Preface

It is absolutely crucial that you do not copy paste the commands given and instead you must type them yourself. The reason for this is there may be line breaks in this document that may be misinterpreted by the Lonestar6 terminal. It is also important to read the whole document and not skip to just the commands. Every command has context to it and you must understand it.

Each command will start with the **$** character just as seen in the lonestar6 terminal. Don't actually type the **$** character, it is simply there to show the start of a command. However, there is an instance in which the **$** character is literal and must be typed and that is when using the environment variables **${HOME}** and **${SCRATCH}.** These are environment variables defined by lonestar6 to link to the home and scratch directories respectively. When using them in commands they must be typed exactly as shown above.

Useful Linux Commands

$ cd – change directory (this command with no arguments will change to home directory)

$ cd .. – change to previous directory

$ ll – list files vertically with information

$ ls – list files horizontally with no file information

$ pwd – print current directory path

$ vim – open file in the vim text editor

$ touch - create a file

$ cp – copy a file from one directory to another

$ mv – move a file or rename a file if in the same directory

This guide id very in depth but I still highly recommend being comfortable with the Linux terminal as well as using vim. At the end of this document there are YouTube video links for further reference.

Login to Lonestar6:

Download PuTTY and open it. By default, it should open on session; if it doesn't, click on session. In the host name box type **[TACC Username]@ls6.tacc.utexas.edu** and click ok. This will allow you to SSH into Lonestar6. After this, you will be prompted to type in your password and an authentication code that will be sent to your phone. After completing these steps, you should now be on the Lonestar6 home page and directory.

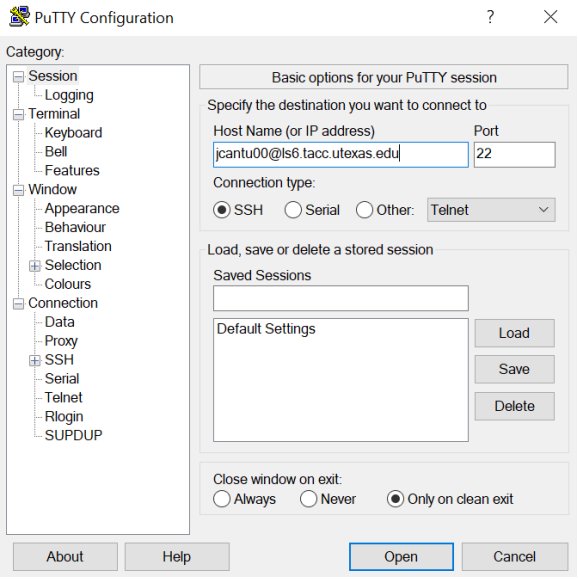

Figure 1 - PuTTY Screen

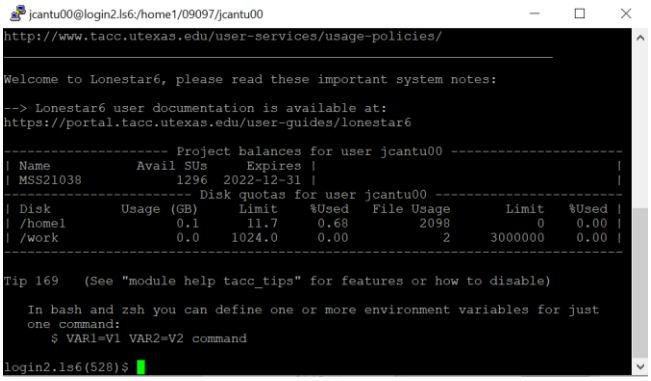

Figure 2 - Lonestar6 Home Page

Setting Up CCAMA and Armadillo Directories into Lonestar6:

The first step is to create the necessary directories we will be using to store the CCAMA and Armadillo files. Start by creating the directory that will hold everything related to the CCAMA Armadillo implementation by typing:

$ mkdir CCAMA_Armadillo_Implementation

Next type the following commands in sequential order:

$ cd CCAMA_Armadillo_Implementation
$ mkdir source_code
$ mkdir Armadillo_Library

What we've done here is first change directories into the **CCAMA_Armadillo_Implementation** directory. Then, we created two new directories, **source_code**, for storing our CCAMA cpp and header files, and **Armadillo_Library** for storing all the files needed for the armadillo installation.

