lux.ucsc.edu

Release v0.1

Apr 25, 2022

Contents:

CHAPTER 1

About

The NSF-funded *lux* supercomputer at UC Santa Cruz will enable computational research in astrophysics in galaxy formation, cosmological structure formation, exoplanet formation and atmospheres, astrophysical transients and gravitational wave sources, and convective and turbulent processes in stellar interiors. Broad computational research at UCSC in planetary science, applied mathematics, chemistry, and materials science will also be performed on the system.

Submit comments or corrections to this documentation by submitting an issue on GitHub.

1.1 Getting Started

1.1.1 Read the User Policies

First things first – please read the *lux* Policies. All *lux* users must understand and abide by the system policies as a condition of using the computer. This includes acknowledging lux in publications when used and following the user best practices.

1.1.2 Getting an account on *lux.ucsc.edu*

There are several ways to get an account on *lux*, described below, including special cases under which accounts may be granted. If you have questions about account access, please feel free to contact Brant Robertson.

Accounts for Co-I research group members

If you are a UCSC faculty, postdoc, or student with a CruzID and part of a Co-I research group (see Project Team Members), your associated faculty member can directly grant you access to *lux*. Please contact the faculty member and ask them to add you to their group. Accounts added in this way will inherit the privileges of the faculty research group on *lux* and use their assigned system resources, and the faculty lead determines the duration and any special conditions of the access.

Accounts for general access by UCSC students, staff, and faculty

With prior approval and agreement, access to *lux* may be granted to UCSC students, staff, and faculty who are not associated with Co-I research groups. Please contact Brant Robertson for more information.

Accounts for UCSC courses

By prior arrangement, temporary *lux* access may be granted to students in UCSC courses during the quarter of instruction. If you are interested, please contact Brant Robertson for more information.

Accounts for Co-I research group external collaborators

Co-Is may grant external (outside UCSC) collaborators access by requesting a UCSC Sundry account (see instructions here or submit a Sundry account request), which will provide a limited time but renewable CruzID for their collaborator. *When requesting a Sundry account please request that the account receive the Gold Active Directory (AU) resource to enable access to the lux cluster.* Once a CruzID is assigned and a Gold password set, the faculty lead can add their collaborator to their group directly. Please be mindful that *lux* was funded to support research at UCSC, and these external collaborators gain the same privileges as other UCSC Co-I research group members. External collaborators still must agree to acknowledge *lux* in their resulting publications. Only Co-I faculty leads can grant external collaborators access to *lux*.

Accounts for UCSC visitors

Visitors to UCSC who are associated with Co-I research groups may be able to obtain accounts following the instructions for *external collaborators*. Groups of visitors may also be granted access under special circumstances, please contact Brant Robertson for more information. All visiting users must agree to acknowledge *lux* in any resulting publications.

1.1.3 Connecting to *lux*

Here are step-by-step instructions for connecting to *lux*:

1) Request an authorized account, following the above *instructions*.
2) Connect to the UCSC Campus VPN using your CruzID and Gold password via the UCSC VPN client. Installation instructions for the VPN client are available on the ITS website. If you have not reset your Gold password recently, you may need to do so in order to connect to the system (visit cruzid.ucsc.edu). You will need to enroll a device in Duo for Multifactor Authentication to connect to the VPN.
3) Connect to the *lux* system via Secure Shell:

   ```
   $ ssh [cruzid]@lux.ucsc.edu
   ```

   Where [cruzid] is your CruzID. When prompted, provide your Gold password and select a Duo MFA method. You will be directed to one of the three login nodes (*lux-0*, *lux-1*, or *lux-2*) where you can submit jobs to the system queues.
4) See Using lux for further instructions.

1.2 *lux* System Configuration

1.2.1 Compute System Overview

The *lux* system is based on 3 login servers (*lux-0*, *lux-1*, and *lux-2*) and 108 Dell compute nodes. The compute nodes each have 2x 20-core Intel Xeon Gold 6248 (Cascade Lake) CPUs, 192GB RAM, 2.4TB SSD local scratch space, and 100 GB/s Mellanox HDR non-blocking Infiniband networking. There are 80 CPU-only nodes (*node001-node080*) and 28 GPU-enabled nodes (*gpu001-gpu028*) that have 2x NVIDIA 32GB V100 GPUs.

