Center for
HIGH PERFORMANCE COMPUTING
THE UNIVERSITY OF UTAH

Fall 2021

Smoke Forecasting With a Coupled Fire-Atmosphere Model

Derek V. Mallia, Department of Atmospheric Science, University of Utah
Adam Kochanski, Department of Meteorology and Climate Science, San Jose State University

The number of large wildfires has been steadily increasing since the early 1980s, and it is suspected that wildfire smoke is responsible for deteriorating air quality across the western U.S. Wildfire intensity is projected to increase through the end of the 21st century due to climate change, which is increasing temperatures, exacerbating drought conditions, and accelerating springtime snow melt. Similarly, smoke emissions from wildfires are also expected to increase in the coming decades and will continue to deteriorate air quality across the western U.S. Wildfire smoke consist of small particulates with a diameter less that 2.5 microns ($PM_{2.5}$), and secondary pollutants such as ozone, both of which can degrade air quality and be harmful when inhaled by humans. Across the globe, an estimated 3.3 million deaths per year can be linked to poor air quality. According to the World Health Organization, 92% of the world's population lives in regions where the air is considered unhealthy. Smoke emitted from biomass burning is estimated to be responsible for 5% of all air quality-related deaths. Furthermore, the toxicity of wildfire smoke is higher relative to other sources of atmospheric pollutants. With smoke emissions projected to increase across the western U.S., tools are needed that can forecast fire growth, fire behavior, and smoke dispersion for fire management operations and for limiting human exposure to poor air quality.

Forecasting Wildfires and Smoke with Computer Simulations

Numerical weather prediction models are powerful tools that can forecast the future state of the atmosphere using mathematical equations that describe how air moves and how heat and moisture are exchanged throughout the atmosphere. Today, weather prediction models have evolved such that they can simulate meteorology with high fidelity and resolve meteorological phenomena at sub-kilometer grid scales. Recent advances in computing technology has facilitated the development and deployment of a new generation of weather and fire forecasting tools; namely, coupled-fire atmosphere models. Coupled fire-atmosphere models can simultaneously forecast meteorology and wildfire growth. In these models, the weather can impact fire growth rates, while energy released by the wildfire also affects local meteorology by modifying near-fire winds and creating a buoyant smoke plume over the fire, i.e., the *wildfire plume rise*. Coupled fire-atmosphere models can also forecast fuel moisture conditions, which greatly influences fire growth rates and fire behavior, and chemical interactions between the smoke and other pollutants in the atmosphere.

WRF-SFIRE-CHEM is a state-of-the-art coupled fire atmosphere model integrated with chemistry, which couples the National Center of Atmospheric Research (NCAR)'s Weather Research and Forecast model (WRF-CHEM) with a fire spread model (SFIRE). WRF-SFIRE is a community, open-source model (*https://wiki.openwfm.org/*) that parameterizes fire progression using a two-dimensional semi-empirical rate-of-spread model. The fire spread model within WRF-SFIRE accounts for the effects of wind, fuel moisture, fuel type, and slope on fire behavior. Further updates were made to WRF-SFIRE that allows the model to estimate smoke emissions based on the amount and type of vegetation that is consumed by the fire. Smoke emissions produced by the wildfire are then lofted upwards by the buoyant wildfire plume rise and detrained into the free atmosphere.

Much of WRF-SFIRE's early development, testing, and data analysis was carried out on CHPC's Kingspeak cluster, where simulated coupled fire-atmosphere simulations used approximately 196 cores and took approximately 3.5 hours to generate a 24-hour forecast. WRF-SFIRE simulations can take up anywhere between 25 to 100 GB of disk space, depending on the size of the run. CHPC computing resources are still used today for WRF-SFIRE model development and research applications.

