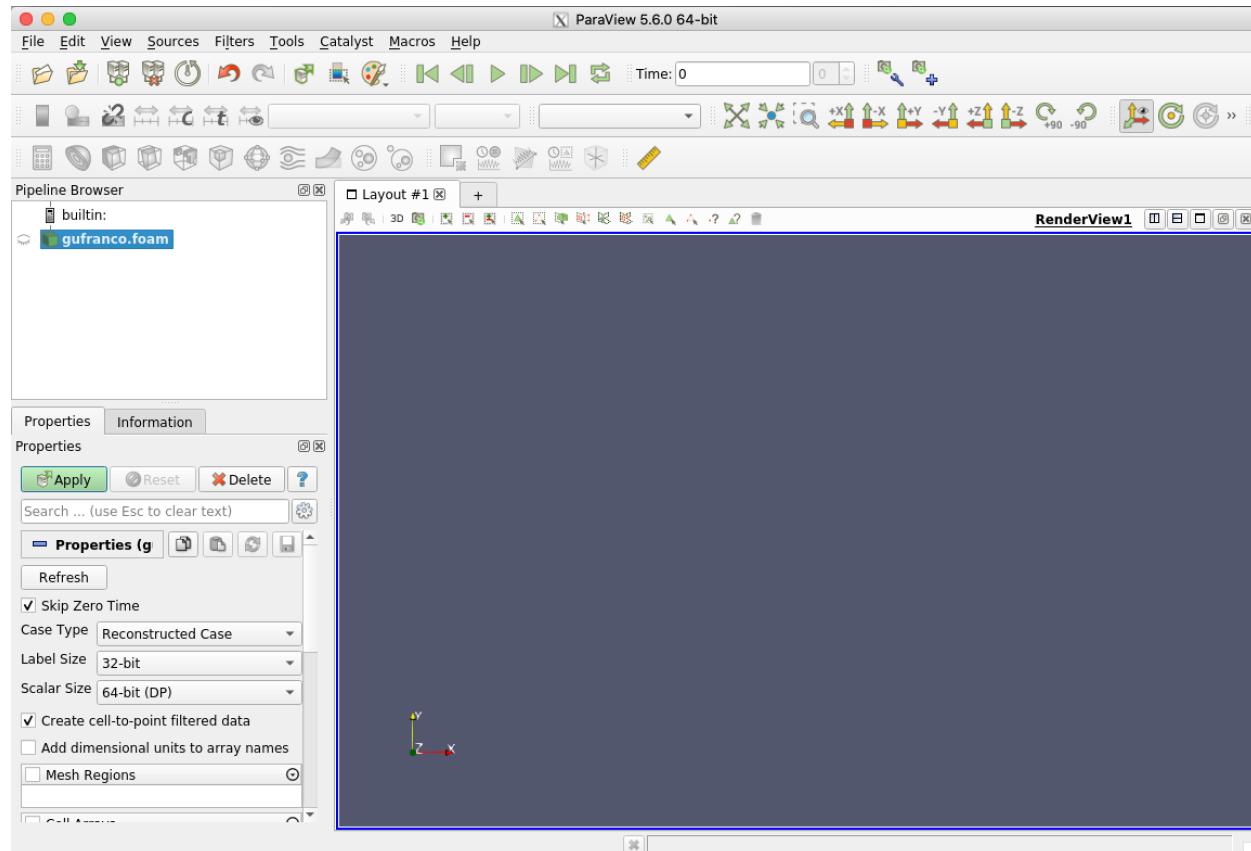
Visualization of openFoam output can be executed from the head node. To use paraFoam, make sure you have logged in with X11 forwarding enabled.

Connect to Thorney using X11 Forwarding:

```

$> ssh -X <username>@ssh.wvu.edu
$> ssh -X tf.hpc.wvu.edu
$> module load singularity
$> singularity shell /shared/containers/openfoam10-paraview56.sif
Singularity> paraFoam -builtin

```



7.4 Bioinformatics: Using Bowtie2

This tutorial will show how to use Job Arrays to manage the alignment of a big pair of sequencing data.

For the purpose of the tutorial, consider that you receive a pair of files like this:

```
Estrogen_24_1.clean.fq  
Estrogen_24_2.clean.fq
```

And you have a reference like:

```
Oncorhynchus_mykiss_chr.fa
```

This tutorial will show you how to do the alignments concurrently by splitting the fq files and use BSseeker and bowtie2 for the alignment. When the job is done we use samtools to merge the results in a single BAM file.

7.4.1 1. Transfer the files

The files referred above are really big, the first step is to send those files to the cluster. The files are too big for store them on your home account so you have to use the scratch space. There are two ways of sending the files to the cluster. Using command line tools like rsync or using the Globus connector to send those files from your computer to the cluster.

If you are using Linux or Mac on your side, the command to send those files will be,

```
rsync -av Estrogen_24_1.clean.fq Estrogen_24_2.clean.fq Oncorhynchus_mykiss_chr.  
fa [username]@spruce.hpc.wvu.edu:/scratch/[username]
```

Globus is another alternative, follow the instructions: [Transferring_Files]

7.4.2 2. Preparing folders

It is always good practice to keep some structure on the files that you will use on your research. We suggest creating a few directories, for the references, for the big sequence fq files, output and temporary files.

```
cd /scratch/[username]  
mkdir REFS OUTPUT TMP WGBS
```

Now we need to move the big fq files to WGBS and the reference to REFS

```
mv Estrogen_24_1.clean.fq Estrogen_24_2.clean.fq WGBS  
mv Oncorhynchus_mykiss_chr.fa REFS
```

7.4.3 3. Splitting the files

Our first couple of jobs consists on splitting the two big fq files. We can get an idea about how big they are by counting the number of lines

```
$ wc -l WGBS/Estrogen_24*.fq
`` 670407612 WGBS/Estrogen_24_1.clean.fq``
`` 670407612 WGBS/Estrogen_24_2.clean.fq``
`` 1340815224 total``
```

They have the same number of lines, more than 670 million lines. It is important to realize that you cannot simple cut those files arbitrarily. Each 4 lines makes a block of information that cannot be splitted

```
@ST-E00144:275:HW5VJCCXX:7:1101:2564:1327 1:N:0:NTGAGCCA
```

```
NGAGAATATTTGGTTATTTTTAGTTTGATTGGAAGATTATTAGATAGAGAATATTGTTGGTTATTTTTAGTTTGATTGGAGGATGTAG
+
#AAAAA7-7A
```

1. !/bin/bash
2. PBS -N SPLIT
3. PBS -l nodes=1:ppn=1
4. PBS -l walltime=4:00:00
5. PBS -m ae
6. PBS -q standby

```
cd $PBS_O_WORKDIR
```

```
mkdir -p WGBS/split echo $file split -d -a 3 -l 2400000 WGBS/${file} WGBS/split/${file}_
```

The submission script will create a job called “SPLIT” using one node (nodes=1) and one process per node (ppn=1). We set a walltime in 4 hours, the limit for the standby queue. We suppose to submit this job from the scratch directory. So the first step is to move into that directory using cd \$PBS_O_WORKDIR, we create a folder for the split files inside WGBS. The split command:

```
split -d -a 3 -l 2400000 WGBS/${file} WGBS/split/${file}_
```

The split command will create split files with 3 digits (-a 3), starting from 000. It will split every 2.4 million lines, the file indicated under the file variable that will will use next. All those split files will be stored at WGBS/split using the same name as the original but with a suffix being underscored “_” and the 3 digits mentioned before.

We submit the jobs like this:

```
qsub runjob_split.pbs -v file=Estrogen_24_1.clean.fq
qsub runjob_split.pbs -v file=Estrogen_24_2.clean.fq
```

Those two jobs takes around 30 minutes in one of our compute nodes. Once that splitting is complete you can see many files populating the WGBS/split folder. Those are the files we will use in our next step

7.4.4 4. Using BSseeker2

The webpage of the project is here: 1 Copy the link and download the software, at the time of writing this (April 2017), the latest version was 2.1.0 and the download URL was:

```
wget ``\ ``\ https://github.com/BSSeeker/BSseeker2/archive/BSseeker2-v2.1.8.tar.gz
<https://github.com/BSSeeker/BSseeker2/archive/BSseeker2-v2.1.8.tar.gz>`__
```

Once you have the code, uncompress it with:

```
tar -zxvf BSseeker2-v2.1.8.tar.gz
```

The submission script (runjob.pbs) will look like:

```
#!/bin/sh

#PBS -N ALIGN_${PBS_ARRAYID}
#PBS -l nodes=1:ppn=4
#PBS -l walltime=04:00:00
#PBS -l feature='!smp'
#PBS -m ae
#PBS -q standby
#PBS -t 0-279

module load compilers/python/2.7.13
module load genomics/bowtie2

if [ ${PBS_ARRAYID} -lt 10 ]
then
    JAID=00${PBS_ARRAYID}
elif [ ${PBS_ARRAYID} -lt 100 ]
then
    JAID=0${PBS_ARRAYID}
else
    JAID=${PBS_ARRAYID}
fi

SCRATCH=/scratch/[username]
REFS=${SCRATCH}/REFS
WGBS=${SCRATCH}/WGBS
TEMP=${SCRATCH}/TMP
OUTP=${SCRATCH}/OUTPUT
FQ1=${WGBS}/split/Estrogen_24_1.clean.fq_${JAID}
FQ2=${WGBS}/split/Estrogen_24_2.clean.fq_${JAID}
FA=${REFS}/Oncorhynchus_mykiss_chr.fa

echo 'Job start: ' `date`
cd $PBS_O_WORKDIR
cd BSseeker2_v2.1.0

mkdir -p ${TEMP}

python bs_seeker2-align.py -1 $FQ1 -2 $FQ2 -m 3 --aligner=bowtie2 -o ${OUTP}/Estrogen_24.
↳ ${JAID}.align -f sam -g $FA --temp_dir=${TEMP}

echo 'Job complete: ' `date`
```

There are a number of details to mention here. We cut the fq files in 2.4 million lines in order to be sure we can process them in our 4 hour walltime. The Job Array starts with 0 and ends at 279. This piece of code ensures 3 digit numbers filling with leading zeros as needed:

```
if [ ${PBS_ARRAYID} -lt 10 ]
then
    JAID=00${PBS_ARRAYID}
elif [ ${PBS_ARRAYID} -lt 100 ]
then
    JAID=0${PBS_ARRAYID}
else
    JAID=${PBS_ARRAYID}
fi
```

The rest of the script defines locations for references, the split files and output. Finally it launches BSseeker2. Remember to replace [username] with your username.

```
SCRATCH=/scratch/[username]
REFS=${SCRATCH}/REFS
WGBS=${SCRATCH}/WGBS
TEMP=${SCRATCH}/TMP
OUTP=${SCRATCH}/OUTPUT
FQ1=${WGBS}/split/Estrogen_24_1.clean.fq_${JAID}
FQ2=${WGBS}/split/Estrogen_24_2.clean.fq_${JAID}
FA=${REFS}/Oncorhynchus_mykiss_chr.fa

echo 'Job start: ' `date`
cd $PBS_O_WORKDIR
cd BSseeker2_v2.1.0

mkdir -p ${TEMP}

python bs_seeker2-align.py -1 $FQ1 -2 $FQ2 -m 3 --aligner=bowtie2 -o ${OUTP}/Estrogen_24.
-$JAID.align -f sam -g $FA --temp_dir=${TEMP}

echo 'Job complete: ' `date`
```

The command bowtie creates several threads, to ensure enough cores we set ppn=4.

7.4.5 4. Checking completion for all jobs

Sometimes jobs failed for one reason or another. It could be hard to check the output one by one to see if some job needs to be executed again. This small python script could help you with that task (BSresubmit.py)

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

import argparse
import os

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Process some integers.')
    parser.add_argument('-output_directory', '-d', metavar='PATH', type=str, help=
    'Directory for search output files', default='.')
    (continues on next page)
```

(continued from previous page)

```

parser.add_argument('-prefix', '-p', metavar='PREFIX', type=str, help='Filenames must be prefix+NUM+suffix', required=True)
parser.add_argument('-suffix', '-s', metavar='SUFFIX', type=str, help='Filenames must be prefix+NUM+suffix', required=True)
parser.add_argument('-min_split_number', '-m', metavar='N', type=int, help='min number of split', default=0)
parser.add_argument('-max_split_number', '-n', metavar='N', type=int, help='max number of split', required=True)

args = parser.parse_args()

failures=[]
for i in range(args.min_split_number, args.max_split_number+1):
    filename="%s%s%03d%s" % (args.output_directory,os.sep, args.prefix, i, args.suffix)
    if not os.path.isfile(filename):
        failures.append(i)
        continue

    rf=open(filename)
    data=rf.readlines()
    if not 'END' in data[-1]:
        failures.append(i)

if len(failures)==0:
    print("All the jobs are complete in the range")
else:
    print("Some jobs need to be resubmitted:")
    print(failures)

```

In order to check if all the jobs are complete execute:

```
`` ./BSresubmit.py -d OUTPUT -p Estrogen_24. -s .align.bs_seeker2_log -n 279``
```

Once you have all the data you can move forward and merge the files using samtools:

7.5 Visualization: VisIt

From: <https://wci.llnl.gov/simulation/computer-codes/visit>

VisIt is an Open Source, interactive, scalable, visualization, animation and analysis tool. From Unix, Windows or Mac workstations, users can interactively visualize and analyze data ranging in scale from small (<101 core) desktop-sized projects to large (>105 core) leadership-class computing facility simulation campaigns. Users can quickly generate visualizations, animate them through time, manipulate them with a variety of operators and mathematical expressions, and save the resulting images and animations for presentations. VisIt contains a rich set of visualization features to enable users to view a wide variety of data including scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured, adaptive and unstructured meshes. Owing to its customizable plugin design, VisIt is capable of visualizing data from over 120 different scientific data formats ([see this partial list](#)).

7.5.1 Introduction

VisIt is available on Spruce Knob in a client server model. In order to utilize Visit, you will need to install a local (client) copy of VisIt on your workstation/laptop. This will allow you to visualize your data on your workstation but be able to run VisIt against data stored in Spruce Knob filesystems. For any back-end processing that VisIt might need, it can also take advantage of Spruce Knob's processing power.

7.5.2 Client Installation

By the time this wikipage was reviewed (July 19, 2017) the latest version available was 2.12.3 released on June, 2017. Client executables are available from [1](#). Be sure to read the install notes for your particular installation need. In most cases you will use the 64 bit version appropriated for your OS (Windows, Mac or Linux)

Note: You will need to download a version that matches an available version on Spruce Knob. Currently we have 2.12.2 released on April 2017. If both client and server are 2.12.x they should work correctly

7.5.3 Running VisIt in the Client/Server Model

After successfully installing VisIt on your workstation, you can run through the following steps to start using it on in connection with Spruce Knob. **Note:** The following setups up VisIt to run in Parallel Mode but the same instructions apply for a serial run. When you enter the number of processors you need during the submission process, just select 1.

Configuring Client to use Spruce Knob as a Server

- Execute 'visit':

You can just use the icon or the command line. The command line is useful to identify problems during the first configuration of Visit to use Spruce as server. On MacOS the command line is accesible with:

`` /Applications/VisIt.app/Contents/Resources/bin/visit ``

- Go-to **Options** menu and select **Host Profiles**
- In the Host profiles window, select **New** and enter the following information on the **Host Settings** tab.
- Select the **Launch Profile** tab and select **New Profile** and enter the following information in the **Settings** sub tab.