Before we start to download and install Armadillo, we must understand that Armadillo is a wrapper library. In essence it serves to wrap more complex libraries such as OpenBLAS to create easier to understand syntax similar to MATLAB. What this means is that before installing

Installing OpenBLAS onto the Lonestar6 Environment:

Armadillo we must install OpenBLAS so that Armadillo can then wrap around it. To do this, we will navigate into the newly created **Armadillo_Library** directory and create a new directory called **OpenBLAS_Library** to store all our OpenBLAS files. Type the following commands to do so.

$ cd Armadillo_Library
$ mkdir OpenBLAS_Library
$ cd OpenBLAS_Library

Now that we have created and navigated into the directory that will store our OpenBLAS files, we must download openBLAS from its github repository by typing:

$ git clone https://github.com/xianyi/OpenBLAS.git

If done correctly after the download, your screen should appear like the figure below and a new directory called **OpenBLAS** should've been created as shown. You can see the directory by typing **ll** into the terminal.

```
login1.ls6(1023)$ git clone https://github.com/xianyi/openBLAS.git
Cloning into 'openBLAS'...
remote: Enumerating objects: 64042, done.
remote: Counting objects: 100% (2177/2177), done.
remote: Compressing objects: 100% (1707/1707), done.
remote: Total 64042 (delta 2047), reused 526 (delta 470), pack-reused 61865
Receiving objects: 100% (64042/64042), 43.78 MiB | 36.10 MiB/s, done.
Resolving deltas: 100% (52520/52520), done.
Updating files: 100% (11935/11935), done.
login1.ls6(1024)$ ll
total 8
drwx------ 18 jcantu00 G-824208 4096 Dec 16 23:28 openBLAS
```

Figure 3 - Finished Downloading openBLAS from github Screen

Now navigate into the **openBLAS** directory by typing

$ cd OpenBLAS

In here we will make the openBLAS library and tell it to use the GNU fortran compiler gfortran

$ make FC=gfortran

Following this, a lot of configuration stuff will start popping up, along with a few warnings. Simply allow all of that to finish on its own, it should just take about 20 seconds, and then if your screen looks like the figure below, you've done it correctly.

If this doesn't' work and error keeps popping up, remove the newly created **OpenBLAS** directory and exit lonestar6 and give some time before returning or reset your computer and give it some time. It should eventually work.

```
make[1]: Leaving directory '/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/OpenBLAS/exports'

 OpenBLAS build complete. (BLAS CBLAS LAPACK LAPACKE)

  OS               ... Linux
  Architecture     ... x86_64
  BINARY           ... 64bit
  C compiler       ... GCC  (cmd & version : cc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-10))
  Fortran compiler ... GFORTRAN  (cmd & version : GNU Fortran (GCC) 9.4.0)
  Library Name     ... libopenblas_zenp-r0.3.21.dev.a (Multi-threading; Max num-threads is 128)

To install the library, you can run "make PREFIX=/path/to/your/installation install".

Note that any flags passed to make during build should also be passed to make install
to circumvent any install errors.
```

Figure 4 - Finished making openBLAS screen

Finally, we can install openBLAS. We do this by using the **make install** command and specify our installation location with the **PREFIX** option. Type the following