WRF-SFIRE was used to forecast large wildfire incidents in Utah such as the Pole Creek Fire, which burned over 100,000 acres. During the 2020 fire season coupled fire-atmosphere forecasts were run for multiple 2020 California lightning fires such as CZU Complex, SCU complex and Caramel fire. This year, WRF-SFIRE simulations were generated at San Jose State University Wildfire Interdisciplinary Research Center, which targeted the Dixie, Caldor, McCash, and KNP Complex fires (Figure 1). Smoke and visibility range forecasts from WRF-SFIRE were shared with incident commanders and air quality advisors to assist with firefighting and aircraft operations. Currently, WRF-SFIRE forecasts are automated with a sophisticated system that assimilates fuel moisture observations, satellite detections and aircraft fire observations, and executes forecasts on two dedicated computing clusters with over a thousand of processors.

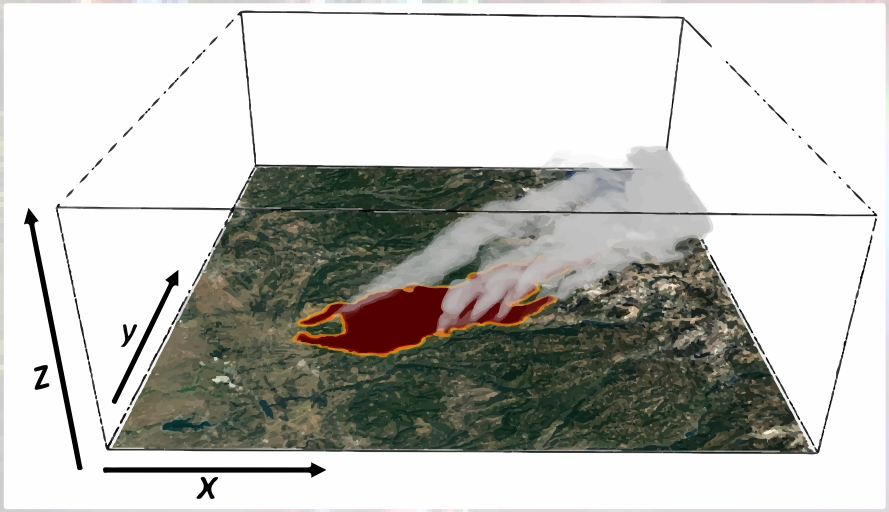

Figure 1. *WRF-SFIRE forecast for the Caldor Fire in California during the 2021 wildfire season.*