- Select the **Parallel** sub tab and enter the following information. Note, you can use a different **Queue** setting but in most cases standby should be sufficient unless you suspect your task will last longer than 4 hours.
- Within the **Parallel** sub tab, select the **Advanced** sub tab, and enter **module load visualization/visit/2.10.2** in the **Sublauncher pre-mpi command** field. Screen should look as follows:
- Select **Apply** in the lower left corner.
- Select **Options** and **Save Settings** from the menu on the main VisIt user interface.

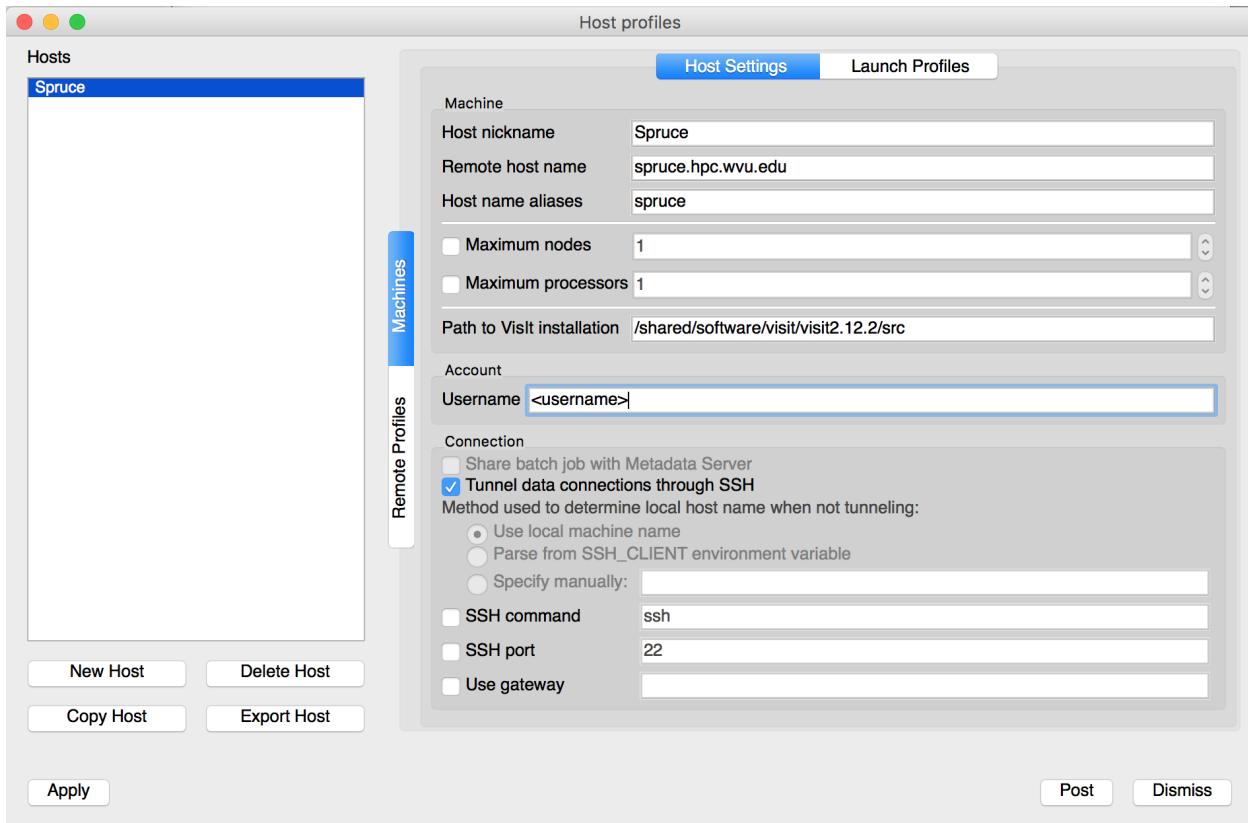


Fig. 1: host_settings_1.png

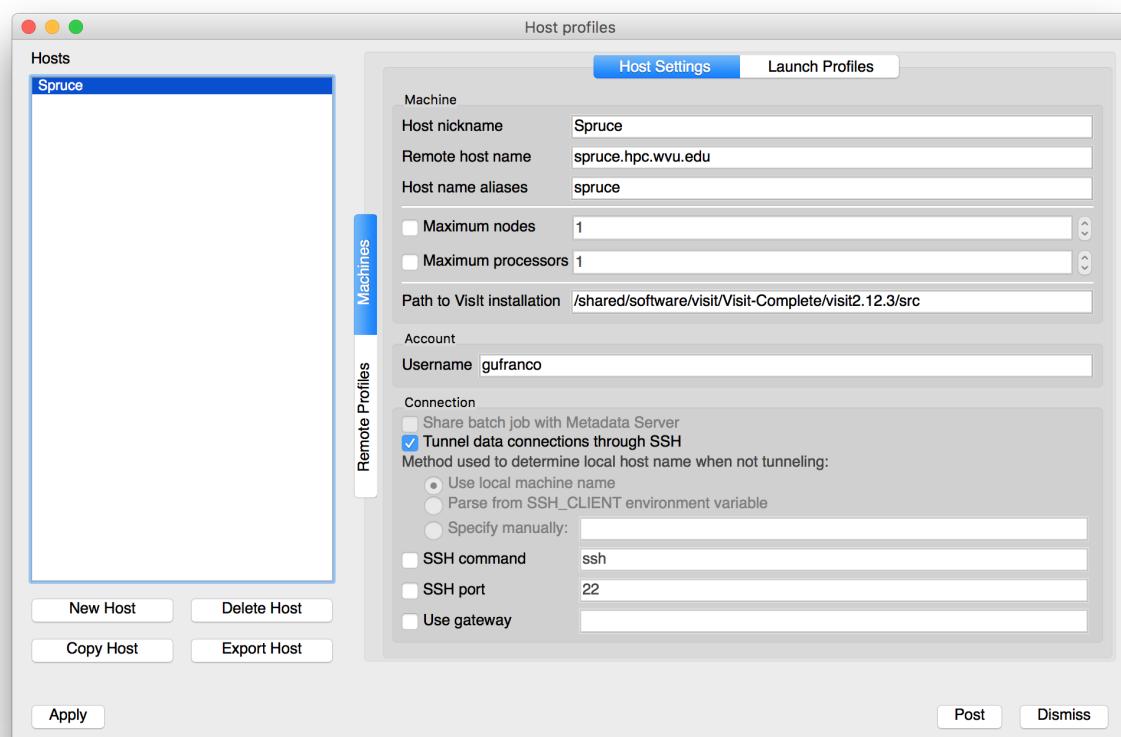


Fig. 2: host_settings_2.png

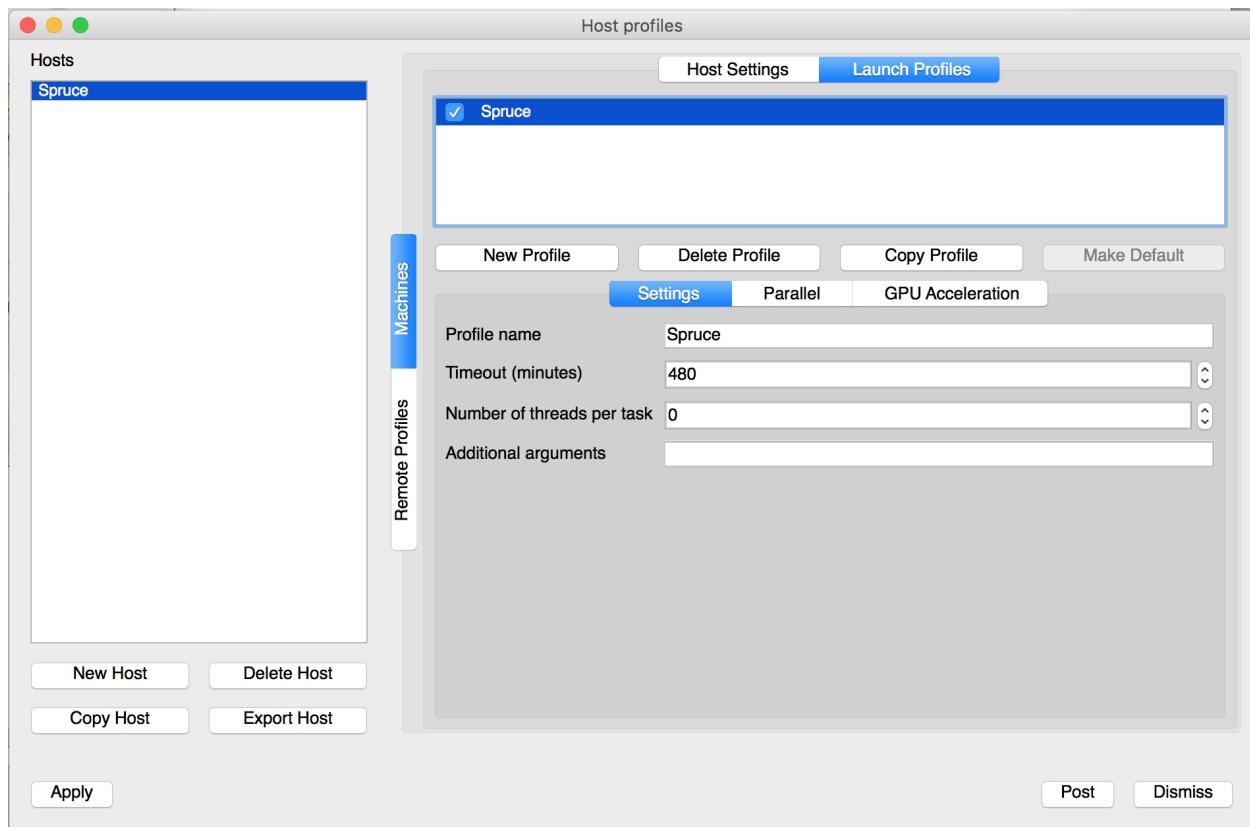


Fig. 3: launch_profiles_1.png

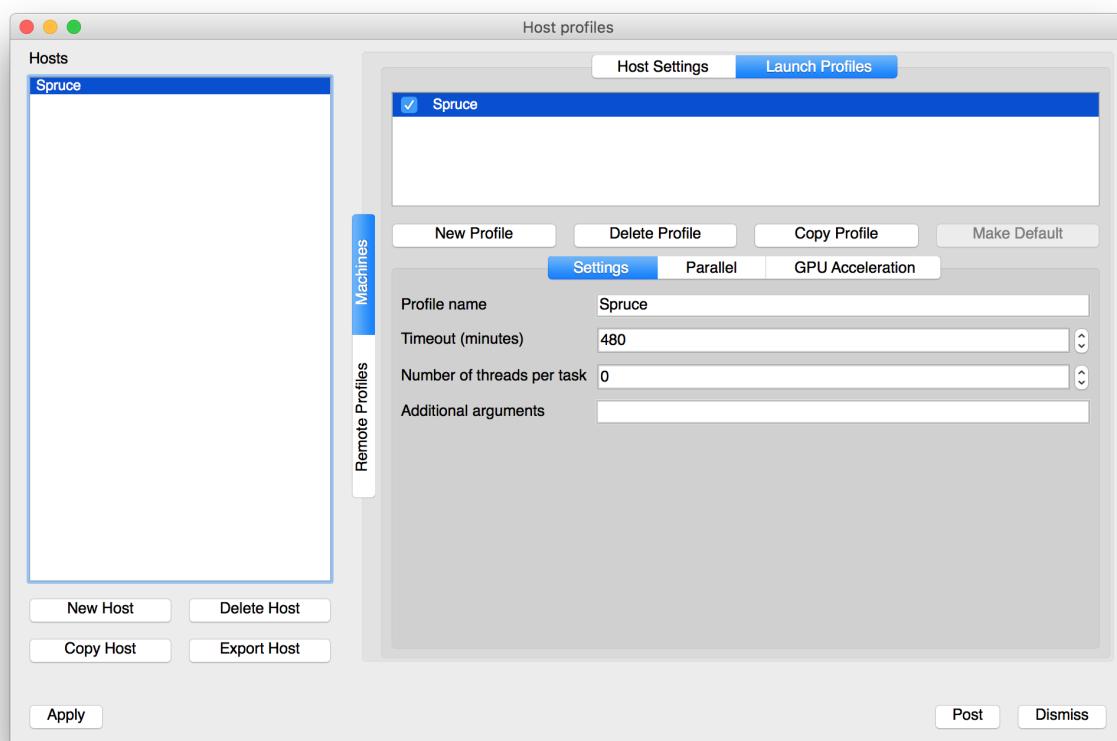


Fig. 4: launch_profiles_2.png

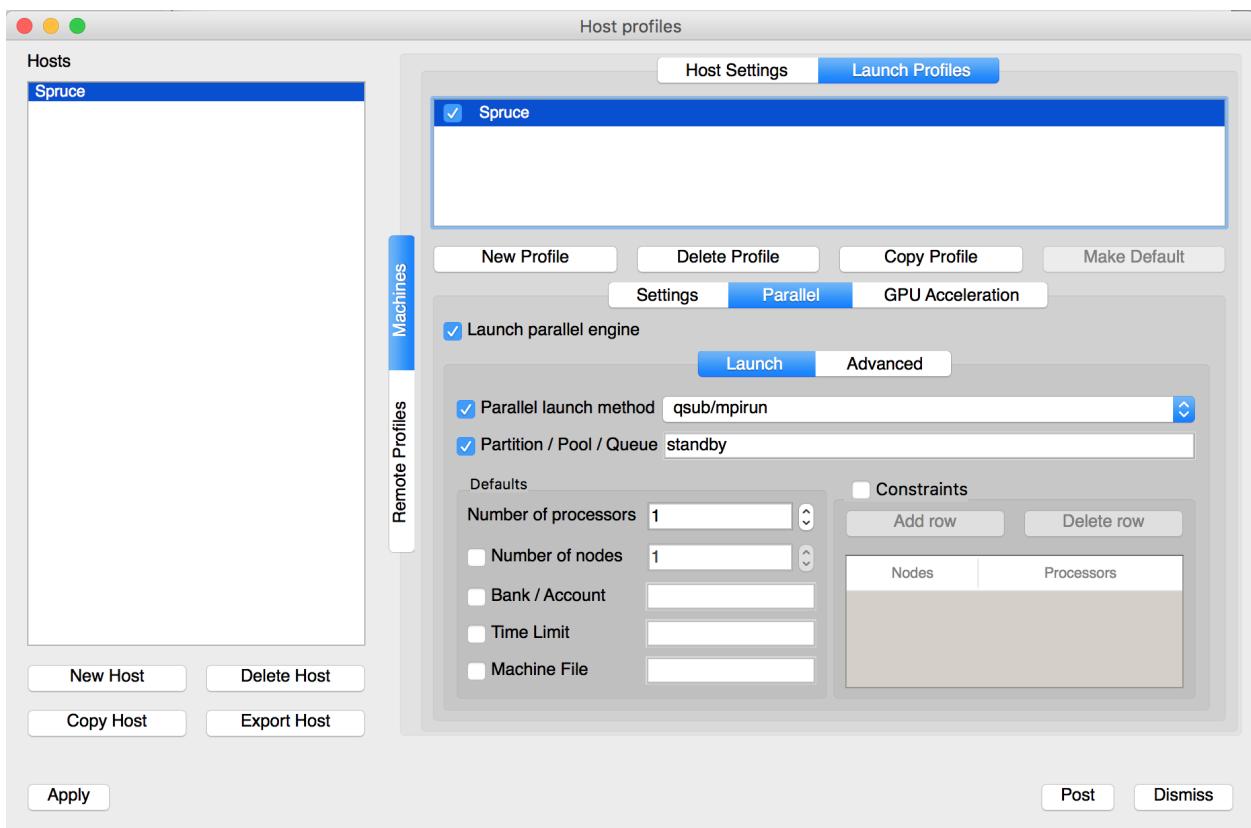


Fig. 5: parallel_options_1.png

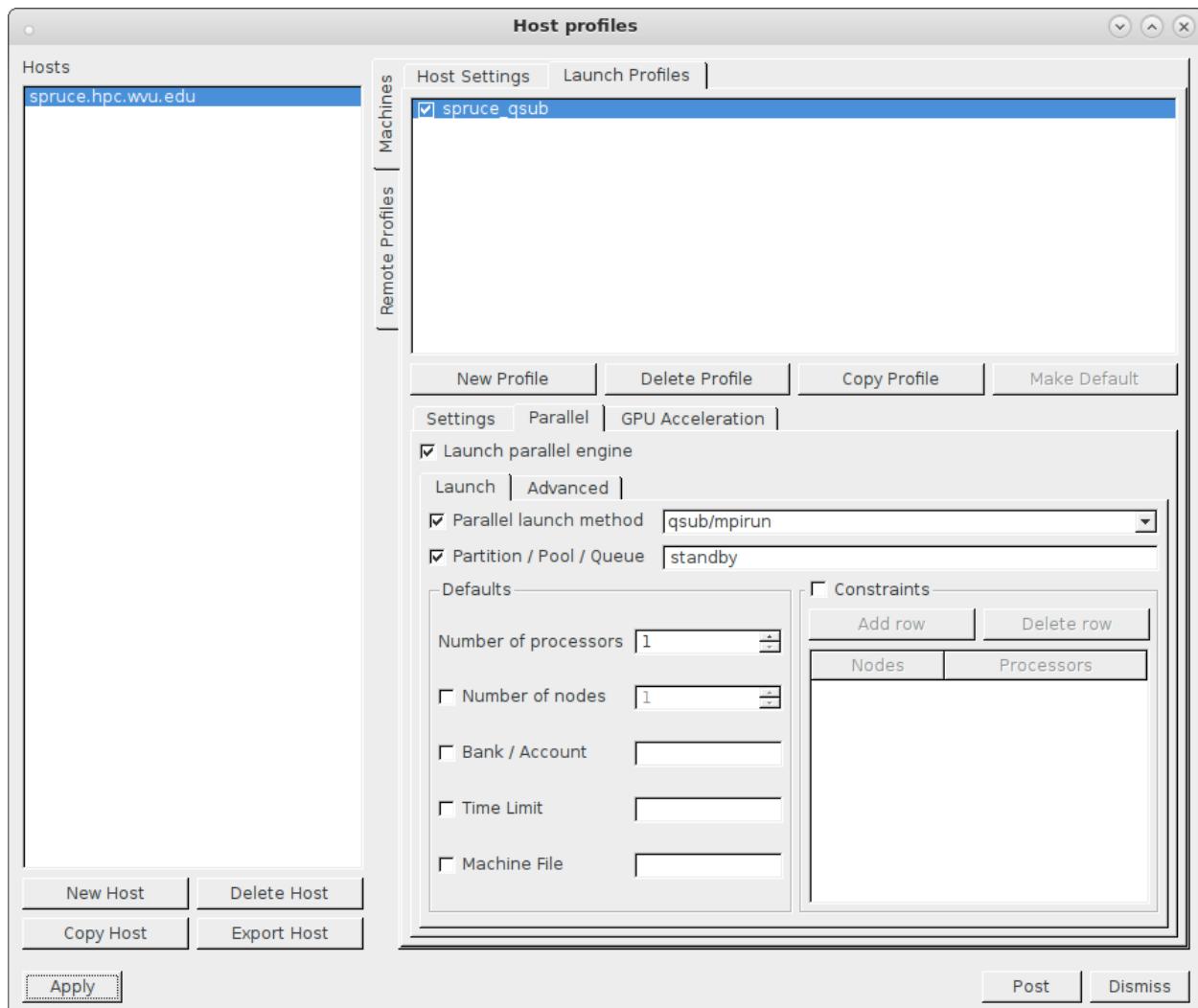


Fig. 6: parallel_options_2.png

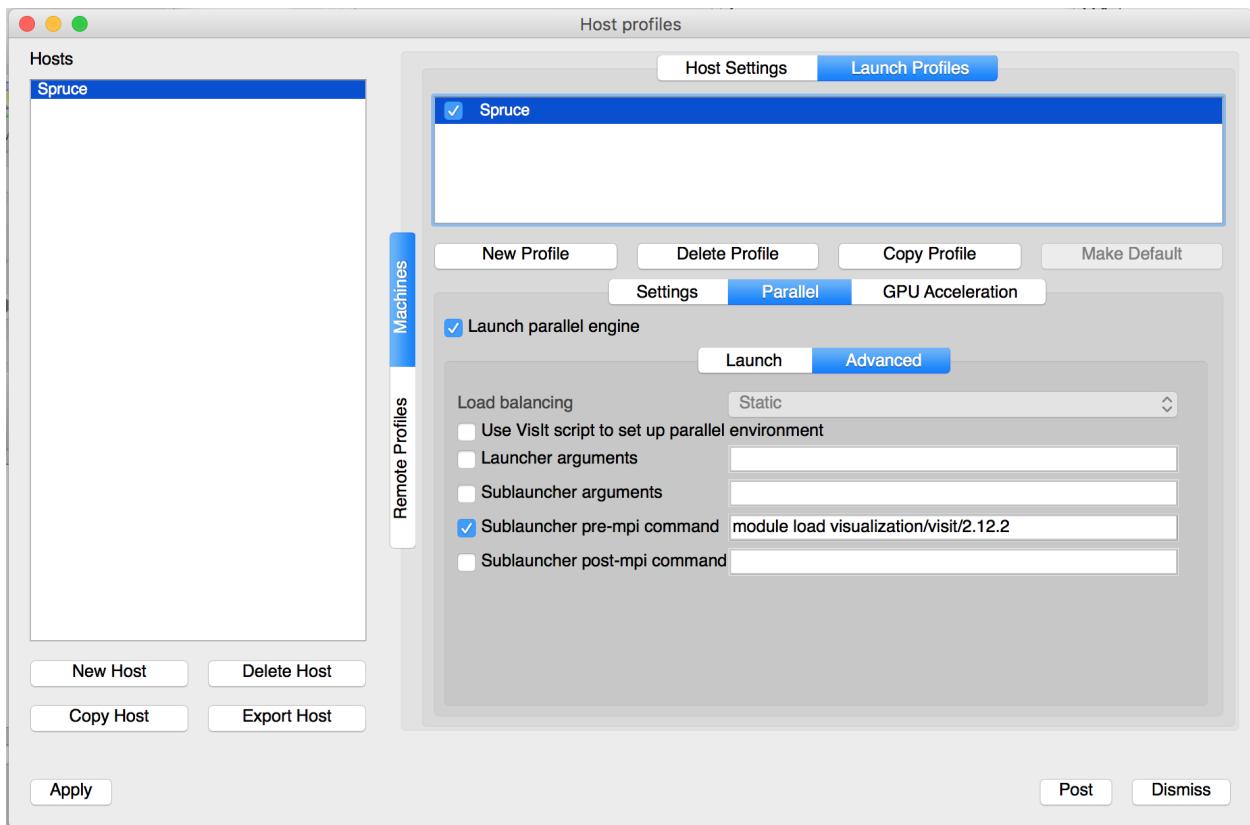
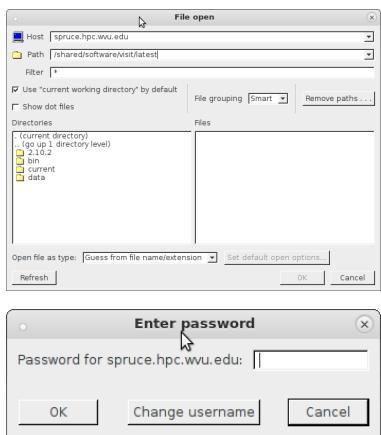


Fig. 7: parallel_advanced.png

Connect your client to Spruce Knob

- If VisIt is not already running on your workstation, start it by executing **visit** on the command line.
- Select **Open** under the **Sources** section.
- Select **spruce.hpc.wvu.edu** in the Host Section and enter your password when prompt. You should see screen similar to the following.



- You should now see the contents of your home directory and ready to use VisIt on your local data.

Visualize an Example Visit Dataset

Note: This assumes you have already connected to Spruce Knob from your client workstation following the instructions from the previous section.

- Change **Path** to **/shared/software/visit/latest/data** and select **globe.silo**.
- If necessary, change the options for job submission and select **OK**.
- The following dialog box will appear until a spot is open on the cluster for your job to run. When using the standby queue, this should be less than 60 seconds assuming the cluster is not completely busy.
- In **Plots** select **Pseudocolor** and **disp_magnitude**.
- Select **Draw** and you should see an output similar to the following.

7.6 Compiling Planetary Modeling Packages

These instructions describe how to compile various packages for Earth modeling, climate, weather and related applications.

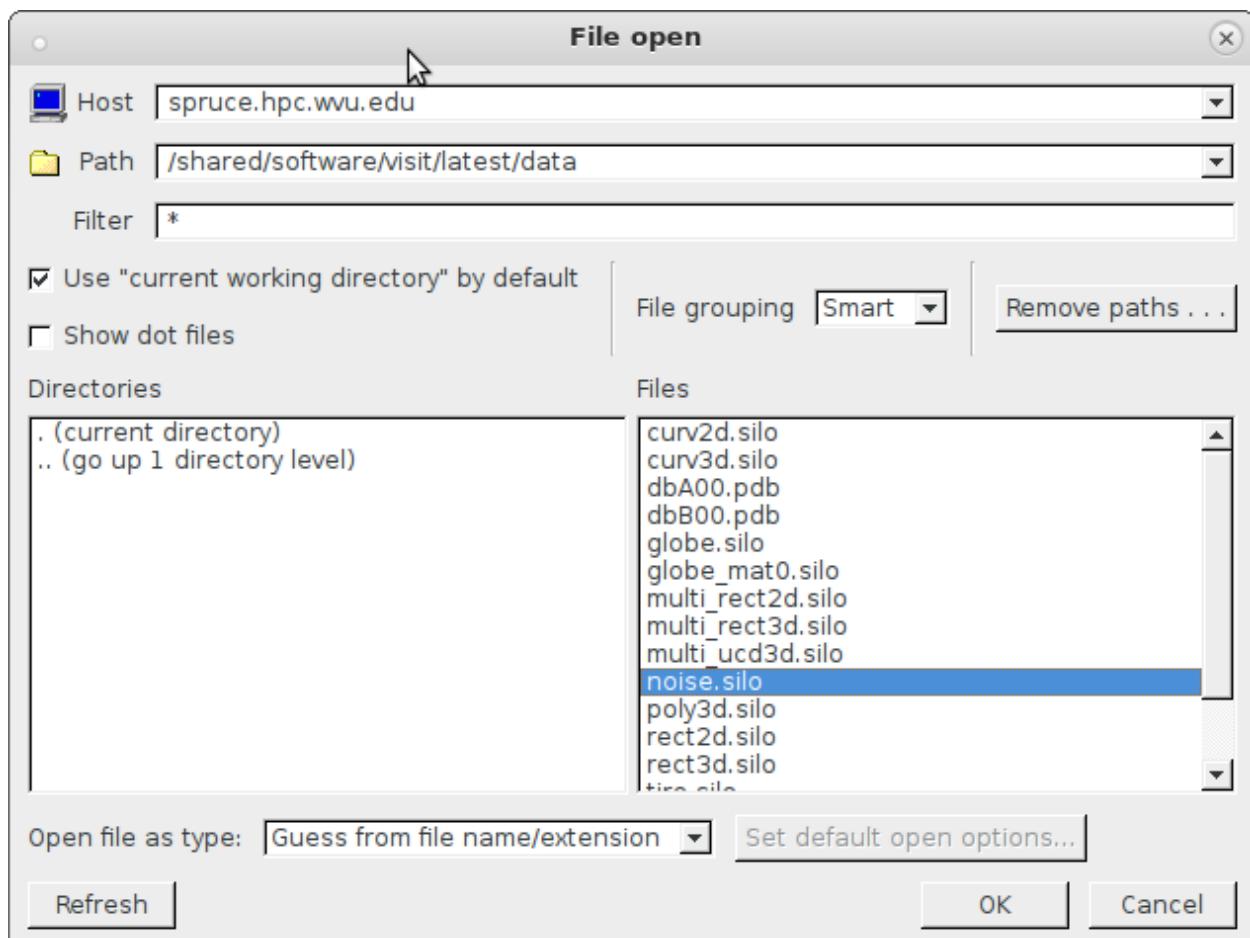


Fig. 8: example_data.png

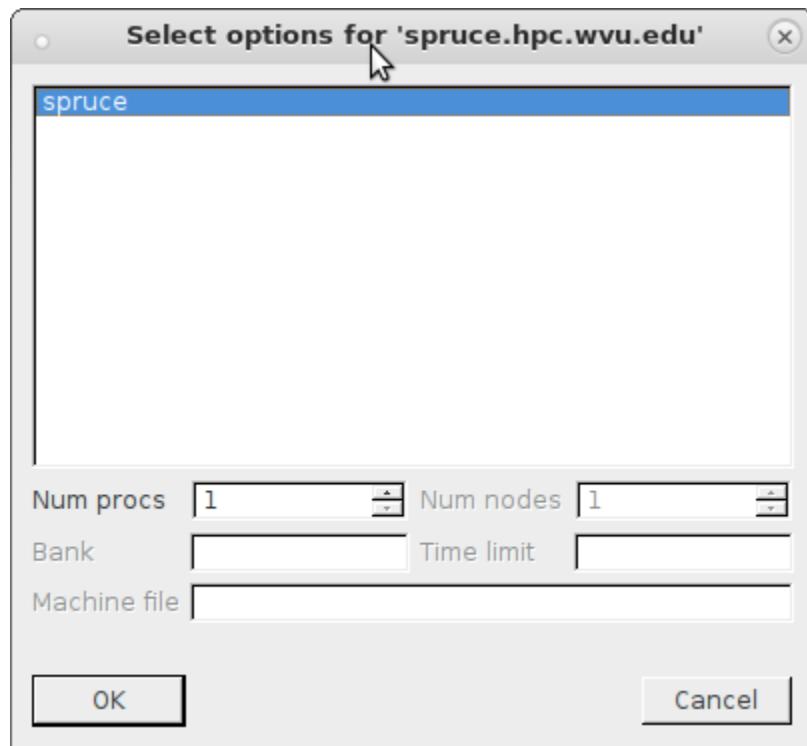


Fig. 9: select_options.png

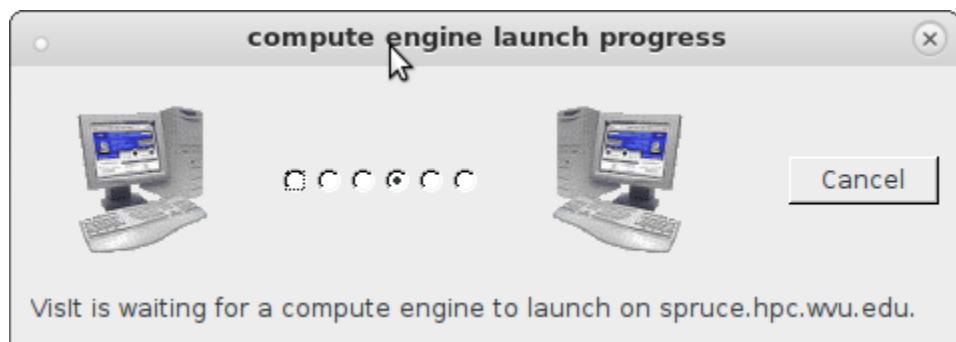


Fig. 10: compute_engine.png

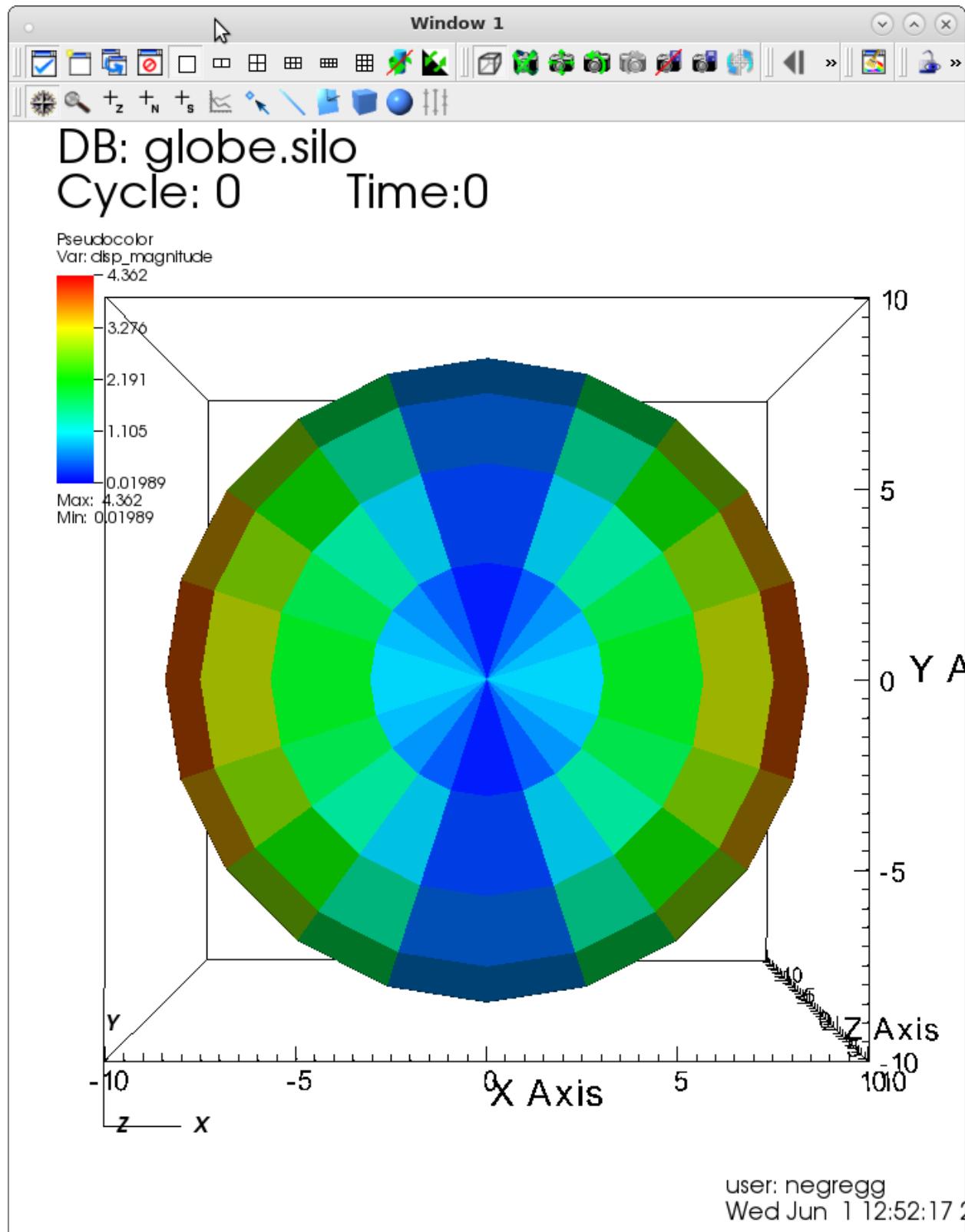


Fig. 11: visit_output.png

7.6.1 Earth System Modeling Framework (ESMF)

The Earth System Modeling Framework (ESMF) is high-performance, flexible software infrastructure for building and coupling weather, climate, and related Earth science applications. The ESMF defines an architecture for composing complex, coupled modeling systems and includes data structures and utilities for developing individual models.

We describe how the code was compiled for Thorny Flat using Intel Compiler Suite and MPI libraries.

Download the code

The code can be clone with git from:

```
git clone https://git.code.sf.net/p/esmf/esmf
```

The code was cloned at this at /shared/src/planetary/esmf

Setting the environment variables

The building system is driven by a set of environment variables that need to be exported before executing make. This compilation will be done exclusively using Intel compilers. First we need to load the module for Intel 2018:

```
module load lang/intel/2018_u4
```

The variables that need to be exported are:

```
export ESMF_DIR=/shared/src/planetaryesmf
export ESMF_COMM=intelmpi
export ESMF_COMPILER=intel
export ESMF_INSTALL_PREFIX=/shared/software/planetary/esmf
```

Compiling the code

To compile the code just execute:

```
make
```

The compilation takes a while, notice that the first lines shows the values of several environment variables, the 4 that we just set and a few others internal to the makefile:

```
-----
ESMF_VERSION_STRING: 8.0.0 beta snapshot
ESMF_8_0_0_beta_snapshot_30-48-g8e8a213
-----
# On branch master
nothing to commit, working directory clean
-----
Make version:
GNU Make 3.82
Built for x86_64-redhat-linux-gnu
Copyright (C) 2010 Free Software Foundation, Inc.
```

(continues on next page)

(continued from previous page)

License GPLv3+: GNU GPL version 3 or later <<http://gnu.org/licenses/gpl.html>>
 This is free software: you are free to change and redistribute it.
 There is NO WARRANTY, to the extent permitted by law.

 Fortran Compiler version:

```
mpiifort for the Intel(R) MPI Library 2018 Update 4 for Linux*
Copyright(C) 2003-2018, Intel Corporation. All rights reserved.
Intel(R) Fortran Intel(R) 64 Compiler for applications running on Intel(R) 64, Version
  ↵18.0.5.274 Build 20180823
Copyright (C) 1985-2018 Intel Corporation. All rights reserved.
```

 C++ Compiler version:

```
mpiicpc for the Intel(R) MPI Library 2018 Update 4 for Linux*
Copyright(C) 2003-2018, Intel Corporation. All rights reserved.
Intel(R) C++ Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 18.0.
  ↵5.274 Build 20180823
Copyright (C) 1985-2018 Intel Corporation. All rights reserved.
```

 Preprocessor version:

```
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

 * User set ESMF environment variables *
ESMF_COMM=intelmpi
ESMF_COMPILER=intel
ESMF_DIR=/shared/src/planetary/esmf
ESMF_INSTALL_PREFIX=/shared/software/planetary/esmf

 ...
 ...

At the end of compilation you should get:

```
make[2]: Leaving directory `/gpfs20/shared/src/planetary/esmf'
make[1]: Leaving directory `/gpfs20/shared/src/planetary/esmf'
make[1]: Entering directory `/gpfs20/shared/src/planetary/esmf'
make[1]: Nothing to be done for `build_tracelibs'.
make[1]: Leaving directory `/gpfs20/shared/src/planetary/esmf'
make[1]: Entering directory `/gpfs20/shared/src/planetary/esmf'
make[1]: Leaving directory `/gpfs20/shared/src/planetary/esmf'
ESMF library built successfully on Mon Apr 15 11:30:40 EDT 2019
To verify, build and run the unit and system tests with: make check
```

(continues on next page)

(continued from previous page)

```
or the more extensive: make all_tests
```

The tests can be performed with:

```
make all_tests
```

After the tests you should see and output like this:

```
The stdout files for the examples can be found at:  
/shared/src/planetary/esmf/examples/examples0/Linux.intel.64.intelmpi.default  
  
Found 85 multi-processor examples, 85 passed and 0 failed.  
  
make[2]: Leaving directory `/gpfs20/shared/src/planetary/esmf'  
  
SYSTEM TESTS SUMMARY  
Found 45 multi-processor system tests, 45 passed and 0 failed.  
  
EXAMPLES SUMMARY  
Found 85 multi-processor examples, 85 passed and 0 failed.  
  
UNIT TESTS SUMMARY  
Found 3460 non-exhaustive multi-processor unit tests, 3460 passed and 0 failed.  
  
make[1]: Leaving directory `/gpfs20/shared/src/planetary/esmf'
```

Installing ESMF

Finally, install the libraries and supporting binaries with:

```
make install
```

7.6.2 Thermosphere Ionosphere Electrodynamics General Circulation Model (TIE-GCM)

The NCAR Thermosphere-Ionosphere-Electrodynamics General Circulation Model (TIE-GCM) is a comprehensive, first-principles, three-dimensional, non-linear representation of the coupled thermosphere and ionosphere system that includes a self-consistent solution of the middle and low-latitude dynamo field. The model solves the three-dimensional momentum, energy and continuity equations for neutral and ion species at each time step, using a semi-implicit, fourth-order, centered finite difference scheme on each pressure surface in a staggered vertical grid. It can be run in either serial or parallel mode on a variety of platforms, including Linux workstations and supercomputers. The time step is typically 120 s.