Instructions for getting access and using the system are available on the Getting Started and Using *lux* pages.

1.2.2 Facilities Statement for Grants

You may wish to include a description of *lux* in your facilities / equipment statement for grant or national supercomputing applications:

> UC Santa Cruz hosts a new high-performance computer *lux* funded by NSF MRI grant AST 1828315, installed at the UCSC Data Center. This new Dell system features 108 nodes, each with 2x 20-core Intel Xeon Gold 6248 (Cascade Lake) CPUs, 192GB RAM, 2.4TB SSD local scratch space, and 100 GB/s Mellanox HDR non-blocking Infiniband networking. Of the compute nodes, 28 include 2x NVIDIA V100 32GB GPUs (56 total). The *lux* system also features a 3.6PB DataDirect Networks Lustre storage system Infiniband-connected directly to the compute nodes, allowing for high throughput attached storage.

1.3 Using *lux*

We aim to create and maintain a great user experience with the *lux* cluster. Below, we provide a description of how to use the system.

1.3.1 Software and Environment

Starting Out

Once you connect to *lux* (see Connecting to lux), you will be placed on one of three login nodes (*lux-0*, *lux-1*, *lux-2*). Connections are cyclically assigned across these nodes to distribute the user load, but for most purposes the nodes will function identically for the user.

The user home directories are mounted at `/home` across the system, as are the `/data` directories. These global filesystems are imported from the DDN storage server via the HDR Infiniband network fabric, and support fast I/O from anywhere on the cluster. See *File System* below

Software Stack

The primary software stack for *lux* currently consists of

- `gcc 4.8.5` for GNU compilers, including C, C++, and Fortran.
- `icc 19.0.5.281` for Intel compilers, including C, C++, Fortran, and Intel MPI.
- `openmpi` for OpenMPI parallel compilers, based on `gcc 4.8.5`.
- `python 3.6.7` for Python and related libraries.
- `slurm 18.08.4` for queueing and job submission.

Each of these elements of the software stack are either automatically loaded by default (`gcc 4.8.5`, `python 3.6.7`, and `slurm 18.08.4`) or loadable as a `module` (`openmpi` and Intel `icc/mpii*` compilers; See *Modules* below).

Modules

To load software and update environmental variables for path, library linking, etc., the *lux* system relies on the Modules package. To load a `module`, just:

```
$ module load [module_name]
```

where `[module_name]` is an available module. To see modules available on the system, type:

```
$ module avail
```

To list all currently loaded modules, write:

```
$ module list
```

By default, only `slurm` and `python/3.6.7` are loaded, along with the metapackage `shared` that gives access to shared software modules installed in `/cm/shared`. The default is set in your `~/.bashrc` file and can be changed by altering the file. Note that `slurm` is required to run jobs on the system.

To remove a single module, simply type:

```
$ module remove [module_name]
```

To remove all currently loaded modules, write:

```
$ module purge
```

For more information, see `man module`. CHECK THAT MODULES ARE IMPORTED VIA SLURM BATCH AND INTERACTIVE JOBS

File System and Quotas

The file system for the cluster is based on a DataDirect Networks Lustre appliance, which hosts the `/home` and `/data` user directory structures and the `/cm/shared` directory that contains common software and modules.

The filesystems on *lux* are subject to storage quotas. While there is substantial storage available on the system, some members of the *lux* project team and affiliated departments and divisions have directly purchased storage in support of research projects. Some limited free storage is available to users.

Your `/home/[user name]` directory have a storage quota of *5 GB*.

Your `/data/users/[user name]` directory have a storage quota of *100 GB*.

If you belong to a research group affiliated with a *lux* Co-I, you may have access to a `/data/[research group]` directory with significant storage. While these directories have storage quotas, they should be sufficient for most research group needs. If you have questions about accessing the your research group's `/data` directories, please contact your affiliated Co-I.

d.