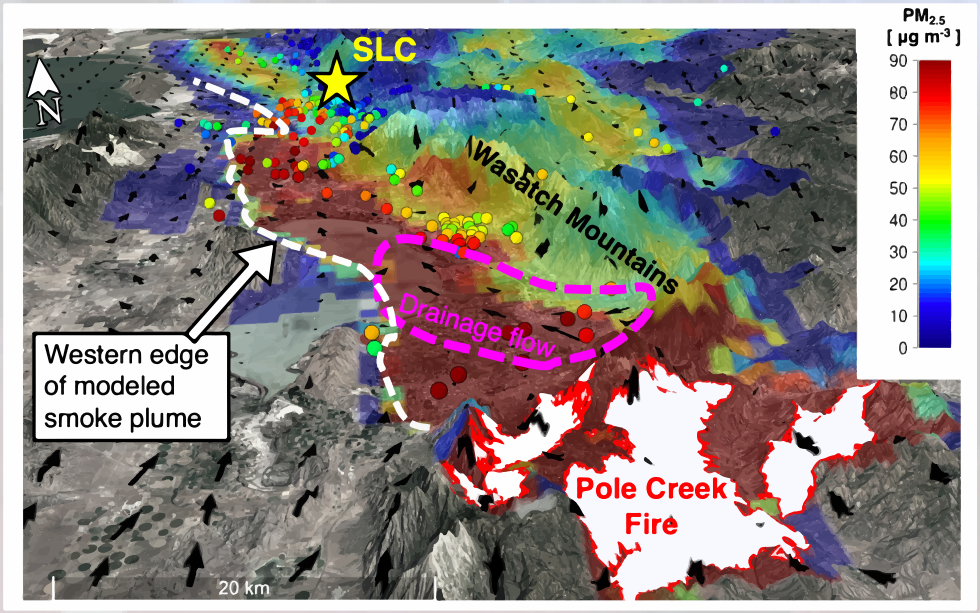

Figure 2. *WRF-SFIRE forecasted surface smoke concentrations ($PM_{2.5}$) from the 2018 Pole Creek Fire near Salt Lake City, Utah (yellow star) during the early morning of September 15th, 2018. Color-filled contours represent model simulated smoke concentrations while the color-filled circles represent smoke measurement from low-cost sensors. The black arrows represent modeled winds from WRF while the white polygon depicts the location of the fire. Modified from Mallia et al. 2020a.*

To evaluate the performance of WRF-SFIRE, we leveraged the University of Utah's Air Quality and yoU (AQ&U) network, which consisted of 509 sensor nodes throughout northern Utah. These measurements were used to evaluate the timing and intensity of the smoke episode, along with the shape and orientation of the smoke plume. Results from this analysis found that WRF-SFIRE was able to capture the shape of the smoke plume as smoke was carried from the fire through Spanish Fork Canyon (Figure 2). Nighttime drainage flow originating from Spanish Fork Canyon then advected smoke towards Utah Lake, and subsequently around Point of The Mountain and into the Salt Lake Valley, before being carried northward towards Idaho. While WRF-SFIRE was able to capture the timing and shape of the smoke plume, forecasted smoke concentrations for the SLV were overestimated by ~20%. Further analyses revealed that WRF-SFIRE overestimated fire growth, which subsequently resulted in over predicted fuel consumption and smoke emissions.

An update was recently made to WRF-SFIRE, which improved how winds were represented within forest

canopies. Preliminary results indicate that adding a canopy model parameterization within WRF-SFIRE dramatically improved forecasted fire growth rates, especially for fires burning in heavily forested areas (Figure 3).

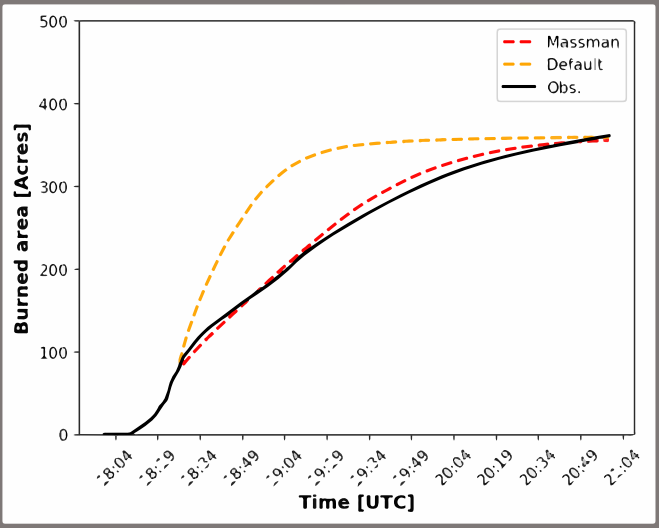

Figure 3. *Time series data of the observed fire growth (solid black line), and the WRF-SFIRE predicted fire growth with (red dash line) and without (orange dash line) a canopy model parameterization.*

Looking ahead

Coupled fire-atmosphere models are becoming increasingly popular tools for fire and smoke forecasting. With wildfire activity projected to increase in the coming decades, it will be crucial to further expand the forecasting capabilities of coupled fire-atmosphere models and to integrate these models into larger forecasting systems. Emerging technologies, such as satellite observations will also play an increasingly important role towards improving fire and smoke forecasting. For research applications, coupled fire-atmosphere models such as WRF-SFIRE will also play a crucial role towards better understanding fire behavior, elucidating important chemical mechanisms that control smoke plume chemistry, and projecting how climate change might impact wildfires in the future.