Download the code

The code can be downloaded from [HAO-UCAR webpage](#)

After registering 3 files need to be downloaded for TIE-GCM 2.0:

```
tiegcm2.0.tar
tiegcm2.0_res2.5_data.tar
tiegcm2.0_res5.0_data.tar
```

The files can be uncompress with the command:

```
tar -xvf tiegcm2.0.tar
tar -xvf tiegcm2.0_res2.5_data.tar
tar -xvf tiegcm2.0_res5.0_data.tar
```

Editing the make configuration

Once you have downloaded and uncompress the files. You need to adapt/create a Make configuration file suitable for the cluster were the model will run. Go to `tiegcm2.0/scripts` and make a copy of `Make.intel_hao64` as we will use it as template for our cluster. This tutorial assumes that you are using Thorny Flat, so copy the file as `Make.intel_thorny`:

```
cd tiegcm2.0/scripts
cp Make.intel_hao64 Make.intel_thorny
```

Modify `Make.intel_thorny` to match the file below:

```
# 
# Included makefile for Intel ifort compiler with openmpi on 64-bit HAO machines.
# Intel ifort (IFORT) 12.0.0 20101006
#
F90      = ifort
MPIF90   = mpiifort
MPIRUN   = mpirun -l -s all
FFLAGS   = -r8 -heap-arrays
DBGFLAGS = -debug full -traceback
DBGFLAGS += -check bounds -check format -check output_conversion -check pointers -check_
            uninit
DBGFLAGS += -fpe-all=0 # this traps all floating point exceptions
#
# Makefile will use OPTIM = -g if set debug=TRUE in job script.
OPTIM    = -O3
LIBS     = -lcurl -lnetcdf -lnetcdff
HOST     = $(shell hostname)
#
# Library and Include file paths:
#
LIB_NETCDF = /shared/software/libs/netcdf/4.x_intel18_impi18/lib
INC_NETCDF = /shared/software/libs/netcdf/4.x_intel18_impi18/include
#
# This public release of ESMF was built at hao with intel on Dec 22, 2015:
#
```

(continues on next page)

(continued from previous page)

```

LIB_ESMF = /shared/software/planetary/esmf/lib/lib0/Linux.intel.64.intelmpi.default
#
# Make machines.ini file for MPI execution:
#
prereq: machines.ini mpirun.command
machines.ini: export HN=$(HOST)
machines.ini: export NP=$(NPROC)
machines.ini: FORCE
    @echo "Making machines.ini.."
    @echo `hostname` > machines.ini
    @awk 'BEGIN{ for (i=2; i <= ENVIRON["NP"]; i++) print ENVIRON["HN"] }' >>_>
machines.ini

mpirun.command: FORCE
    @echo "Making mpirun.command: MPIRUN=$(MPIRUN)"
    @echo $(MPIRUN) > mpirun.command

FORCE:

```

Notice that we have change the compilers, as loading the module for Intel 2018 will add them to the search \$PATH. It is important to add -lncdf -lnetcdf as newer versions of NetCDF separate the fortran and C interfaces and both libraries need to be loaded. You should also load the module for netcdf:

```
module load libs/netcdf/4.x_intel18_impi18
```

In the previous section we have compiled and installed ESMF. The location on Thony Flat being /shared/software/planetary/esmf so the variable LIB_ESMF reflects the location for ESMF libraries.

This Make configuration file should remain in tiegcm2.0/scripts and will be captured by the execution script.

Editing the execution script

Now we have to edit the execution script tiegcm-thorny.job. It is a good idea to make a copy of it before editing:

```
cp tiegcm-linux.job tiegcm-thorny.job
```

There just a few changes to introduce here. The whole script will not be shown here, just the pieces that needs some changes. The first portion should look like this:

```

#!/bin/csh
#
# Job script to build and execute the model on a 64-bit Linux desktop machine.
#
# User must set shell variables below:
#
#   modeldir: Root directory to model source (may be an SVN working dir)
#   execdir: Directory in which to build and execute (will be created if necessary)
#   tgcmdir: Directory in which startup history and data files are accessed.
#             (If tgcmdir is not set, the model will use env var TGCMDATA)
#   input:   Namelist input file for the chosen model resolution
#   output:  Stdout file from model execution (will be created)
#   modelres: Model resolution (5.0 or 2.5 degrees)

```

(continues on next page)

(continued from previous page)

```

# make:      Build file with platform-specific compile parameters (in scripts dir)
# mpi:       TRUE/FALSE for MPI run (non-MPI runs are not supported in v2.0 and later)
# debug:     If TRUE, build and execute a "debug" run (debug compiler flags are set)
# exec:      If TRUE, execute the model (build only if exec is FALSE)
# utildir:   Directory containing supporting scripts (default $modeldir/scripts)
# runscript: LSF script with run commands (submitted with bsub from execdir)
#
# To switch to 2.5-deg resolution, set modelres below to 2.5,
# and change execdir, tgcmdata and namelist input if necessary.
#
set modeldir = tiegcm2.0
set execdir = tiegcm.exec
set tgcmdata = tiegcm_res2.5_data
set input = tiegcm_res2.5.inp
set output = tiegcm_res2.5.out
set modelres = 2.5
set make = Make.intel_thorny
set mpi = TRUE
set nproc = 20
set debug = FALSE
set exec = TRUE
set utildir = $modeldir/scripts
...

```

Notice that the make file changed to `Make.intel_thorny`. Next, the number of processors is fixed to 20, notice that on Thorny Flat we have nodes with 24 and 40. The best number depends on how intense in the calculation. For 5.0 degrees the code completes in just a few minutes. Remember to change `modelres` accordingly: 5.0 or 2.5 degrees and related files.

The final change is half down the script were there is a conditional for using the intel compiler. Around line 210 I include a condition to set the compiler to intel:

```

set compiler = 'unknown'
if ($make == Make.intel_hao64) then
    set compiler = intel
else if ($make == Make.pgi_hao64) then
    set compiler = pgi
else if ($make == Make.gfort_hao64) then
    set compiler = gfort
else if ($make == Make.intel_thorny) then
    set compiler = intel
endif
echo Compiler: $compiler

```

Using 20 cores, it takes around 7 minutes on Thorny Flat to get the results:

```

Executing mpirun -l -s all with executable ./tiegcm2.0 at Tue Apr 16 10:32:11 EDT 2019
Linux MPI run of ./tiegcm2.0 completed at Tue Apr 16 10:39:45 EDT 2019
Overwriting file /gpfs20/shared/src/planetary/tiegcm2.0/tiegcm_res2.5.out with non-
→ASCII characters removed.
mklogs: Extracting log files from /gpfs20/shared/src/planetary/tiegcm2.0/tiegcm_res2.5.
→out

```

7.6.3 The Global Ionosphere/Termosphere Model (GITM)

GITM is a 3-dimensional spherical code that models the Earth's thermosphere and ionosphere system using a stretched grid in latitude and altitude. The number of grid points in each direction can be specified, so the resolution is extremely flexible. GITM explicitly solves for the neutral densities of O, O₂, N(2D), N(2P), N(4S), N₂, and NO; and ion species O+(4S), O+(2D), O+(2P), O₂+, N+, N₂+, and NO+. One major difference between GITM and other thermosphere codes is the use of an altitude grid instead of a pressure grid. The vertical grid spacing is less than 3 km in the lower thermosphere, and over 10 km in the upper thermosphere. GITM allows for non-hydrostatic solutions to develop (i.e., the full vertical momentum equation is solved), so more realistic dynamics in the auroral zone can be simulated.

The procedure to compile the code is not properly documented, so instructions here are specific for our clusters, in particular Thonry Flat.

Cloning the repository

The package can be cloned from the Github page:

```
git clone https://github.com/aaronjridley/GITM.git
```

Configuring the build

The build system is managed by a Perl script located on the root folder `Config.pl`. All documentation about building this code is provided by executing:

```
./Config.pl -h
```

We will compile the code on Thonry Flat with support for MPI and HDF5, the modules to load are:

```
module load lang/intel/2018_u4 libs/hdf5/1.10.5_intel18_impi18
```

To configure the build for Intel compilers and support for Parallel HDF5 execute:

```
./Config.pl -install -compiler=mpiifort,icc -hdf5
```

The command should produce an output like this:

```
Installing GITM2
touch Library/src/Makefile.DEPEND Library/src/Makefile.RULES
cp src/ModSize.f90.orig src/ModSize.f90
touch src/Makefile.DEPEND src/Makefile.RULES srcInterface/Makefile.DEPEND
Enabling HDF5 library in Makefile.conf
set_hdf5_: cp share/Library/src/ModHdf5Utils_orig.f90 share/Library/src/ModHdf5Utils.f90
set_spice_: cp share/Library/src/ModSpice_empty.f90 share/Library/src/ModSpice.f90
Configuring GITM for Earth!!
```

Any attempt to recompile with different settings usually requires removing all the sources and clone the repository back, several pieces of code remain even executing `./Config.pl -uninstall` or `make clean`. This is particularly true if you change compiler.

Building the code

After configuration several make configuration files are copied to the root folder of GITM and the code is ready for compilation. Execute the configure script again to confirm yours settings:

```
$ ./Config.pl

GITM2 is installed in directory /gpfs20/shared/src/planetary/aaronjridley/GITM
as a stand-alone code.
The installation is for the Linux operating system.
Makefile.conf was created from share/build/Makefile.Linux.mpiifort
and share/build/Makefile.icc
The selected F90 compiler is ifort.
The selected C compiler is icc.
The default precision for reals is double precision.
The maximum optimization level is -O4
Debugging flags: no
Linked with MPI: yes
Linked with HDF5: yes
Linked with HYPRE: no
Linked with FISHPAK: no
Linked with SPICE: no
Number of cells in a block: nLon=9, nLat=9, nAlt=50
Max. number of blocks : MaxBlock=4
Planet=Earth
```

Execute make to compile the code:

```
make
```

The binary GITM.exe will be located at src folder.

7.7 NAMD

Nanoscale Molecular Dynamics (NAMD, formerly Not Another Molecular Dynamics Program) is computer software for molecular dynamics simulation, written using the Charm++ parallel programming model. It is noted for its parallel efficiency and is often used to simulate large systems (millions of atoms). It has been developed by the collaboration of the Theoretical and Computational Biophysics Group (TCB) and the Parallel Programming Laboratory (PPL) at the University of Illinois at Urbana–Champaign.

In this tutorial we will demonstrate the very basics of running NAMD on our clusters. NAMD is particularly adapted for several paradigms in parallel computing from multithreading to distributed parallel programming and support for GPUs.

The purpose of this tutorial is not to teach the chemistry and parameters more suited for a given problem. We will just show run NAMD jobs on our clusters. NAMD offers a user guide to understand how to prepare simulations from the chemical point of view.

<https://www.ks.uiuc.edu/Research/namd/2.13/ug/>

The tutorial will use a couple of examples from the set of benchmarks available for NAMD

<https://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz>

<https://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz>

7.7.1 Downloading the input files

Select a good location for downloading the inputs and executing the simulations. For example use \$SCRATCH/NAMD:

```
mkdir $SCRATCH/NAMD  
cd $SCRATCH/NAMD
```

Download the two examples from the NAMD UIUC webserver:

```
wget https://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz  
wget https://www.ks.uiuc.edu/Research/namd/utilities/stmv.tar.gz
```

The examples are compressed, execute this command to uncompress them on the current folder:

```
tar -zxvf apoa1.tar.gz  
tar -zxvf stmv.tar.gz
```

7.7.2 Preparing the submission script

We will use apoa1 for demonstrate how to create a submission script. ApoA1 has 92K atoms and should run efficiently on a single node.

First, go to the recently created folder apoa1 and start creating a file runjob.pbs with your text editor of choice:

```
$> cd apoa1  
$> vim runjob.slurm
```

The content of the submission could be like this:

```
#!/bin/bash  
  
#SBATCH --job-name=NAMD  
#SBATCH -N 1  
#SBATCH -n 2  
#SBATCH -p standby  
  
module purge  
module load atomistic/namd/2.13_multicore  
  
cd $SLURM_SUBMIT_DIR  
namd2 +p2 +setcpuaffinity apoa1.namd
```

The first line is called a shebang and indicates that the file below was written in bash, one of the several shell interpreters in Linux, for this simple example, there is nothing particular for bash and other interpreters are possible (csh, ksh, dash) or the system /bin/sh that in the case of Spruce is a symbolic link to bash itself.

The next 4 lines start with #SBATCH. Those are comments for bash but are interpreted by *sbatch* and they are important for requesting resources and configuring several aspects of non-interactive jobs.

First is the name of the job “NAMD”, totally arbitrary string to identify the job. Second is the line requesting resources. In this case we are requesting 2 cores on a single node. Third, is the queue where the job will execute, “standby” is actually the default queue so the line is not necessary here, but it is important to use it if you plan to run on a different queue. The fourth line attach the standard error with the standard output. NAMD will almost never write on the standard error, so using this command prevents us from having empty files at the end of each simulation.

The next section clean the list of modules and load the corresponding module for NAMD. In this case we are using `atomistic/namd/2.13_multicore` that is intended only for executions on a single node.

Next line changes the working directory to the place where the submission script and inputs are located and from we will submit the job.

Finally the command for NAMD. `namd2 +p2 +setcpuaffinity apoa1.namd`. The command tells NAMD to use 2 cores and make those two cores sit on the same socket optimizing the usage of Cache Level 3 in some cases. The input file for NAMD is `apo1.namd` and that completes the submission script.

7.7.3 Submitting the job

Submit the job with the following command:

```
$> sbatch runjob.slurm
```

You should get a line indicating the JobID, I got this:

```
4731853
```

You can use the number to monitor the state of the job.

For example executing:

```
$ squeue -u $USER

active jobs-----
JOBID          USERNAME      STATE PROCS   REMAINING        STARTTIME
4731853        <username>    Running     2      3:59:35  Fri Nov  1 13:14:12
1 active job           2 of 3360 processors in use by local jobs (0.06%)
                           1 of 175 nodes active (0.57%)
```

The job is now running and received 4 hours to complete, that is the walltime on standby queue.

With `showq` you can monitor the status of your jobs, if they are in queue or running.

This is an small job that completes in minutes. Once the job is complete you will see new files on the folder where you execute the job.