1.3.2 Running Jobs

The *lux* system uses Slurm workload manager to schedule and execute jobs on the cluster nodes. If you are not familiar with Slurm don't worry, Slurm works like PBS, LSF, and other schedulers you may have used in the past. Below we provide information for running batch and interactive jobs on the *lux* nodes via Slurm.

Slurm Cheat Sheet

There is a great slurm command cheat sheet available!

Queues

The *lux* system has a variety of job queues (called *partitions* in Slurm parlance) available for users. The currently available queues include:

- **windfall** The default queue, submits to *node[001-080]* and *gpu[001-028]*. The windfall queue has no per user or time limitations, but can be pre-empted by all other queues.
- **defq** The CPU-only queue, submits to *node[001-080]*, available to lux collaboration members. Will pre-empt windfall, limited to 24 hours / 16 nodes.
- **gpuq** The GPU-enabled node queue, submits to *gpu[001-028]*, available to lux collaboration members. Will pre-empt windfall, limited to 24 hours / 16 nodes.

There are also queues for individual research groups with a limited number of nodes available at the highest priority, pre-empting jobs in the windfall, defq, and gpuq paritions.

In the following, please substitute one of these queues when instructed to specify the `[queue name]`.

To get information on the status of the queues, use the `sinfo` command:

```
$ sinfo

PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
windfall*    up   infinite     80  idle node[001-080]
    windfall*    up   infinite     28  idle gpu[001-028]
```

This shows the queue names (PARTITION), their availability (AVAIL), any time limit (TIMELIMIT), the state of nodes (STATE), the number of nodes in that state (NODES), and the list of nodes in that state (NODELIST).

To see the list of jobs in the queues, use the `squeue` command:

```
$ squeue
```

This shows the id of each job (JOBID), the queue the job is assigned to (PARTITION), the name of the job (NAME), the job owner (USER), the job state (ST), the runtime of the job (TIME), the number of nodes used or requested by the job (NODES), and the list of nodes assigned to the job or the reason the job is still queued [NODELIST/(REASON)].

Detailed information about the queues can be retrieved using `scontrol show partition`:

```
$ scontrol show partition
```

To cancel a job:

```
$ scancel [JOBID]
```

where [JOBID] is the job you wish to cancel.

Batch Job Submission

To run a batch job across multiple nodes, from a login node execute the following command:

```
$ sbatch --partition=[queue name] --account=[queue name] [script name]
```

Substitute the name of the queue you wish to use for `[queue name]`. This will submit a slurm batch script file `[script name]` to the specified queue. Both :file:'–partition=[queue name]' and :file:'–account=[queue name]' must be specified.

Example Batch Script for OpenMPI

We provide below an example slurm batch script, which executes an mpi job with 80 mpi processes distributed across 2 nodes, with 40 mpi processes per node (e.g., one per core) to the :file:cpu queue

```
#!/bin/bash
#SBATCH --job-name=mpi_job_test      # Job name
#SBATCH --partition=cpuq             # queue for job submission
#SBATCH --account=cpuq               # queue for job submission
#SBATCH --mail-type=END,FAIL         # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=[USER ID]@ucsc.edu   # Where to send mail
#SBATCH --ntasks=80                  # Number of MPI ranks
#SBATCH --nodes=2                    # Number of nodes
#SBATCH --ntasks-per-node=40         # How many tasks on each node
#SBATCH --time=00:05:00              # Time limit hrs:min:sec
#SBATCH --output=mpi_test_%j.log     # Standard output and error log

pwd; hostname; date

echo "Running program on $SLURM_JOB_NUM_NODES nodes with $SLURM_NTASKS total tasks,␣
↪with each node getting $SLURM_NTASKS_PER_NODE running on cores."

module load openmpi

mpirun -N 2 --map-by ppr:40:node ./mpi_test

date
```

This example can be submitted to the queues following the instructions in *Batch Job Submission* above.