WRF-SFIRE's development and operational forecasts have utilized CHPC computing resources, and much of the work that has been achieved here would not have been possible without the support of CHPC. To learn more about coupled fire-atmosphere modeling with WRF-SFIRE, the authors encourage the readers to visit our web page at: *https://wiki.openwfm.org/*. To learn more on what Derek is up to with his research, visit his web page at: *https://home.chpc.utah.edu/~u0703457/dereks_homepage/*.

Recent operational forecasts performed using WRF-SFIRE are available at: *https://www.sjsu.edu/wildfire/wildfire-information.php*.

References:

Mallia, D. V., A. Kochanski, K. E. Kelly, R. Whitaker, W. Xing, L. Mitchell, A. Jacques, A. Farguell, J. Mandel, P.-E. Gaillardon, T. Becnel, and S. Krueger (2020a): Evaluating wildfire smoke transport within a coupled fire-atmosphere model using a high-density observation network for an episodic smoke event along Utah's Wasatch Front. J. Geophys. Res., 125, e2020JD032712.

Mallia, D. V., A. Kochanski, S. Urbanski, J. Mandel, A. Farguell, and S. Krueger (2020b): Incorporating a canopy parameterization within a coupled fire-atmosphere model to improve a smoke simulation for a prescribed burn. Atmosphere, 11(8), 832.

Setting up Community Containers in the User Space

Martin Cuma, CHPC Scientific Consultant

Computation applications are becoming increasingly complex to install, with many dependent programs and libraries. CHPC often installs these applications manually, or with the Spack package manager (*https://spack.readthedocs.io/en/latest/*). Oftentimes, however, before we take the native installation path, we first check to see if there is a container that provides the needed application. Using a program from a container can be easier since someone else has done the difficult task of building the program and all its dependencies, and packaged it all in the container.

Quoting Docker, the pioneer in containerization, documentation, "A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings." (*https://www.docker.com/resources/what-container*).

While CHPC Support Staff will do container installations for the user upon request, we believe that in many cases the container installation is simple enough that some users will be able to do the installation themselves. Details on this process are provided in this article. For the purpose of this article the installation of the BLAST, a popular genomics application, will be used.

Finding a container

There are numerous repositories of community-built containers, the most popular being DockerHub, *https://hub.docker.com/*. Be aware that there may be many dif-

ferent containers for the application in need so some scrutiny needs to be used to pick the appropriate one. While containers in theory run everything in the user space, eliminating possibly malicious behavior, there are still occasional vulnerabilities found in the container runtimes and as such it is important to use only community containers from reliable sources. Our approach is to trust containers from credible sources, such as organizations, with extra point given if they make available their container build files. Be cautious about containers built by individuals without any further information about how the container was built.

Start with doing an internet search for the application in question along with the dockerhub keyword, e.g., search with *dockerhub blast*. The two top search returns are `ncbi/blast` and `biocontainers/blast`. Both are good; NCBI is the creator of the program, while biocontainers is an organization providing containers for biological sciences. An added benefit of the `biocontainers/blast` is that it provides the **Source Repository** (in the right side of the DockerHub web page – see Figure 1.). Another useful metric on the trustworthiness of the container is the **Pulls** statistic in the upper right corner of the DockerHub web page – the number of times this container has been downloaded. The more the better. Both containers check this mark as well. Another useful metric is when was the container last updated. The `biocontainers/blast` is older, which may favor the `ncbi/blast` if one wants the latest version.

Installing the Container

Before downloading the container, one needs to think about where to put it. Home directory is a good possibility, but keep in mind that most home directories have 50 GB quota and containers can get large, in the order of few GB each, so watch the space they use. Research groups with purchased group storage will be better served to keep container images there. We also recommend to put the containers to a well-defined location, for example in a directory named `containers`.

