Mamivand Lab Summer 2022 Guide

Leo Smith

July 22, 2022

1 Introduction

This guide is meant to be an outline of the research project of Leo Smith and Carl Agren in the summer of 2022. A repository for code and data can be found on GitHub using this link. For further questions, feel free to contact Leo at lsmith7@cord.edu.

2 MOOSE

MOOSE is a tool that we use to run phase-field simulations of spinodal decomposition according to the Cahn-Hilliard Equation. In order to use MOOSE, you need to be able to access the Borah cluster since it provides the computing power required for the simulations. Information on the Borah cluster and how to use it can be found at the Research Computing website. Once you have access to Borah, you can compile MOOSE into your BSU scratch directory. These steps guide you through that process.

2.1 Compiling MOOSE

First, you should login to Borah with the following command.

```
ssh <username>@borah-login.boisestate.edu
```

Then, you will be prompted to enter your password. Note that your password will not show up when typing; this is normal. Press enter to login.

Before you do anything, make sure the required modules are loaded. In this project, we used the "slurm" module and the "moose-dev-gcc" module. If these modules are not already loaded, use the following commands:

```
module load slurm
module load moose-dev-gcc
```

The next step is to go into your "scratch" storage and make a projects folder, which is where all your big files will be stored.

```
cd scratch
mkdir projects
cd projects
```

Now that you are inside the projects folder, you can clone MOOSE from GitHub into this folder, go into the folder, and switch to the master branch.

```
git clone https://github.com/idaholab/moose.git
cd moose
git checkout master
```

The next step is to compile PETSC and libmesh. Compiling these packages can take several hours. The following commands *usually* work, but there could still be some errors. In the case of errors, the MOOSE discussion forums and the Research Computing team at BSU are good places to go for help.

```
unset PETSC_DIR PETSC_ARCH
./scripts/update_and_rebuild_petsc.sh --download-cmake
export PETSC_DIR=/bsuhome/<username>/scratch/projects/moose/petsc
export PETSC_ARCH=arch-moose
./scripts/update_and_rebuild_libmesh.sh
```

2.2 Building a Test Application

Now that MOOSE has been successfully compiled, you can create an application. First navigate into your projects directory. If you currently in the MOOSE folder directory, you can use the following command to enter the parent (projects) directory.

```
cd ..
```

Now, you can make the application using the following commands.

```
./moose/scripts/stork.sh <application_name>
cd <application_name>
```

Next, you need to select which physics modules in MOOSE that you want to use. For phase field simulations, these modules are the "phase field" and "tensor mechanics" modules. To select modules, you must edit the Makefile. This requires using a text editor, where, in this case, the "vim" editor is used (but you can use any editor you want).

```
vim Makefile
```

Press "I" to start editing the file. Change the required physics modules from "no" to "yes". Type ":wq" to save the file and exit vim.

Run the following commands to compile the application and run the tests.

```
make -j 8
./run_tests -j 8
```

If the tests passed, you are ready to run phase-field simulations in MOOSE!

2.3 Phase-Field Simulations

In most cases, you want to run many phase-field simulations at once, with each job having a different set of parameters. This is done using a for-loop that iterates over a comma-delimited spreadsheet containing several rows of different parameter combinations. So, this guide will cover the process of running a batch of phase-field simulations in MOOSE. The following files are needed:

1. **FeCrCo.i** This is the input file (written in C) that contains the variables, kernels, equations, and parameters that MOOSE uses to solve the Cahn-Hilliard equation. Specifically, this file is used to specify the mesh size, the constants and equations, the ending time and time steps, the output file format, among others. For a good guide to the syntax used in this input file, see the MOOSE syntax guide.

2. **Bash_CSV.sh** This file is used to iterate over the input data CSV and run a job with the parameters from each row. This is also the file where the name of the job directory (containing the input parameters) is created. Most importantly, this file substitutes the input parameters into FeCrCo.i using the "sed" command. Note: in this file, you must change the [name]-opt to the name of your application in order for it to work.

3. **slurm-batch.bash** This file is used to officially schedule each job on Borah using the "sbatch" command. It is also where the name of the job, the time limit, and the number of nodes, CPUs, and cores are specified. Note: in this file, make sure you load all of the modules you need for the job, such as moose-dev-gcc and slurm. There is also a [name]-opt that you should change such that it matches the name of your application.