Example Batch Script for Intel MPI

We provide below an example slurm batch script, which executes an mpi job with 80 mpi processes distributed across 2 nodes, with 40 mpi processes per node (e.g., one per core):

```
#!/bin/bash
#SBATCH --job-name=mpi_job_test      # Job name
#SBATCH --partition=cpuq             # queue for job submission
#SBATCH --account=cpuq               # queue for job submission
#SBATCH --mail-type=END,FAIL         # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=[USER ID]@ucsc.edu   # Where to send mail
#SBATCH --ntasks=80                  # Number of MPI ranks
#SBATCH --nodes=2                    # Number of nodes
#SBATCH --ntasks-per-node=40         # How many tasks on each node
#SBATCH --time=00:05:00              # Time limit hrs:min:sec
```

(continues on next page)

(continued from previous page)

```
#SBATCH --output=mpi_test_%j.log     # Standard output and error log

pwd; hostname; date

echo "Running program on $SLURM_JOB_NUM_NODES nodes with $SLURM_NTASKS total tasks,␣
↪with each node getting $SLURM_NTASKS_PER_NODE running on cores."

module load intel/impi

mpirun -n 80 --ppn 40 ./mpi_test

date
```

This example can be submitted to the queues following the instructions in *Batch Job Submission* above.

Job Arrays

To submit a job array, use the `--array=[range]` option (examples taken from the slurm website):

```
# Submit a job array with index values between 0 and 31
$ sbatch --array=0-31    -N1 tmp

# Submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 -N1 tmp

# Submit a job array with index values between 1 and 7
# with a step size of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2   -N1 tmp
```

Interactive Sessions

To create an interactive session on a compute node, from a login node execute the following command:

```
$ srun -N [Num of nodes] --partition=[queue name] --account=[queue name]  --pty bash -
↪i
```

Substitute the name of the queue you wish to use for `[queue name]`. This will create a `bash` shell in an interactive session on [Num of nodes] nodes (`-N [Num of nodes]`).

Here is an example of combining srun + mpirun to run 3610 mpi processes interactively on 79 nodes using openmpi:

```
$ srun -N 79 -n 3160 --partition=cpuq --account=cpuq --pty bash -i
$ mpirun -n 3160 --map-by ppr:40:node ./mpi_test
```

To allocate a multi-node interactive session, use the `salloc` command:

```
$ salloc -n[ncores] sh
> srun [executable]
> exit
```

This set of command allocates `[ncores]` cores on the system and starts a shell `sh`. Then `srun` command executes the job `[executable]`, and `exit` ends the session.

X Forwarding

Forwarding of X windows is enabled on the system. First, be sure you have connected to *lux* with X forwarding enabled:

```
$ ssh -Y [username]@lux.ucsc.edu
```

Second, request an interactive shell with the `--x11` flag enabled:

```
$ srun --x11 -N 1 --partition=cpuq --account=cpuq --pty bash -i
```

This should connect to a node in an interactive shell with x-forwarding enabled. Then try opening a xterm:

```
$ xterm &
```

With any luck, a few seconds later a remote xterm from the node should appear on your screen.

Jupyter Notebooks

Yes, you can use Jupyter notebooks on the nodes! (Please do not use notebooks on the login nodes, those jobs will be terminated!) Here are some step-by-step instructions.

First, connect to *lux* with x forwarding following the instructions *X Forwarding* above. Before you request a node, load the following modules:

```
$ module load python/3.6.7
$ module load jupyter
$ module load numpy matplotlib [...]
```

Afterward, request an interactive shell with x forwarding using the `srun` command with the `--x11` flag. Once you log into the node, do the following:

```
$ xterm &
$ chromium-browser &
```

Once the xterm and the browser both appear on your remote desktop, *in the xterm* execute the following command:

```
$ jupyter-notebook
```

This should redirect the browser window to a Jupyter instance. You'll have access in the Jupyter instance to any python modules you've loaded.

Tensorflow

The *lux* system has GPU-enabled TensorFlow on the `gpu` nodes. To access the `gpu` nodes in an interactive shell, just:

```
$ srun --x11 -N 1 --partition=gpuq --account=gpuq --pty bash -i
```

Then at the prompt load the required modules:

```
$ module load cuda10.0 libcudnn7/7.6 python/3.6.7 numpy/1.17.0 h5py/2.9.0 tensorflow-
↪gpu/1.14.0
```

Note the `tensorflow-gpu` module is only available on the `gpu` nodes.