The next step is to choose what container runtimes to use. There are several, but most have limitations that make them less fit for HPC use, which is why CHPC supports Singularity (*https://singularity.hpcng.org/*). To create (build) a Singularity container hosted on Dockerhub, first load the Singularity module. Then copy the last part of the **Docker Pull Command** in the Dockerhub container page, which is the container's address, and run the singularity build command. In some cases, like the biocontainers/blast example, we also need to specify the build tag. Click on the **Tags** tab in the DockerHub page, and copy the container address including the tag:

```
$ module load singularity
$ singularity build blast.sif docker://biocontain-
ers/blast:v2.2.31_cv2
```

The Singularity will download all the container layers, and build a singularity container in a single file called `blast.sif`.

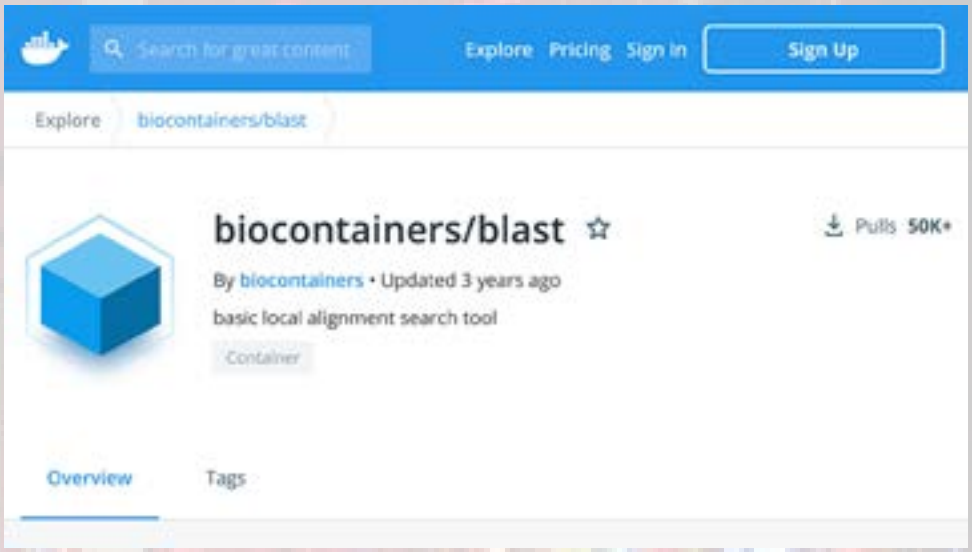

Figure 1. *DockerHub webpage for BLAST container from Biocontainers*

Exploring the Container

After creating the container, it is useful to open a shell in the container to explore it. That can help with figuring out where the programs that we need are installed, and if there's something else, like databases, that need to be provided externally.

```
$ singularity shell blast.sif
Singularity> which makeblastdb
/opt/conda/bin/makeblastdb
```

Notice that when the shell is in the container, the prompt has changed to `Singularity>`, to signalize that the shell is now using the container. The which command returns a path to the program in question. This is helpful to see if the programs we need are indeed found in the container, or, better to say, are included in the binary search path. If the needed program is not found, look through the container file system to try to find it, focusing on common directories like `/opt` or `/home`.

Once we have found that the binaries in the container are there, we can run commands directly from the container using the singularity exec command:

```
$ singularity exec blast.sif makeblastdb -help
USAGE
  makeblastdb [-h] [-help] [-in input_file] [-input_
type type]
```

The application's commands executed this way behave the same as if they were executed natively.

Creating a Module File

While we can run programs directly from the container as shown above, it is cumbersome as one has to remember to use the singularity command and the path to the container. What we do when we deploy containers CHPC-wide is to create a module file that sets up aliases for select programs from the container, which allows to map the programs from the container to the outside. The modules approach also makes it easy to maintain and use different versions of the program.