$ make PREFIX=${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library

After installation, your screen should look like the figure below.

```
login1.ls6(1008)$ make PREFIX=${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library install
make -j 128 -f Makefile.install install
make[1]: Entering directory '/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/OpenBLAS'
Generating openblas_config.h in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/include
Generating f77blas.h in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/include
Generating cblas.h in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/include
Copying LAPACKE header files to /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/include
Copying the static library to /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib
Copying the shared library to /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib
PKGFILE="/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib/pkgconfig/openblas.pc"
Generating openblas.pc in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib/pkgconfig
Generating OpenBLASConfig.cmake in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib/cmake/openblas
Generating OpenBLASConfigVersion.cmake in /home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib/cmake/openblas
Install OK!
make[1]: Leaving directory '/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/OpenBLAS'
login1.ls6(1009)$
```

Figure 5 - Finished Installing OpenBLAS Screen

Now let's navigate back to the **OpenBLAS_Library** directory to check if everything was installed correctly.

$ cd ..
$ ll

After typing the above commands, you should see the figure below

```
login2.ls6(1023)$ cd ..
login2.ls6(1024)$ ll
total 12
drwx------  2 jcantu00 G-824208    6 Dec 13 02:50 bin
drwx------  2 jcantu00 G-824208  169 Dec 13 18:11 include
drwx------  4 jcantu00 G-824208  180 Dec 13 02:50 lib
drwx------ 18 jcantu00 G-824208 8192 Dec 13 02:50 OpenBLAS
login2.ls6(1025)$
```

Figure 6 - Contents of OpenBLAS_Library Directory After Installation

Now let's check if the lib directory installed correctly. Type the following commands and your screen should look like figure 7.

$ cd lib
$ ll

```
login2.ls6(1025)$ cd lib/
login2.ls6(1026)$ ll
total 43452
drwx------ 3 jcantu00 G-824208       22 Dec 13 02:50 cmake
lrwxrwxrwx 1 jcantu00 G-824208       30 Dec 13 02:50 libopenblas.a -> libopenblas_zenp-r0.3.21.dev.a
lrwxrwxrwx 1 jcantu00 G-824208       31 Dec 13 02:50 libopenblas.so -> libopenblas_zenp-r0.3.21.dev.so
lrwxrwxrwx 1 jcantu00 G-824208       31 Dec 13 02:50 libopenblas.so.0 -> libopenblas_zenp-r0.3.21.dev.so
-rw-r--r-- 1 jcantu00 G-824208 30010612 Dec 13 02:50 libopenblas_zenp-r0.3.21.dev.a
-rwxr-xr-x 1 jcantu00 G-824208 14482784 Dec 13 02:50 libopenblas_zenp-r0.3.21.dev.so
drwx------ 2 jcantu00 G-824208       25 Dec 13 02:50 pkgconfig
login2.ls6(1027)$
```

Figure 7- Contents of lib Directory in OpenBLAS_Library

If all of this looks correct, then you have successfully installed OpenBLAS onto Lonestar6.

Installing Armadillo onto the Lonestar6 Environment:

Now that OpenBLAS is installed, we can proceed with installing Armadillo. Type **pwd** to check the path you're in. If your path looks like figure 8, then type the following command to navigate into the **Armadillo_Library** directory.

```
login2.ls6(1010)$ pwd
/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib
login2.ls6(1011)$
```

Figure 9 - Current Path

$ cd ../..

Now that we're in the **Armadillo_Library,** we're going to create a directory to store the Armadillo download files.

$ mkdir src
$ cd src

Now we will download the zipped Armadillo file and extract those files. Note that at the time of writing this guide, the most recent version of Armadillo is 11.4.2, if there is new stable version when reading this simply replace those numbers with the new version numbers.

$ wget http://sourceforge.net/projects/arma/files/armadillo-11.4.2.tar.xz
$ tar xf armadillo-11.4.2.tar.xz

Navigate into the newly created armadillo-11.4.2 directory.