The following test script should execute successfully once the modules are loaded:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

If you have any difficulty running the test MNIST classification script, please let us know.

1.3.3 Known Issues

Below, we list the known issues with the system. We are looking into them! But if you encounter unusual behavior, please have a look here first (but please also let us know about it!).

Authentication Issues

If you recieve an error about "Permission denied" when accessing your /home directory, please let us know as it's related to the authentication scheme on the servers that host the /home and /data directories. We are looking into a permanent fix.

Intel MPI Issues

Currently, Intel MPI-compiled binaries will not run on more than 20-30 cores per node. We recommend using Open-MPI until this issue is resolved. If you encounter Intel MPI or Open MPI issues, please let us know.

1.4 Information for *lux* Co-Investigators

1.4.1 Group Management

Each Co-I on the *lux* project has an access group they can manage, which allows for you to enable access to your students, postdocs, and collaborators directly.

For people with existing UCSC CruzID accounts, just follow the *Adding Users* directions below to enable their access to *lux*. No intervention from ITS or Brant is needed, and you can manage your access group directly. These accounts share your resources, disk space, group accounting, etc. Once you've enabled their access, just direct them to Getting Started and Using lux for further information.

For external collaborators without a UCSC CruzID account, you will need to request a sundry account (see more info here). The sundry accounts will grant your collaborator a CruzID, but may only be authorized for one year at a time (renewable annually). As long as the sundry account is active and in your access group, your collaborator will be able to connect to *lux*. The sundry accounts will have the same resources as others in your access group. Also, please be sensitive to the ongoing UCSC-based research on *lux* and realize that enabling access to external competitors can reduce our institutional colleagues' competitive advantage.

1.4.2 Adding Users

Here are instructions for adding users to your *lux* access group:

- Log onto cruzid.ucsc.edu using your CruzID and Gold password.
- On the blue menu, click **Advanced->Group Management**.
- This should take you to the **Group Management** page, and you should see a group called **lux-[your last name]**. Please click that link.
- You should now see **Manage Group - lux-[your last name]**. You can then click **+ Group Members** and add someone by name or CruzID.
- If you have questions or issues, please email Brant.

1.5 Policies

1.5.1 Overview

Thanks for becoming a user of the *lux* system at UCSC. Below, we highlight the user policies for the system. Anyone using the *lux* cluster assents to these policies as a condition of access to the *lux* cluster. For questions, please email Brant Robertson.

1.5.2 Acknowledgments

All users of *lux* must include the following acknowledgment in any publication resulting or benefitting from usage of the system:

> **We acknowledge use of the lux supercomputer at UC Santa Cruz, funded by NSF MRI grant AST 1828315.**

Including this funding statement is critical, and users and any related Co-I are responsible for adding this acknowledgment. Failure to include this statement in publications that used *lux* will be considered a *policy violation*.

1.5.3 Notification of Publications

Please let Brant Robertson know about any publications arising or benefitting from your access to *lux* (and please include the *acknowledgments* in the published version!).

1.5.4 Scheduler and Queues

Users are expected to submit jobs and access nodes through the provided Slurm queues.

1.5.5 Storage and Quotas

User accounts are subject to storage quotas.

1.5.6 Backups

Backups of `/home` will be performed infrequently. The `/data` filesystem will not be backed up.

1.5.7 Software and Libraries

The usage of lux is shared among many disciplines, and this breadth makes the detailed management of the system challenging. Software from many different disciplines requires a large range of libraries and packages, and the maintenance of all such libraries may be unmanageable. Widely-used libraries may be installed as system modules for use by multiple users. Research or custom-built software will need to be installed and maintained by their users.

1.5.8 Login Node Usage

The *lux* system frontend servers (*lux-0*, *lux-1*, *lux-2*) are designed to support users of the cluster in its intended usage mode (i.e., distributed jobs running on the nodes). Any computationally or memory intensive job should not be run on the frontend, and instead should be executed on a node allocated via the Slurm scheduler. This policy especially includes the execution of python scripts, including simulation post processing and plotting – these are tasks that should be run on the nodes. Any job running on the frontend can be terminated without notice to the user.