We first need to create a custom modules environment, documented in detail at our help pages (*https://www.chpc.utah.edu/documentation/software/modules-advanced.php#custom*). The first steps are to create a directory where the modules will be and a subdirectory here where we put the module for the application we are installing, followed by adding the module file, based on the template that we provide:

```
mkdir ~/MyModules
mkdir ~/MyModules/blast
cp   /uufs/chpc.utah.edu/sys/modulefiles/templates/
container-template.lua   ~/MyModules/blast/2.2.31.
lua
```

The first part of the module file, which has the program specific information, will need to be modified for your container:

```
-- required path to the container sif file
local  CONTAINER="/uufs/chpc.utah.edu/common/home/
u0101881/containers/blast.sif"
-- required text array of commands to alias from
the container
local  COMMANDS  =  {"makeblastdb","blastn","blast-
p","blastx"}
```

The above two are required entries, with the `CONTAINER` variable being the full path to the container's SIF file, and the `COMMANDS` variable, which list the program commands in the container that we want to map in the module. Here we listed the few most commonly used BLAST commands in order to conserve space, but, there are about another dozen commands that should be listed for full functionality.

In addition, you may also want to update the following lines:

```
-- these optional lines provide more information
about the program in this module file
whatis("Name         : BLAST")
whatis("Version      : 2.2.31+")
whatis("Category     : genomics")
whatis("URL     : https://blast.ncbi.nlm.nih.gov")
whatis("Installed on : 10/05/2021")
whatis("Installed by : Martin Cuma")
```

Using Programs from the Container

To activate the user-based container module, add the user modules path and the load our new container module:

```
$ module use ~/MyModules
$ module load blast/2.2.31
```

We can then test that the program commands function and have the correct version:

```
$ blastp -help
…
DESCRIPTION
   Protein-Protein BLAST 2.2.31+
…
```

After testing in the terminal, we are ready to create a SLURM job script to run on our clusters, keeping in mind that BLAST does allow parallel execution over multiple threads within a single node. For additional information on using SLURM on CHPC resources see *https://www.chpc.utah.edu/documentation/software/slurm.php*. An example batch script to do this:

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --account=owner-guest
#SBATCH --partition=kingspeak-guest

module use ~/MyModules
module load blast/2.2.31
MYDIR=`pwd`
curl   -O   ftp://ftp.ncbi.nih.gov/refseq/D_rerio/
mRNA_Prot/zebrafish.1.protein.faa.gz
gunzip zebrafish.1.protein.faa.gz
makeblastdb -in zebrafish.1.protein.faa -dbtype prot
curl  https://www.uniprot.org/uniprot/P04156.fasta
>> P04156.fasta
blastp  -query  $MYDIR/P04156.fasta  -db  $MYDIR/
zebrafish.1.protein.faa -num_threads $SLURM_TASKS_
PER_NODE -out results.txt
```

We can now submit this job by:

```
$ sbatch blast.slr
```

Further Directions

While the approach described above will work for most programs in containers, there are some containers that expect certain locations of files or databases which require more customizations, or that don't have the executables readily available. If you run into problems with setting up your container or its module file, we will be happy to help at *helpdesk@chpc.utah.edu*.

R Version Updated

Wim Cardoen, CHPC Scientific Consultant

The statistical package R has been updated to its latest release v.4.1.1 ("Kick Things"). The CHPC version has been compiled using gcc 10.2.0 and linked using Intel's MKL library (version 2021.1.1).

The corresponding environment can be loaded as follows:

```
module load R/4.1.1
```

The environmental variable OMP NUM THREADS has been set to 1 in the LMod module. To fully use its multi-threaded functionality (e.g. on a compute node) we recommend to set the aforementioned environmental variable to the number of available threads/tasks.

On a compute node this would require the following command:

```
export OMP_NUM_THREADS=$SLURM_NTASKS # Bash Shell
```

Or

```
setenv OMP_NUM_THREADS $SLURM_NTASKS # Tcsh Shell
```

Additional information on the use of R on CHPC resources can be found at *https://www.chpc.utah.edu/documentation/software/r-language.php*.

Managing Your Own Software Installations Using Anaconda

Brett Milash, CHPC Scientific Consultant

Introduction

As the software environment at high-performance computing centers becomes more complex, with seemingly countless new software packages and updates becoming available daily, the task of managing a central software catalog becomes nearly impossible, particularly when programs like R and python do not support multiple different versions of the same libraries. CHPC has almost 1,700 software modules installed at this point, and only a handful of those modules are relevant to an individual user. A solution to this problem is for users to manage their own software installations as needed. One of the best tools available to do this now is Anaconda (*https://www.anaconda.com*), a freely available package manager that simplifies installing, updating, and deleting software packages in your own disk space at CHPC. When coupled with the software module system LMod, anaconda enables you to install and manage your own versioned software repository.

Rationale

The rationale for using anaconda with LMod is driven by several factors: the requirement for a package manager that can handle a wide variety of software packages, the need to isolate mutually incompatible software packages, and the desire to maintain multiple versions of the same package. The combination of anaconda and lmod meets all these needs. Anaconda.org (*https://anaconda.org*) hosts over 7,500 different packages, each package listing its prerequisites or dependencies. When these packages are installed each with their own copy of anaconda the possibility of package incompatibility is removed. The lmod software module system (*https://lmod.readthedocs.io*) provides the capacity to load or unload software modules in your shell environment, and can do so while supporting multiple versions of each software package if desired.

Recommendation

The strategy we recommend is to install a separate copy of miniconda3, a lightweight version of anaconda, with each software package you install, and create a LMod module to load or unload that package from your environment.

For illustration purposes, imagine I want to use scikit-learn (*https://scikit-learn.org*), a python-based machine learning tool kit. The steps to install my own copy of scikit-learn are as follows:

1. Anaconda installation

Download the miniconda3 installer:

```
$ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

The above command simply downloads the latest miniconda3 installer from the repo.continuum.io site into my current directory. Once the download is completed, I run the installer:

```
$ bash ./Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/software/pkg/scikit-learn/1.0 -s
```

What do all those arguments mean? The "`-b`" runs the installer in batch mode, so no interaction with the script is necessary. The "`-p $HOME/software/pkg/scikit-learn/1.0`" argument installs the software in my `$HOME/software/pkg` directory, creating a scikit-learn/1.0 directory for this particular package. The "`-s`" argument prevents the installer from altering my dot files, so my shell environment will not be changed. We will do that part through the module file instead. When the installer is finished, the `$HOME/software/pkg/scikit-learn/1.0` directory will have been created, and it will contain just over 300 Mb of software for the miniconda3 installation (compared to many gigabytes for a full anaconda installation).

2. Module creation and activation

Next, I need to create and load a module file for the package. The module file is responsible for setting up my shell environment when I load the module and restoring my environment when I unload the module. To do this I need to instruct the module system to use my module files, create a directory for my new module file, and download the CHPC's template module file:

```
$ module use $HOME/MyModules
$ mkdir -p $HOME/MyModules/scikit-learn
$ cd $HOME/MyModules/scikit-learn
$   wget   https://raw.githubusercontent.com/CHPC-UofU/anaconda-modules/master/miniconda3/latest.lua
$ mv latest.lua 1.0.lua
```

Notice I've renamed the template file to "`1.0.lua`" since I am installing version 1.0 of scikit-learn. Finally, I need to change one location in the template file, to point to my software install location. This location is on line 7 of the file. I need to change:

```
local myanapath = "software/pkg/miniconda3"
```

to

```
local myanapath = "software/pkg/scikit-learn/1.0"
```

This is the same location I specified when I ran the miniconda3 installer above.

Now I load the module, and confirm that it has loaded:

```
$ module load scikit-learn/1.0
$ which conda
~/software/pkg/scikit-learn/1.0/bin/conda
```

Here I can see that the "`conda`" command that I'm running is in the miniconda installation that I just created. So far so good.