The file `NAMD.o<JobID>` will contain all the standard output and error produced by NAMD during execution. For the job above, my file is `NAMD.o4731853`

Read the file with care, pay attention to warnings and errors. For the sake of simplicity we will just show the last lines:

```
ENERGY:      499      20158.0682      19847.7298      5722.4089      178.6361      -
→ 337058.4824      23219.2403      0.0000      0.0000      45439.2653      -222493.      -
→ 1338      165.2956      -267932.3991      -222047.2061      165.2956      -2020.8690      -
→ -2376.3392      921491.4634      -2020.8690      -2376.3392

Info: Benchmark time: 2 CPUs 0.384412 s/step 4.44921 days/ns 542 MB memory
TIMING: 500 CPU: 195.068, 0.381542/step Wall: 195.075, 0.381408/step, 0 hours
→ remaining, 542.000000 MB of memory in use.

ETITLE:      TS          BOND          ANGLE          DIHED          IMPRP          -
→ ELECT          VDW          BOUNDARY          MISC          KINETIC          TOTAL      -
→ TEMP          POTENTIAL          TOTAL3          TEMPAVG          PRESSURE          -
→ GPRESSURE          VOLUME          PRESSAVG          GPRESSAVG          (continues on next page)
```

(continued from previous page)

```

ENERGY:      500      20974.8941      19756.6569      5724.4523      179.8271      -
      ↵337741.4189      23251.1007      0.0000      0.0000      45359.0789      -222495.
      ↵4088      165.0039      -267854.4877      -222061.0906      165.0039      -3197.5170      ↵
      ↵ -2425.4142      921491.4634      -3197.5170      -2425.4142

WRITING EXTENDED SYSTEM TO OUTPUT FILE AT STEP 500
WRITING COORDINATES TO OUTPUT FILE AT STEP 500
The last position output (seq=-2) takes 0.002 seconds, 542.000 MB of memory in use
WRITING VELOCITIES TO OUTPUT FILE AT STEP 500
The last velocity output (seq=-2) takes 0.002 seconds, 542.000 MB of memory in use
=====

WallClock: 197.820267  CPUTime: 196.267166  Memory: 542.000000 MB
[Partition 0] [Node 0] End of program

```

This information is useful for adjusting your request for resources for additional and similar jobs. In particular this job took around 4 minutes and 540 MB of memory.

7.7.4 Running NAMD on GPUs

Now we will demonstrate the changes to be done when running with GPUs Go one folder up where the tar.gz and apoal folder is located:

```
cd $SCRATCH/NAMD
```

Make a copy of the apoal folder, we will introduce a few changes to the submission script:

```
cp -r apoal apoal-CUDA
cd apoal-CUDA
rm -rf NAMD.o*
```

Edit the submission script like this:

```
#!/bin/bash

#PBS -N NAMD
#PBS -l nodes=1:ppn=2
#PBS -q comm_gpu
#PBS -j oe

module purge
module load atomistic/namd/2.13_multicore-CUDA

nvidia-smi

cd $PBS_O_WORKDIR

namd2 +p2 +setcpuaffinity apoal.namd
```

We are now using `comm_gpu` as the queue, that will give us machines with GPUs and load the module for NAMD that support GPUs

Submit the job as usual:

```
qsub runjob.pbs
```

After completion you will see a file NAMD.o<JobID> with a content like this:

```
Fri Nov  1 13:36:10 2019
+-----+
| NVIDIA-SMI 396.26          Driver Version: 396.26 |
+-----+
| GPU  Name      Persistence-M| Bus-Id     Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+=====+=====
|   0  Tesla K20m        Off  | 00000000:08:00.0 Off |                0 |
| N/A  27C    P0    49W / 225W |     0MiB /  4743MiB |      0%     Default |
+-----+
|   1  Tesla K20m        Off  | 00000000:24:00.0 Off |                0 |
| N/A  41C    P0    49W / 225W |     0MiB /  4743MiB |      0%     Default |
+-----+
|   2  Tesla K20m        Off  | 00000000:27:00.0 Off |                0 |
| N/A  31C    P0    50W / 225W |     0MiB /  4743MiB |     94%     Default |
+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name           Usage     |
|=====+=====+=====+=====
| No running processes found               |
+-----+
Charm++: standalone mode (not using charmrun)
Charm++> Running in Multicore mode: 2 threads
Charm++> Using recursive bisection (scheme 3) for topology aware partitions
Converse/Charm++ Commit ID: v6.8.2-0-g26d4bd8-namd-charm-6.8.2-build-2018-Jan-11-30463
Warning> Randomization of virtual memory (ASLR) is turned on in the kernel, thread_
↳ migration may not work! Run 'echo 0 > /proc/sys/kernel/randomize_va_space' as root to_
↳ disable it, or try running with '+isomalloc_sync'.
CharmLB> Load balancer assumes all CPUs are same.
Charm++> cpu affinity enabled.
Charm++> Running on 1 unique compute nodes (16-way SMP).
Charm++> cpu topology info is gathered in 0.001 seconds.
Info: Built with CUDA version 9010
Did not find +devices i,j,k,... argument, using all
Pe 1 physical rank 1 binding to CUDA device 1 on salg0001.hpc.wvu.edu: 'Tesla K20m' ↳
↳ Mem: 4743MB Rev: 3.5 PCI: 0:24:0
Pe 0 physical rank 0 binding to CUDA device 0 on salg0001.hpc.wvu.edu: 'Tesla K20m' ↳
↳ Mem: 4743MB Rev: 3.5 PCI: 0:8:0
Info: NAMD 2.13 for Linux-x86_64-multicore-CUDA
Info:
...
...
...
ENERGY:    499      20158.0880      19847.6155      5722.4116      178.6402      -
↳ 337057.2604      23219.0226      0.0000      0.0000      45438.8148      -222492.
↳ 6677      165.2939     -267931.4825     -222046.7421      165.2939      -2020.7666 ↳
↳ -2376.2036     921491.4634     -2020.7666     -2376.2036
(continues on next page)
```

(continued from previous page)

Warning: Energy evaluation is expensive, increase outputEnergies to improve performance.
 Info: Benchmark time: 2 CPUs 0.0134606 s/step 0.155794 days/ns 685.648 MB memory
 TIMING: 500 CPU: 6.86896, 0.0139479/step Wall: 6.88698, 0.013933/step, 0 hours
 remaining, 685.648438 MB of memory in use.

ETITLE:	TS	BOND	ANGLE	DIHED	IMPRP	
<u> </u> ELECT	VDW	BOUNDARY	MISC	KINETIC	TOTAL	<u> </u>
<u> </u> TEMP	POTENTIAL	TOTAL3	TEMPAVG	<u> </u> PRESSURE		<u> </u>
<u> </u> GPRESSURE	VOLUME	PRESSAVG	GPRESSAVG			
ENERGY:	500	20974.9322	19756.5540	5724.4551	179.8309	-
<u> </u> 337740.2041	23250.9041	0.0000	0.0000	45358.6460	-222494.	
<u> </u> 8818	165.0023	-267853.5278	-222060.5633	165.0023	-3197.5089	<u> </u>
<u> </u> -2425.3239	921491.4634	-3197.5089	-2425.3239			

WRITING EXTENDED SYSTEM TO OUTPUT FILE AT STEP 500
 WRITING COORDINATES TO OUTPUT FILE AT STEP 500
 The last position output (seq=-2) takes 0.002 seconds, 701.625 MB of memory in use
 WRITING VELOCITIES TO OUTPUT FILE AT STEP 500
 The last velocity output (seq=-2) takes 0.002 seconds, 701.641 MB of memory in use
=====

WallClock: 9.114665 CPUTime: 8.700677 Memory: 701.644531 MB
 [Partition 0] [Node 0] End of program

Notice the reduction in execution time by using the GPUs

CLUSTERS SPECIFICATIONS

8.1 Mountaineer Cluster

Mountaineer was the first centrally managed HPC cluster at WVU. Mountaineer was replaced by the HPC cluster Spruce Knob on Fall of 2018.

Mountaineer was built with 32 compute nodes. Each compute nodes was a dual socket computer (2 CPUs per node). Each CPU on Thorny had 6 cores. The total amount of cores is the product 32 (nodes) x 2 (CPUs per node) x 6 (cores per CPU) equal to 384 cores.

The total amount of memory per node is 48GB, considering that each node has 12 cores, the “Fair share” of memory for Mountaineer is 4GB per core. It is important to know this number because you can take optimize the use of the cluster for adjusting your job requirements in such a way that your jobs use 4GB per core. For example, if you have a job that uses 16GB on two cores, maybe you could ask for 4 cores if by doing that your memory usage will not go to 32GB, that is sometimes the case. By a smart selection of the number of cores and memory, you could maximize your job and the overall performance of the cluster will be better. All of that assuming that your code takes advantage of parallelization.

8.1.1 Hardware

Mountaineer is a 384 core cluster with 1.5 TB of RAM.

- **Hostname**
 - mountaineer.hpc.wvu.edu
- **Resource manager and system scheduler**
 - Torque v. 4.2.7
 - Moab Cluster Manager v. 7.2.7
- **Compute Resources**
 - **Nodes, CPUs and cores**
 - * 32 compute nodes
 - * Dual socket mainboards
 - * Dual six-core Intel Xeon X5650 2.67 GHz CPUs.
 - **Memory**
 - * 48 GB of RAM per node
 - **Network Specifics**

- * 10 Gb fiber connections between compute nodes
- * 1 Gb Ethernet connection from nodes to network attached storage

8.2 Spruce Knob

Spruce Knob is a 3,376 core cluster spread over 176 nodes (155 active nodes with 3090 cores on February 2022) using a shared FDR Infiniband Network. The system is a heterogeneous cluster in which there are different node types. In addition, each year a new addition is added to the cluster, which is known as a new phase. The cluster has five phase. New hardware is no longer being added to this cluster. Hardware that is still under warranty will be serviced. Hardware that is out of warranty will be removed from the cluster upon failure.

8.2.1 Overview

Resource manager and system scheduler

- Torque v. 6.1.x
- Moab Cluster Manager v. 9.1.x

Total Compute Resources

- 3,376 Cores
- 176 Compute Nodes
- 4 Management Nodes
- 18 GPUs

Shared Interconnect

- Mellanox SX6512 Infiniband Switch
- 216 ports all non-blocking
- FDR (56 Gb/s) speed
- All Fibre Connections

8.2.2 Hardware

Spruce Knob is a heterogeneous cluster with compute nodes with a variety of Intel Microarchitectures. The table below shows the differences between the 4 architectures:

Table 1: Microarchitectures present on *Spruce Knob*

Microarchitecture	Launched	Extensions
Sandy Bridge	2011	MMX, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, AVX, VT-x, VT-d, AES-NI, CLMUL, TXT
Ivy Bridge	2012	MMX, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, AVX, F16C, AES-NI, CLMUL, RDRAND, TXT, VT-x, VT-d
Haswell	2013	MMX, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, AVX, AVX2, FMA3, VT-x, VT-d, AES-NI, CLMUL, RDRAND, TXT
Broadwell	2014	MMX, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.1, SSE4.2, AVX, AVX2, TSX, FMA3, AES-NI, CLMUL, RDRAND, TXT, VT-x, VT-d

The Advanced Vector Extensions (AVX, also known as Sandy Bridge New Extensions) are extensions to the x86 instruction set architecture for microprocessors from Intel and Advanced Micro Devices (AMD) proposed by Intel in March 2008 and first supported by Intel with the Sandy Bridge processor shipping in Q1 2011. AVX provides new features, new instructions and a new coding scheme.

AVX2 (also known as Haswell New Instructions) expands most integer commands to 256 bits and introduces fused multiply–accumulate operations (FMA). They were first supported by Intel with the Haswell processor, which shipped in 2013.

The AVX and AVX2 are important for Scientific Computing because AVX uses sixteen YMM registers to perform a Single Instruction on Multiple pieces of Data. Each YMM register can hold and do simultaneous operations (math) on eight 32-bit single-precision floating point numbers or four 64-bit double-precision floating point numbers.

Notice that all nodes on Spruce support AVX extensions but not all nodes support AVX2.

Phase 0/1 Hardware

- Intel(R) Xeon(R) CPU E5-4620
- Intel(R) Xeon(R) CPU E5-2650 v2

Node Type	Description	Compute Nodes
Sandy Bridge	<ul style="list-style-type: none"> • 4 X 8 Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz • 64GB memory • 1TB HDD 	14
Ivy Bridge	<ul style="list-style-type: none"> • 2 X 8 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz • 32GB memory • 1TB HDD 	1
Ivy Bridge	<ul style="list-style-type: none"> • 2 X 8 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz • 64GB memory • 1TB HDD 	95
Ivy Bridge	<ul style="list-style-type: none"> • 2 X 8 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz • 256GB memory • 1TB HDD 	1
Ivy Bridge	<ul style="list-style-type: none"> • 2 X 8 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz • 512GB memory • 1TB HDD 	3

Phase 2 Hardware

- Intel(R) Xeon(R) CPU E5-2650 v3

Node Type	Description	Compute Nodes
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 64GB memory • 1TB HDD 	14
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 128GB memory • 1TB HDD 	3
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 512GB memory • 1TB HDD 	1

Phase 3 Hardware

- Intel(R) Xeon(R) CPU E5-2650 v3

Node Type	Description	Compute Nodes
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 64GB memory • 1TB HDD 	1
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 128GB memory • 1TB HDD 	10
Haswell	<ul style="list-style-type: none"> • 2 X 10 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz • 512GB memory • 1TB HDD 	1

Phase 4 Hardware

- Intel(R) Xeon(R) CPU E5-2650 v4

Node Type	Description	Compute Nodes
Broadwell	<ul style="list-style-type: none"> • 2 X 12 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz • 64GB memory • 1TB HDD 	3
Broadwell	<ul style="list-style-type: none"> • 2 X 12 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz • 128GB memory • 1TB HDD 	7
Broadwell	<ul style="list-style-type: none"> • 2 X 12 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz • 512GB memory • 1TB HDD 	1

8.2.3 Queues

Queue	Walltime
debug	15:00
standby	4:00:00
comm_mmem_week	168:00:00
comm_256g_mem	168:00:00
comm_mmem_day	24:00:00
comm_gpu	168:00:00
comm_smp	168:00:00
comm_large_mem	168:00:00

8.2.4 Research Team Queues

Research teams that have bought their own compute nodes have private queues that link all their compute nodes together. Only users given permission from the research team's buyer (Usually the labs PI) will have permission to directly submit jobs to these queues. While these are private queues - unused resources/compute nodes from these queues will be available to the standby queue (see below). However, per the system-wide policies, all research team's compute nodes must be available to the research team's users within 4 hours of job submission. By default, these queues are regulated by first come, first serve queuing. However, individual research teams can ask for different settings for their respective queue, and should contact the RC HPC team with these requests.