Computational tasks explicitly permitted on the frontend system are text/code editing, compilation, data transfer, and job monitoring. If you have a question about how to use the lux frontend systems, please email Brant Robertson.

1.5.9 Policy Violations

Users who violate the above policies risk losing temporary or permanent access to *lux*. The project Co-Is agree to help enforce policies for their affiliated users. If you have questions about policy violations, please email Brant Robertson.

1.6 Project Team

1.6.1 Members

Lead Principal Investigator: Brant Robertson (Astronomy & Astrophysics)

Co-Principal Investigators: Nicholas Brummell (Applied Mathematics), Nicole Feldl (Earth and Planetary Sciences), Piero Madau (Astronomy & Astrophysics), Enrico Ramirez-Ruiz (Astronomy & Astrophysics)

Co-Investigators: Ilan Benjamin (Chemistry), Ryan Foley (Astronomy & Astrophysics), Jonathan Fortney (Astronomy & Astrophysics), Pascale Garaud (Applied Mathematics), Tesla Jeltema (Physics), Alexie Leauthaud (Astronomy & Astrophysics), David Lederman (Physics), Dongwook Lee (Applied Mathematics), Douglas Lin (Astronomy & Astrophysics), Carl Maltzahn (Computer Science and Engineering), Ruth Murray-Clay (Astronomy & Astrophysics), Francis Nimmo (Earth and Planetary Sciences), Yuan Ping (Chemistry), Brad Smith (Computer Science and Engineering; Information Technology Service), Xi Zhang (Earth and Planetary Sciences)

1.6.2 Thank Yous

Without support of the following people and organizations, the lux system would not have been possible – Thank you!!

National Science Foundation

UCSC Office of the Executive Vice Chancellor, UCSC Office of the Vice Chancellor for Research, UCSC Office of the Vice Chancellor for Information Technology, UCSC Division of Physical and Biological Sciences, UCSC Jack

Baskin School of Engineering, UCSC Department of Astronomy and Astrophysics, UCSC Department of Earth and Planetary Sciences, UCSC Department of Applied Mathematics, UCSC Department of Physics, UCSC Department of Chemistry, UCSC Information Techology Services

Special thanks to our purchaser Nancy Nieblas, who worked tirelessly to enable the acquisition of the system.

Andrea Hesse, Josh Sonstroem, Alison Lindberg, Kirk Loftis, Tania Pannabecker, Chris Parker

Paul Albe, Andre Daniels, Robert Kemp, Cliff Pearson, George Peek, Kari Robertson, Jim Warner

1.6.3 In Memoriam

We acknowledge the loss of our friend and collaborator Stephen Hauskins, a long time staff member of the UCSC ITS and the ITS divisional liaison for Physical and Biological Sciences. Many of us benefitted from Stephen's' technical support for the division, including the Hummingbird computing cluster. The lux project worked closely with Stephen during its formative months, as he lended us his technical expertise enthusiastically. Stephen decommissioned the previous Hyades system, worked with the lux project on the vendor selection for the computer, provided almost daily input into the configuration and management of lux, and was to work with us on the user support going forward. His kindness, expertise, and helpfulness will be greatly missed.

1.7 *lux* Jupyter Hub

1.7.1 Jupyter Hub Overview

The *lux* system has a Jupyter Hub that enables users to run Jupyter notebooks remotely via a web interface. For more information, visit the Jupyter Hub website.

To connect to the *lux* Jupyter Hub:

1. Connect to the UCSC campus VPN.
2. Visit https://lux-0.ucsc.edu:8000.
3. Login using your UCSC Gold credentials (same as used when connecting via ssh).

Once you have connected, you will see a Launcher window that contains a variety of Python 3.7 kernels that can execute directly on the nodes via slurm. In addition to the shared *windfall*, *cpuq*, and *gpuq* partitions, each of the *lux* group partitions have kernels available. You will need to have membership in the corresponding partition to run a Jupyter notebook. Jupyter notebooks running on the *windfall* partition are limited to a single core and 4GB of memory, with the individual nodes potentially shared between multiple users. The *cpuq*, *gpuq* and group partitions have exclusive access when running Jupyter notebooks.