4. **input_data.csv** This file contains rows of various combinations of input parameters to be used in phase-field simulations.

5. **csvgen_database.py** This is a python file that is used to generate an input data CSV containing the parameter multiples that you specify.

6. **parameters.py:** This is a python file that is used to get the initial parameter values, which are then multiplied in various combinations by the csvgen file to create an input data spreadsheet.

All of these files should be transferred into the application you just made. A good way to do this is by using FileZilla. Once the files are in the application folder, allow access to execute Bash_CSV.sh using the following command.

```
chmod +x Bash_CSV.sh
```

Then, simply execute the script as follows. All of the jobs will be scheduled on Borah.

```
./Bash_CSV.sh
```

2.4 Paraview

Once one or all of your MOOSE jobs have finished on Borah, you can start looking at the output files. In our project, we configured the input file to generate two types of output files: exodus and CSV files. The CSV file contains the min and max compositions of each element at each time step. The exodus files contain the data needed to generate microstructure images with Paraview or Peacock. If you have a Mac or Linux machine, Peacock can work for viewing microstructures from exodus files. Windows users will need to use Paraview, which is what we used for this project.

To use Paraview, first download the application to your local device. Since exodus files are large, this can take a bit of time. Then, simply open Paraview and navigate into the subdirectory of your application containing the exodus files. Open the ".e" file in Paraview and click "Apply" under Properties. Then, clip the drop-down labeled "Solid Color" and choose "c1". This will show you the microstructure of Fe at each time step. To see all the time steps in sequence, press the play button at the top.

From here, you can extract the microstructure as a PNG image. In our project, we used python codes to extract the microstructure images in batches. The codes for batch extraction are located in GitHub.

3 Troubleshooting

1. **Locale error on Windows**

   To fix this error, you can run the following commands.

   ```
   export LC_CTYPE=en_US.UTF-8
   export LC_ALL=en_US.UTF-8
   ```

   To make sure these commands are run every time you open up a terminal, append them to your .bashrc file in Borah.

2. **"Cannot find -lx11"**

   This error can fixed with the following command. Make sure to append this to your slurm file as well.

   ```
   export LIBRARY_PATH=/cm/shared/software/opt/linux-centos7-x86_64/gcc-9.2.0/libx11
       -1.7.0-u5s2reu2nx5vykhfimela2hmutk2vrg3/lib:${LIBRARY_PATH}
   ```

3. **Line breaks error**

   If you edit files in a Windows application such as Excel or VS Code, sometimes the line breaks format get changed. In this case, you need to change it back to Unix line breaks. This can be done with the following command:

   ```
   dos2unix <file>
   ```

4 Useful Commands

```
cd <subdirectory>
```

Allows you to change your directory in the Linux environment. Use "." for the current directory, ".." for the parent directory, and "∼" for the home directory.

```
ls
```

Lists all of the contents of the current directory. To see the hidden files such as .bashrc, use "-a" as well.

```
rm <file>
```

Removes a file. Use "-r" to delete an empty directory, and "-rf" to delete a non-empty directory.

```
cp <file> <location>
```

Copies a file to a new location. Use "-R" to copy a directory.

```
mv <file> <location>
```

Moves a file to a new location. This command can also be used to rename a file or directory when "new_location" is not a directory.

```
du -hs
```

This is a nifty command that shows you the total amount of storage utilized by a directory. It can be useful for keeping track of how much space you are using within Borah.

```
squeue
```

Shows the current jobs running in Borah.

```
scancel <job_number>
```

Cancels a current Borah job.

```
sbatch <file>
```

Runs a slurm file in Borah.

```
module load <module>
```

Loads a module.

```
module unload <module>
```

Unloads a module.

```
module list
```

Lists all available modules.

```
module purge
```

Removes all of the currently loaded modules.

5 Acknowledgements

We thank Amir Abbas Kazemzadeh Farizhandi for providing template codes for this project. We also thank Dr. Mahmood Mamivand for his exceptional guidance. Finally, we thank the Research Computing team at BSU for guidance in utilizing the Borah cluster and compiling MOOSE.