8.2.5 Standby Queue

The standby queue is for using resources from research teams queues that are not currently being used. Priority on the standby queue is set by fair share queuing. This means that user priority is assigned based on a combination of the size of the job and how much system resources the user have used during the given week, with higher priority assigned to larger jobs and/or user jobs that have used fewer system resources in the week. Further, the standby queue has a 4 hour wall time.

8.2.6 Community Node Queues

Spruce Knob has several queues that start with the word 'comm'. These queues are linked to the 51 compute nodes bought using NSF funding sources, and as such is open for Statewide Academic use, hardware/resource information can be found on the [Spruce Knob Systems page](#). These queues are separated by node type (i.e. large memory, gpu, smp) and can be used by all users. Currently, these nodes are regulated by fair share queuing. This means that user priority is assigned based on a combination of the size of the job and how much system resources the user have used during the given week, with higher priority assigned to larger jobs and/or user jobs that have used less system resources in the week. Further, all community queues have a 24 hour wall time, except for the week long medium memory queue (comm_mmem_week). comm_mmem_week allows jobs up to a week (168 hours); however, this queue class also limits the maximum number of nodes to 11, and a single user can not exceed 80 CPUs total within this queue. These restrictions are set to prevent a single user occupying a large number of the community resources for an excessively long time.

8.3 Thorny Flat Cluster

Thorny Flat is WVU Latest HPC Cluster. It was deployed in February 2019 and funded in large part by NSF Major Research Instrumentation (MRI) Grant Award #1726534. The cluster has a total of 6196 CPU cores spread over 170 nodes using a shared Intel Omnipath 100Gbps Interconnect. The system is a heterogeneous cluster in which there are different node types. In addition, each year a new addition is added to the cluster, which is known as a new phase. The cluster has five phases and currently is in phase 2.



8.3.1 Overview

(This text can be used for proposals to Grant Funding Agencies)

Thorny Flat is a general-purpose High-Performance Computing (HPC) cluster. Thorny Flat serves the HPC needs for West Virginia University (WVU) and other higher education institutions in West Virginia. It is hosted in Pittsburgh Supercomputer Center and was built thanks to NSF Major Research Instrumentation (MRI) Grant Award #1726534

Thorny Flat is a cluster of 170 compute nodes plus 4 management nodes. 101 nodes have a dual-socket motherboard with Intel(R) Xeon(R) Gold 6138 Processors for a total of 40 cores per node. The remaining 7 nodes have dual socket motherboard with Intel(R) Xeon(R) Gold 6126 Processors for a total of 24 cores per node. The total number of cores is 4208.

The 7 nodes with 24 cores also have 3 NVIDIA(R) Quadro P6000 24GB PCIe GPUs for a total of 21 GPU cards on the cluster. Memory on compute nodes range from 96GB to 768GB. The machines are interconnected using Intel(R) Omnipath(R) 100 Gbps with a blocking ratio of 5:1.

Thorny Flat scored 115 TeraFLOPS using just the 101 CPU-only compute nodes. Score was measured from the HPL Linpack benchmark.

Thorny Flat uses Torque and Moab for Resource Management and Job Scheduling. It has a variety of compilers, numerical libraries and scientific software specifically compiled and optimized for the hardware architecture.

Resource manager and system scheduler

- Torque v. 6.1.x
- Moab Cluster Manager v. 9.1.x

Total Compute Resources

- 6196 Cores
- 159 Compute Nodes (CPU only)
- 7 Compute nodes with 3 NVIDIA P6000 (21 GPU cards)
- 3 Compute nodes with 8 NVIDIA RTX6000 (24 GPU cards)
- 1 Compute node with 2 NVIDIA A100
- 4 Management Nodes

Shared Interconnect

- Intel Omnipath 100 Gbps
- 5:1 Blocking

8.3.2 Hardware

Phase 0/1 Hardware

- Intel® Xeon® Gold 6138 Processor
- Intel® Xeon® Gold 6126 Processor

Node Type	Description	Community Nodes	Condo Nodes	Total
Small Memory	<ul style="list-style-type: none"> • 2 x Intel® Xeon® Gold 6138 Processor (20 cores/cpu) • 96GB memory • 240GB SSD • 100 Gb Omnipath • 5 yr warranty 	64	13	77
Medium Memory	<ul style="list-style-type: none"> • 2 x Intel® Xeon® Gold 6138 Processor (20 cores/cpu) • 192GB memory • 240GB SSD • 100 Gb Omnipath • 5 yr warranty 	0	16	16
Large Memory	<ul style="list-style-type: none"> • 2 x Intel® Xeon® Gold 6138 Processor (20 cores/cpu) • 384GB memory • 240GB SSD • 100 Gb Omnipath • 5 yr warranty 	0	4	4
XL Memory	<ul style="list-style-type: none"> • 2 x Intel® Xeon® Gold 6138 Processor (20 cores/cpu) • 768GB memory • 240GB SSD • 100 Gb Omnipath • 5 yr warranty 	3	1	4
8GPU Thorny Flat Cluster	<ul style="list-style-type: none"> • 2 x Intel® Xeon® Gold 6126 Processor (12 cores/cpu) • 384GB memory • 240GB SSD • 100 Gb Omnipath • 5 yr warranty 	6	1	7
				359

8.3.3 Queues

The current state and limits of queues can be found using the qstat command.

server: trcis002.hpc.wvu.edu									
Queue	Memory	CPU	Time	Walltime	Node	Run	Que	Lm	State
standby	--	--		04:00:00	--	0	0	--	E R
comm_small_week	--	--		168:00:0	--	0	0	--	E R
comm_small_day	--	--		24:00:00	--	0	0	--	E R
comm_gpu_week	--	--		168:00:0	--	0	0	--	E R
comm_xl_week	--	--		168:00:0	--	0	0	--	E R
							0	0	

There are three main queue types - research team queues, the standby queue, and community node queues.

8.3.4 Research Team Queues

Research teams that have bought their own compute nodes have private queues that link all their compute nodes together. Only users given permission from the research team's buyer (Usually the labs PI) will have permission to directly submit jobs to these queues. While these are private queues - unused resources/compute nodes from these queues will be available to the standby queue (see below). However, per the system-wide policies, all research team's compute nodes must be available to the research team's users within 4 hours of job submission. By default, these queues are regulated by first come, first serve queuing. However, individual research teams can ask for different settings for their respective queue, and should [contact the RC HPC team](#) with these requests.

8.3.5 Standby Queue

The standby queue is for using resources from research teams queues that are not currently being used. Priority on the standby queue is set by fair share queuing. This means that user priority is assigned based on a combination of the size of the job and how much system resources the user have used during the given week, with higher priority assigned to larger jobs and/or user jobs that have used fewer system resources in the week. Further, the standby queue has a 4 hour wall time.

8.3.6 Community Node Queues

Thorny Flat has several queues that start with the word 'comm'. These queues are linked to the 73 compute/GPU nodes bought using NSF funding sources, and as such is open for Statewide Higher Education use, hardware/resource information can be found on the [Thorny Flat Systems page](#). These queues are separated by node type (i.e. extra large memory, and gpu) and can be used by all users. Currently, these nodes are regulated by fair share queuing. This means that user priority is assigned based on a combination of the size of the job and how much system resources the user have used during the given week, with higher priority assigned to larger jobs and/or user jobs that have used less system resources in the week. Further, all community queues have a week wall time, except for the (comm_small_day). comm_small_day allows jobs up 24 hours; and, this queue class has access to a larger number of resources than than comm_small_week). These restrictions are set to prevent a single user occupying a large number of the community resources for an excessively long time.

8.4 Go First Data-Analytics Cluster

GoFirst Cluster is a dedicated WVU MS Business Data Analytics computing resource that allows students in the Business Data Analytics M.S. program to gain experience in a controlled, secure cloud-computing environment. GoFirst is built from four compute nodes running HDFS shared filesystem, to run Hadoop and Spark jobs using RStudio as a front-end interface.

8.4.1 Hardware

Log in Node

- Dual CPU E5-2650 v4 @ 2.20GHz
- 24 Cores Total
- 256GB Memory
- 500 GB Raw Space

Name Node

- Dual Intel 2620 v3 @ 2.4 GHz
- 12 Cores Total
- 64 GB Memory
- 3.4 TB Raw Space

3 Worker Nodes

- Dual Intel 2650 v3 @ 2.3 GHz
- 20 Cores Each (60 Total)
- 128 GB RAM Each (384 GB Total)
- 21 TB Combined Worker Node Storage (7 TB usable)

Additional Features

- 10 Gbps Ethernet Interconnect
- Shares HPC resources (i.e. additional storage).

8.4.2 Usage

Users connect to Citrix through MasterApps.wvu.edu. In Citrix the user should have IE and Firefox browsers they can launch. After launching one of them they can go to gbef4001.hpc.wvu.edu to access RStudio Server and work within RStudio. Jobs can be launched through Hadoop or Spark 2.

REFERENCES

9.1 Common Unix commands

The Research Computing HPC server's use Red Hat Enterprise Linux (RHEL) as the operating system. Many clusters around the world run exclusively Unix/Linux based operating systems. We strongly encourage users to actively get familiar with Unix command line interface and GNU/Linux in particular. Outstanding and Free documentation is provided at [<http://www.tldp.org> The Linux Documentation Project], specifically their [<http://www.tldp.org/guides.html> Guides] which cover basic topics including Using Linux and shell scripting to advanced File system and kernel modularization guides. Below is a succinct list of Unix commands that will help you get started in moving around and manipulating files.

9.1.1 Moving around the file system

pwd List current directory

ls List contents of current directory

ls -l List contents of current directory with more information per file including permissions, last edited time, and size of file

ls -lh Same as ls -lh except file size is included in ‘human readable’ form (gigabytes, megabytes, kilobytes)

ls -lt Same as ls -l, except list files in chronological order with newer files occurring at the top

cd dirname Changes current directory to dirname

cd .. Changes current directory up one hierarchy level

9.1.2 Examining Files

cat <filename> Concatenates *filename* and prints to standard output (screen)

less <filename> A filter that pages through *filename* one full screen at a time. Allows both forward and backward movement through file

more <filename> Similar to less, except cannot move backwards through file

9.1.3 Manipulating Files and Directories

cp <filename1> <filename2> Copies *filename1* to *filename2*. If *filename2* is the name of a directory, copies *filename1* into the directory

cp -i <filename1> <filename2> Copies *filename1* to *filename2* and ask permission before overwriting

cp -r <directory1> <directory2> Copies *directory1* and all of it's contents to *directory2*

mv <filename1> <filename2> Renames “*filename1*” to “*filename2*”. If “*filename2*” is a directory, moves “*filename1*” into directory

mv -i <filename1> <filename2> Renames “*filename1*” to “*filename2*” and ask permission before overwriting files

rm <filename> Removes file

rm -i <filename> Removes file and ask permission before doing so

rm -r <directory> Removes directory and it's contents

rm -ir <directory> Removes directory and it's contents asking permission for each file

mkdir <directory> Create a directory with “*directory*” as a name

rmdir <directory> Remove an empty directory

9.2 Linux Commands

The Research Computing HPC server's use Red Hat Enterprise Linux (RHEL) as the operating system. Many clusters around the world run exclusively Unix/Linux based operating systems. We strongly encourage users to actively get familiar with Unix command line interface and GNU/Linux in particular. Outstanding and Free documentation is provided at [The Linux Documentation Project](#), specifically their [Guides](#) which cover basic topics including Using Linux and shell scripting to advanced File system and kernel modulization guides. Below is a succinct list of Unix commands that will help you get started in moving around and manipulating files.

9.2.1 Moving around the file system

pwd	List current directory
ls	List contents of current directory
ls -l	List contents of current directory with more information per file including permissions, last edited time, and size of file
ls -lh	Same as ls -lh except file size in included in ‘human readable’ form (gigabytes, megabytes, kilobytes)
ls -lt	Same as ls -l, except list files in chronological order with newer files occurring at the top
cd *dirname*	Changes current directory to <i>dirname</i>
cd ..	Changes current directory up one hierarchy level

9.2.2 Examining Files

<code>cat *file-name*</code>	Concatenates <i>filename</i> and prints to standard output (screen)
<code>less *file-name*</code>	A filter that pages through <i>filename</i> one screenful at a time. Allows both forward and backward movement through file
<code>more *file-name*</code>	Similar to less, except cannot move backwards through file

9.2.3 Manipulating Files and Directories

<code>cp *filename1 file-name2*</code>	Copies <i>filename1</i> to <i>filename2</i> . If <i>filename2</i> is the name of a directory, copies <i>filename1</i> into the directory
<code>cp -i *filename1 file-name2*</code>	Copies <i>filename1</i> to <i>filename2</i> and ask permission before overwriting
<code>cp -r *directory1 directory2*</code>	Copies “ <i>directory1</i> and all of its contents to <i>directory2</i> ”
<code>mv *filename1 file-name2*</code>	Renames <i>filename1</i> to <i>filename2</i> . If <i>filename2</i> is a directory, moves <i>filename1</i> into directory
<code>mv -i *filename1 file-name2*</code>	Renames <i>filename1</i> to <i>filename2</i> and ask permission before overwriting files
<code>rm *filename*</code>	Removes file
<code>rm -i *filename*</code>	Removes file and ask permission before doing so
<code>rm -r *directory*</code>	Removes directory and its contents
<code>rm -ir *directory*</code>	Removes directory and its contents asking permission for each file
<code>mkdir *directory*</code>	Create a directory with <i>directory</i> as a name
<code>rmdir *directory*</code>	Remove an empty directory

9.3 Software Centrally Managed

On Thorny Flat there packages centrally managed from three different sources. Packages provided via Environment Modules, Software included in Conda Environments and Software built inside Singularity Containers. The Software below is a catalog of the software offered on Thorny Flat on June 6, 2020. New packages are added over time and some old packages are deprecated and eventually removed.