Once you have completed using the notebook via the kernel you select, please be sure to shut down the kernel before exiting. Otherwise, the slurm job responsible for your kernel will continue to run.

1.7.2 Python 3.7

To use Jupyter Hub, we must use Python 3.7 supplied by the Bright Cluster Management software. If you have a module that needs to be installed for use with Python 3.7, please use the *help* channel on the *lux* Slack workspace.

1.7.3 Do Not Autoload Python

If you plan to use notebooks via the *lux* Jupyter hub, please be sure that you are not auto-loading the module for python/3.8.6 within your .bashrc, .bash_profile or .profile. The notebook's python kernel depends on the cm_local module python37, and your notebook will fail if you already have python/3.8.6 loaded in user environment. You can still module load python/3.8.6 within your sbatch scripts to use the later 3.8.6 version of python in your slurm workflows.

1.7.4 Known Issues

1. The terminal under the Jupyter Hub does have access to groups associated with the University credential system. So do not submit jobs from the terminal within Jupyter.
2. If a queue is full (using all its available resources) and a user requests a node using Jupyter for that queue, the notebook will fail and then quitting the notebook will not cancel the slurm job. You can use :file:'scancel' from a termal connected directly to *lux* to cancel the job.

1.8 Recent gcc versions on Lux

Lux uses by default gcc 4.8.5 owing to the version of CentOS used by the cluster management system. Some people require a more recent version of gcc, and we provide gcc 8.1 as an alternative.

1.8.1 Loading a more recent gcc version

To load a more recent gcc version, you may use the `devtoolset` module:

    module load devtoolset-8/8.1

This module should reset the gcc in your path to v8.1.

1.9 Installing Dedalus on Lux

1.9.1 And linking it to existing modules

What is Dedalus?

Dedalus is an open-source software package that can be used to solve differential equations, like the Navier-Stokes equation, using spectral methods. It solves equations input by users as strings, and can handle initial value problems, eigenvalue problems, and boundary value problems. Due to its flexibility, active userbase, and user-friendly python interfaces, it is quickly becoming a widely-used tool in astrophysical and geophysical fluid dynamics, plasma physics, and other fields.

Why installing Dedalus warrants a little care

As explained on Dedalus' installation page, Dedalus relies on MPI, FFTW (linked to MPI), HDF5, Python3, and the following Python packages: numpy, scipy, mpi4py (linked to that same MPI), and h5py. There are several ways to install Dedalus. The easiest is to use the conda installation script, which can build all of the prerequisite software (after first installing Python using miniconda, as explained in the Dedalus installation instructions), or be linked to existing MPI/FFTW/HDF5 installations. This method is great if you're just starting out, or if you're installing Dedalus on a machine that is missing some or all of the prerequisite software packages. However, cluster/supercomputer support staff generally discourage this sort of an installation when it can be avoided, as it often fails to take advantage of various subtle optimizations that are available on each machine depending on the software environment and CPU architecture (granted, when it comes to Dedalus, these subtleties are probably most important when installing MPI and FFTW).

Additionally, if you would like to contribute to Dedalus, it helps to have Dedalus installed in an editable form, which the conda installation doesn't do without some tinkering. (If you would like to do that with the conda installation, either reach out to Adrian Fraser or ask about it on the Dedalus user group. It's pretty easy.)

If you would like to install Dedalus using the conda install script, based on past experience, it should work pretty well without any major difficulty.

Installing Dedalus on top of Lux's existing modules, from source

Note that these instructions were written in March 2021. There are several aspects that are likely to change over time. In particular, the versions of various modules installed on Lux might change. Aspects that are likely to require modification when these instructions become outdated will be noted *(like this)*.