3. Package installation

Following the scikit-learn instructions, I install the software with the conda command:

```
$ conda install -c conda-forge scikit-learn
```

When the installation has completed, I can see the "python" I'm executing is from the scikit-learn/1.0 directory I've created, and that when I import the sklearn module, it is coming from within the same directory tree, by doing the following command:

```
$ which python
~/software/pkg/scikit-learn/1.0/bin/python
$ python
Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:20:46)
[GCC 9.4.0] on linux
```

Type "`help`", "`copyright`", "`credits`" or "`license`" for more information.

```
>>> import sklearn
>>> sklearn.__file__
'…/software/pkg/scikit-learn/1.0/lib/python3.9/site-packages/sklearn/__init__.py'
```

4. Software use

With the module created and loaded, and software installed, the executable programs' directory will be on my PATH environment variable, and thus the executables are easily found by the shell when I execute them. In this example that executable is python, since scikit-learn is a library used in python scripts, but other anaconda packages include multiple executable programs that will be installed, so this strategy is not just for python.

When I want to use sckit-learn in future sessions, all I need to do is execute the "`module use $HOME/MyModules`" and "`module load scikit-learn/1.0`" commands to access this module. When I'm finished with `scikit-learn` I can execute "`module unload scikit-learn`" to remove it from my environment. Many users will place the "`module use $HOME/MyModules`" command in their "`.custom.sh`" or "`.custom.csh`" file for convenience, especially if they plan on using scikit-learn regularly.

5. Managing updates

If scikit-learn is updated, I can repeat this process to create separate installations for version 1.1, 1.2, 2.0, and so on, that are completely independent of version 1.0. This capability allows for the preservation of my existing software environment while I experiment with the

new version.

6. Use of group space

Many research groups have shared group disk space at CHPC, and this space can be used for software installations as well. The only changes that need to be made to use this space are to create and use a shared software directory for miniconda installations, create and use a shared module directory for LMod module files, and execute the "`module use`" command to select that shared module directory.

Experienced anaconda users may have noticed we are not suggesting the use of conda environments. These environments allow multiple software packages to "piggyback" on a single miniconda installation, reducing the disk space overhead, and requiring "`activation`" or "`deactivation`" similar to LMod modules. In our experience, however, conda environments create a confusing situation where the software install location is not readily apparent, make for more complex LMod modules, and do not always work well for tcsh users. As the disk space overhead of a miniconda installation is only 300 Mb, we feel that software installations without conda environments is a better solution.

Locating software

Software that is installed by anaconda can be found at *https://anaconda.org*. Many bioinformatics packages are available at *https://anaconda.org/bioconda/*.

Caveats

Some conda installations run into problems, most frequently due to package version number conflicts. In these cases it is possible to install packages using "pip" (for python packages) or from source code, using the familiar "`configure / make / make install`" procedure. For such source code installations it will be necessary to use the "`configure --prefix`" argument to specify the install location.

Python Version 3.9.7 Available

Wim Cardoen, CHPC Scientific Consultant

Python version 3.9.7 (Released on August 30, 2021) was installed within the /uufs tree. The distribution was compiled with gcc version 10.2.0.

Within the distribution an array of scientific packages were installed. Among them the latest versions of NumPy, Scipy, MatplotLib, Jupyter, Pandas, SymPy, .. which were all built with the Intel MKL library for performance.

To load Python 3.9.7:

```
module load python/3.9.7
```

Additional information on the use of python on CHPC resources can be found at *https://www.chpc.utah.edu/documentation/software/python-anaconda.php*.

Please acknowledge the use of CHPC Resources

If you use CHPC computer time or staff resources, we request that you acknowledge this in technical reports, publications, and dissertations. An example of what we ask you to include in your acknowledgement is:

"A grant of computer time from the Center for High Performance Computing is gratefully acknowledged."

If you make use of the CHPC Protected Environment, please also acknowledge the NIH shared instrumentation grant:

"The computational resources used were partially funded by the NIH Shared Instrumentation Grant 1S10OD021644-01A1."

Please submit copies or citations of dissertations, reports, pre-prints, and reprints in which CHPC is acknowledged by sending to *helpdesk@chpc.utah.edu*.

The University of Utah
University Information Technology
Center for High Performance Computing
155 South 1452 East, Room 405
SALT LAKE CITY, UT 84112–0190

Center for HIGH PERFORMANCE COMPUTING
THE UNIVERSITY OF UTAH