9.3.1 Software via Environment Modules

Software Name	TIER 0	Description	Modules
HPL	Benchmarks	High Performance Lin-pack Benchmarks	benchmarks/hpl/2.3_gcc48 benchmarks/hpl/2.3_gcc82

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
CMake	Developers	Build System	dev/cmake/3.15.2 dev/cmake/3.15.4
Doxxygen	Developers	Create documentation from Sources	dev/doxygen/1.8.15
GNU Compiler Collection	Languages	Compilers for C, C++, Fortran and more	lang/gcc/7.5.0 lang/gcc/8.2.0 lang/gcc/8.4.0 lang/gcc/9.3.0
Go Language	Languages	Go Programming Language	lang/go/1.12.7
Intel Compiler Suite	Languages	Intel Compilers Sute: C, C++ and Fortran Includes the Math Kernel Library (MKL) and other Intel optimized Libraries	lang/intel/2018 lang/intel/2018_u4 lang/intel/2019 lang/intel/2019_u5
Java Developers Kit	Languages	Software to compile and run Java programs	lang/java/jdk1.8.0_201
Julia	Languages	Fast Interpreted Language for Scientific Computing	lang/julia/1.1.1 lang/julia/1.2.0
PGI Compiler Suite	Languages	PGI Compilers for C, C++ and Fortran	lang/pgi/19.4

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
Python	Languages	<p>Python Programming Language CPython is the reference C implementation Intel Python offer several optimizations for Numerical operations via MKL</p> <p>Pypy is an optimized implementation of Python</p>	lang/python/cpython_3.6.9_gcc82 lang/python/cpython_3.7.2_gcc82 lang/python/cpython_3.7.4_gcc82 lang/python/intelpython_2.7.14 lang/python/intelpython_2.7.16 lang/python/intelpython_3.6.3 lang/python/intelpython_3.6.9 lang/python/pypy2.7-7.1.1-portable lang/python/pypy3.6-7.1.1-portable lang/python/pypy3.6-v7.1.1-thorny
R Language	Languages	R is a Language used for Statistics	lang/r/3.5.2 lang/r/3.6.2
AtomPAW	Libraries	Projected Augmented Wave code Used by several DFT codes	libs/atompaw/4.1.0.5_gcc48 libs/atompaw/4.1.0.5_intel18
Boost	Libraries	Portable C++ source Libraries	libs/boost/1.70_gcc48_omp216 libs/boost/1.70_gcc82_omp216 libs/boost/1.70_intel18
CFITSIO	Libraries	FITS subroutine Library	libs/cfitsio/3.47_gcc82

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
FFTW	Libraries	Fast Fourier Transforms	libs/fftw/3.3.8_gcc48 libs/fftw/3.3.8_gcc75 libs/fftw/3.3.8_gcc75_ompi3.1.6 libs/fftw/3.3.8_gcc82 libs/fftw/3.3.8_gcc82b libs/fftw/3.3.8_gcc82_ompi4 libs/fftw/3.3.8_gcc84 libs/fftw/3.3.8_gcc84_ompi3.1.6 libs/fftw/3.3.8_gcc93 libs/fftw/3.3.8_gcc93_ompi3.1.6 libs/fftw/3.3.8_intel18
HDF5	Libraries	Open Source File Format for large and complex Data	libs/hdf5/1.10.5_gcc48 libs/hdf5/1.10.5_gcc48_ompi31 libs/hdf5/1.10.5_gcc82 libs/hdf5/1.10.5_gcc82_ompi31 libs/hdf5/1.10.5_intel18 libs/hdf5/1.10.5_intel18_impi18 libs/hdf5/1.10.5_intel19 libs/hdf5/1.10.5_intel19_impi19 libs/hdf5/1.10.6_gcc82_ompi31 libs/hdf5/1.12.0_gcc75 libs/hdf5/1.12.0_gcc75_ompi31 libs/hdf5/1.12.0_gcc84 libs/hdf5/1.12.0_gcc84_ompi31 libs/hdf5/1.12.0_gcc93 libs/hdf5/1.12.0_gcc93_ompi31
LibPSML	Libraries	Pseudopotential Markup Language Used by DFT codes	libs/libpsml/1.1.7_gcc82

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
LibXC	Libraries	Library for Exchange Correlation Functionals Used by DFT codes	libs/libxc/3.0.1_gcc48 libs/libxc/3.0.1_gcc82 libs/libxc/3.0.1_intel18 libs/libxc/4.2.3_intel18 libs/libxc/4.3.4_gcc82 libs/libxc/4.3.4_intel18
Magma	Libraries	Dense Linear Algebra for hybrid Multicore+GPUs	libs/magma/2.5.1_gcc48
NetCDF	Libraries	Open Source File Format for Scientific Data	libs/netcdf/4.1.1_gcc48 libs/netcdf/4.7.1_gcc82 libs/netcdf/4.7.1_intel18 libs/netcdf/4.7.1_intel19 libs/netcdf/4.x_gcc48 libs/netcdf/4.x_gcc48_ompip2 libs/netcdf/4.x_gcc82 libs/netcdf/4.x_gcc82_ompip4 libs/netcdf/4.x_intel18 libs/netcdf/4.x_intel18_impi18 libs/netcdf/fortran-4.5.2_intel18
BLAS/LAPACK (Netlib)	Libraries	Standard Linear Algebra Routines	libs/netlib/3.8.0_gcc82 libs/netlib/3.8.0_intel18
OpenBLAS	Libraries	Optimized BLAS/LAPACK with multithreading and CPU extensions like AVX	libs/openblas/0.3.5_gcc48 libs/openblas/0.3.5_gcc82 libs/openblas/0.3.7_gcc82 libs/openblas/0.3.9_gcc75 libs/openblas/0.3.9_gcc84 libs/openblas/0.3.9_gcc93
BLAS/LAPACK (Reference)	Libraries	Reference Libraries BLAS/LAPACK	libs/refblas/3.8_gcc82

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
Suitesparse	Libraries	Routines for Sparse Linear Algebra operations	libs/suitesparse/5.4.0_gcc82
Swig	Libraries	Building Scripting Language interfaces to C and C++	libs/swig/4.0.1_gcc82
XMLF90	Libraries	Support for XML in Fortran 90	libs/xmlf90/1.5.4_gcc48 libs/xmlf90/1.5.4_gcc82
YaML	Libraries	Yet Another Markup Language	libs/yaml/0.2.2_gcc82
ZeroMQ	Libraries		libs/zeromq/4.3.1_gcc82
Cuda	Parallel	CUDA compiler and runtime	parallel/cuda/10.0.130
HWLOC	Parallel	Hardware Location library	parallel/hwloc/1.10.1_gcc48 parallel/hwloc/1.10.1_gcc82 parallel/hwloc/1.10.1_intel18 parallel/hwloc/1.11.13_gcc82 parallel/hwloc/2.0.3_gcc82 parallel/hwloc/2.0.3_intel18
Intel MPI	Parallel	Intel MPI library 2017	parallel/impi/2017
MPICH	Parallel	MPICH Implementation of MPI	parallel/mpich/3.3_gcc82
MVAPICH	Parallel	MVAPICH Implementation of MPI	parallel/mvapich2/2.3.1_gcc82

continues on next page

Table 1 – continued from previous page

Software Name	TIER 0	Description	Modules
OpenMPI	Parallel	OpenMPI Implementation of MPI	parallel/openmpi/2.1.2_gcc48 parallel/openmpi/2.1.6_gcc48 parallel/openmpi/2.1.6_gcc82 parallel/openmpi/2.1.6_intel18 parallel/openmpi/3.1.4_gcc48 parallel/openmpi/3.1.4_gcc82 parallel/openmpi/3.1.4_intel18 parallel/openmpi/3.1.6_gcc75 parallel/openmpi/3.1.6_gcc84 parallel/openmpi/3.1.6_gcc93
UCX	Parallel	UCX library communication framework	parallel/ucx/1.5.0_gcc82
TMUX	Utils	Terminal Multiplexer	utils/tmux/3.0a

Software Name	TIER 1	Description	Modules
Conda		Package and Environment Management	conda
Matlab		Interactive Scientific Computing	matlab/2018b
Singularity		Containerization for HPC Clusters	singularity/2.5.2

Software Name	TIER 2	Description	Modules
ANSYS Fluids	ANSYS	Computational Fluid Dynamics	ansys/fluids_19.2
CASA	Astronomy	Common Astronomy Software Applications	astronomy/casa/5.3.0 astronomy/casa/5.4.1 astronomy/casa/5.6.0
ABINIT	Atomistic	Plane-wave based DFT code	atomistic/abinit/8.10.2_intel18 atomistic/abinit/8.10.3_gcc82 atomistic/abinit/8.10.3_gcc82_mpiio atomistic/abinit/8.10.3_intel18 atomistic/abinit/9.0.4_gcc82
AMBER	Atomistic	Molecular Dynamics	atomistic/amber/18_cuda atomistic/amber/18_mpi atomistic/amber/18_openmp
ELK	Atomistic	All electron DFT	atomistic/elk/5.2.14_intel18
Espresso	Atomistic	Plane-wave based DFT code	atomistic/espresso/6.4_intel18_seq atomistic/espresso/6.4_intel18_thd
Gaussian	Atomistic	Localized basis DFT code	atomistic/gaussian/g16 atomistic/gaussian/g16_rev1 atomistic/gromacs/2016.6 atomistic/gromacs/2016.6_cuda atomistic/gromacs/2016.6_gcc48_cuda atomistic/gromacs/2016.6_gcc82 atomistic/gromacs/2016.6_plumed_gcc82
372			Chapter 9. References atomistic/gromacs/2018.8_gcc82 atomistic/gromacs/2018.8_plumed_gcc82

Software Name	TIER 3	Description	Modules
Metamodules		GCC + Python + Julia ... Intel Compilers + Python Jupyter Notebooks R Version 3.5.2 R Version 3.6.2	general_gcc82 general_intel18 jupyter_kernels r/3.5.2 r/3.6.2

9.3.2 Software via Conda Environment

Conda Environment Name	Description
base	Base conda environment, the first environment when you activate conda
genomics-core-2020b	Set of packages for Bioinformatics
neural_gpu	Neural Networks and Deep Learning with GPUs
python27	Environment with Python Scientific Packages using Python 2.7
python35	Environment with Python Scientific Packages using Python 3.5
python36	Environment with Python Scientific Packages using Python 3.6
python37	Environment with Python Scientific Packages using Python 3.7
qiime2-2020.2	QIIME2 Next-Generation Microbiome Bioinformatics

9.3.3 Software via Singularity Images

Image Name	Description
casa-5.6.2_centos6.simg	CASA build on CentOS 6
centos-final.simg	Example of image using libgraph 1.0.2, run examples “circle”, “julia” and “sample”
dakota-6.10-release-public-rhel7.simg	Dakota 6.10: Sandia software for optimization and Uncertainty Quantification
docs_hpc_wvu.simg	Contains Sphinx, pandoc and scripts to create documentation for https://docs.hpc.wvu.edu
dolmades.simg	Windows Apps under Linux using Singularity (using wine)
Grass-7.4.0.simg	GRASS (Geographic Resources Analysis Support System), open source GIS
Grass-7.4.simg	GRASS (Geographic Resources Analysis Support System), open source GIS
Grass-7.6.1.simg	GRASS (Geographic Resources Analysis Support System), open source GIS
Jupyter-5.2.2_Python-3.6.8.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks
jupyter_conda.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks
jupyter.simg	Python 3.6.8 with a number of Scientific Packages and Jupyter Notebooks
jupyter-xenial.simg	Python 3.5.2 with a number of Scientific Packages and Jupyter Notebooks
Keras-2.1.4_TensorFlow-1.5.0.simg	Neural Networks and Deep Learning with Keras 2.1.4 and TensorFlow 1.5.0
loos.simg	Lightweight Object-Oriented Structure library (LOOS)
maxwell-b.simg	Maxwell is a Multi-GPU FDFD solver
miniconda3_firefox.simg	Jupyter from miniconda with Firefox
miniconda3.simg	Jupyter from miniconda without firefox
orp-2.2.6.simg	Oyster River Protocol
pycbc-el7.simg	PyCBC Software for Study Gravitational Waves
ParaView-5.6.0.simg	ParaView 5.6: open-source, multi-platform data analysis and visualization
ParaView-5.7.0.simg	ParaView 5.7: open-source, multi-platform data analysis and visualization
QGIS.simg	QGIS Software for Geographic Information Systems

Table 2 – continued from previous page

Image Name	Description	
RStudio-desktop-1.2.5033_R-3.4.4.simg	RStudio Desktop 1.2 with R 3.4.4	
RStudio-server-1.2.5033_R-3.6.2.simg	RStudio Server 1.2 with R 3.6.2	
Stacks-2.5.simg	Stacks: Pipeline for building loci from short-read sequences like illumina	
Tensorflow-1.13.1-gpu-py3-jupyter.simg	TensorFlow with support for GPUs	tensorflow/tensorflow:1.13.1-gpu
Tensorflow-1.13.1-py3-jupyter.simg	TensorFlow without support for GPUs	tensorflow/tensorflow:1.13.1-py3
TensorFlow_gpu_py3.simg	TensorFlow with support for GPUs	tensorflow/tensorflow:latest-gpu
Ubuntu.simg	Ubuntu Xenial	
Visit-2.13.2.simg	Visit: Interactive, scalable, visualization, animation and analysis tool	
Visit-2.13.3.simg	Visit: Interactive, scalable, visualization, animation and analysis tool	
Visit-3.1.xenial.simg	Visit: Interactive, scalable, visualization, animation and analysis tool	
wkhtmltox-0.12.simg	wkhtmltopdf command line tools to render HTML into PDF and other image formats	
wkhtmltox.simg	wkhtmltopdf command line tools to render HTML into PDF and other image formats	
xfemm_v1_9_linux64.simg	Cross platform electromagnetics finite element analysis based on FEMM	
xfemm_v1_9_linux64.stretch.simg	Cross platform electromagnetics finite element analysis based on FEMM	

**CHAPTER
TEN**

INDICES AND TABLES

- genindex
- modindex
- search