First, load the following modules (if you haven't already, do `module list` to see if you already have some of these loaded):

```
module load slurm/18.08.4 openmpi/4.0.1 fftw/3.3.8 python/3.8.6
```

(The versions of each of these modules might change over time)

Next, set the `MPI_PATH` and `FFTW_PATH` environment variables, which tell Dedalus where on Lux this software is installed:

```
export MPI_PATH=/cm/shared/apps/openmpi/openmpi-4.0.1/
export FFTW_PATH=/cm/shared/apps/fftw/fftw-3.3.8/
```

(These paths will change if the openmpi or fftw modules change) **Note that you'll probably want to load these same openmpi and fftw modules and set these same ''MPI_PATH'' and ''FFTW_PATH'' environment variables if you're using the simpler conda install too.**

Then, fetch your Dedalus version of choice from github and place it in whatever directory you want Dedalus to live (your best bet is usually to either download the main AKA "master" branch, which is the default when you click that link, or to download the latest official release – but if you want to test out the new spherical bases/domains being developed, check out their d3 branch). Use `cd` to enter the `dedalus/` folder you just downloaded.

The first time I installed Dedalus on Lux using this method, I hit a strange error that I was not able to reproduce ever again. I document it here in case others hit this error as well, but you can skip this step if you don't hit this error. The error happened when I tried to ''pip3 install'' (explained in the next paragraph). The error was relating to write permissions in the ''/tmp'' folder and the ''dedalus/.git'' folder, and googling around I saw that some others have had similar errors somehow stemming from conflicts between pip3 and git. At the time, I deleted the ''dedalus/.git'' folder and was able to continue without any additional errors (you can re-initiate git tracking after the fact if desired). Since then, however, I've been completely unable to reproduce this error. I'm leaving this paragraph here for now in case others encounter this issue.

Once you've fetched the code and entered into the `dedalus/` directory, we need to add a modification to the file `setup.py` because for some reason `pip3` will otherwise not install Dedalus correctly, because it will ignore half of the instructions we give it. Open `setup.py`. Near the top of the file, you should see a block of `import` statements. immediately after all those `import` statements, add the following:

```
import site
site.ENABLE_USER_SITE = "--user" in sys.argv[1:]
```

(This is to circumvent an issue where ''pip3 install'' won't properly do what the ''–user'' option tells it to do if you simultaneously use the ''-e'' option, causing pip3 to install in a Lux-wide directory, which will fail unless you have root privileges. Either this is a bug with ''pip3'' that will be addressed in some later version of ''pip3'', or this is an issue with Dedalus that will be addressed in some later version of Dedalus, I'm not sure which.)

Finally, run the command `pip3 install --user -e .` from this same `dedalus/` directory, and you should be all set!

Once the installation is finished, run the command `python3 -m dedalus test` and Dedalus will run through its testing suite to make sure it's installed correctly. If everything worked out ok, you should see several yellow warnings and xerrors (expected errors), but no red (serious) errors. And you should be ready to get simulating! Make sure to load all the same modules whenever you want to use dedalus, whether in an interactive session or in an sbatch script for queued jobs.

(At the time of this writing, the latest "release" of Dedalus, which is installed when you use the conda install script, has some minor problems that, depending on OS and scipy release, may lead to errors when you do ''python3 -m dedalus test''. An example of these errors can be seen 'here <https://github.com/DedalusProject/dedalus/issues/110>'__. The errors all mention Hermite bases. If you don't plan on using Hermite bases anyway for your simulations, then you can disregard these errors. If you do plan on using Hermite bases, then this was addressed in a 'recent change <https://github.com/DedalusProject/dedalus/commit/efb13bdaa09816dde3eee897bc2a15fc284ea2f1>'__ on the main/"master" branch on github, so all you need to do is work with the most recent github version rather than the most recent "release" version.)

(Also: these instructions did not work with python3's virtual environments ''venv''. When using ''venv'', python and pip3 are no longer able to see any of the python packages installed as part of the python/3.8.6 module, which leads to a longer and more involved installation because you need to install numpy/scipy/etc locally. Because the python/3.8.6 module seems to include every single python prerequisite that Dedalus needs, I personally find virtual environments to be less important in this situation. There are certainly instances where you might want to use virtual environments anyway, depending on your specific needs/workflow.)