$ cd armadillo-11.4.2
$ ll

The contents of the **armadillo-11.4.2** directory should look like the figure below

```
login2.ls6(1023)$ cd armadillo-11.4.2
login2.ls6(1024)$ ll
total 2216
-rw------- 1 jcantu00 G-824208    432 Jun 16  2016 armadillo_icon.png
-rw------- 1 jcantu00 G-824208 206810 Jun 16  2016 armadillo_joss_2016.pdf
-rw------- 1 jcantu00 G-824208 190737 Jun 16  2016 armadillo_lncs_2018.pdf
-rw------- 1 jcantu00 G-824208 225499 Jun 16  2016 armadillo_nicta_2010.pdf
-rw------- 1 jcantu00 G-824208 244498 Jun 16  2016 armadillo_solver_2020.pdf
-rw------- 1 jcantu00 G-824208 420075 Jun 16  2016 armadillo_spcs_2017.pdf
-rw------- 1 jcantu00 G-824208    287 Jun 16  2016 CHANGELOG.html
drwx------ 4 jcantu00 G-824208     53 Jun 16  2016 cmake_aux
-rw------- 1 jcantu00 G-824208  27803 Dec 13 03:39 CMakeLists.txt
-rwx------ 1 jcantu00 G-824208    428 Jun 16  2016 configure
-rw------- 1 jcantu00 G-824208 644694 Jun 16  2016 docs.html
drwx------ 3 jcantu00 G-824208    174 Dec 13 04:04 examples
drwx------ 3 jcantu00 G-824208     57 Jun 16  2016 include
-rw------- 1 jcantu00 G-824208   2095 Jun 16  2016 index.html
-rw------- 1 jcantu00 G-824208  11560 Jun 16  2016 LICENSE.txt
drwx------ 2 jcantu00 G-824208    191 Jun 16  2016 mex_interface
drwx------ 2 jcantu00 G-824208    139 Jun 16  2016 misc
-rw------- 1 jcantu00 G-824208   1027 Jun 16  2016 NOTICE.txt
-rw------- 1 jcantu00 G-824208 234439 Jun 16  2016 rcpp_armadillo_csda_2014.pdf
-rw------- 1 jcantu00 G-824208  18754 Jun 16  2016 README.md
drwx------ 2 jcantu00 G-824208     80 Jun 16  2016 src
drwx------ 2 jcantu00 G-824208     62 Jun 16  2016 tests1
drwx------ 2 jcantu00 G-824208   4096 Jun 16  2016 tests2
login2.ls6(1025)$
```

Figure 10 - Contents of armadilo-11.4.2

There are 2 ways in which Armadillo can be installed. The first method is using the cmake command. This method uses the CMakeLists file to search through your system to look for any dependencies such as OpenBLAS and wraps all dependencies into the Armadillo run time library. The benefit to this is that if you're using multiple dependencies such as OpenBLAS, SuperLU, and ARPACk, then without generating the Armadillo runtime library every time a program is compiled it will have to be linked to each dependency manually instead of simply linking to the Armadillo runtime library.

The other method is linking manually as mentioned previously and is the method we'll be using as we only have one dependency, and it is much easier. Note it's only easier because we have just one dependency, OpenBLAS. To do this we will copy the **include** directory into the **Armadillo_Library** directory.

$ cp -R include ${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library

The **include** directory contains the necessary header files to wrap the OpenBLAS library into Armadillo syntax. To be sure this directory was copied over navigate to the **Armadillo_Library** directory and your screen should now look like figure 11.

$ cd ../..
$ ll

```
login1.ls6(1014)$ cp -R include ${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library
login1.ls6(1015)$ cd ../..
login1.ls6(1016)$ ll
total 0
drwx------ 3 jcantu00 G-824208 45 Dec 18 01:03 include
drwx------ 6 jcantu00 G-824208 79 Dec 17 21:13 OpenBLAS_Library
drwx------ 3 jcantu00 G-824208 73 Dec 17 21:31 src
```

Figure 11 - Contents of Armadillo_Library Directory Following Installation of Armadillo

Armadillo has now been installed. Because we are linking to OpenBLAS manually, there is no need to make the Armadillo library like we did for OpenBLAS. The figure below showcases what your directory tree should now look like.

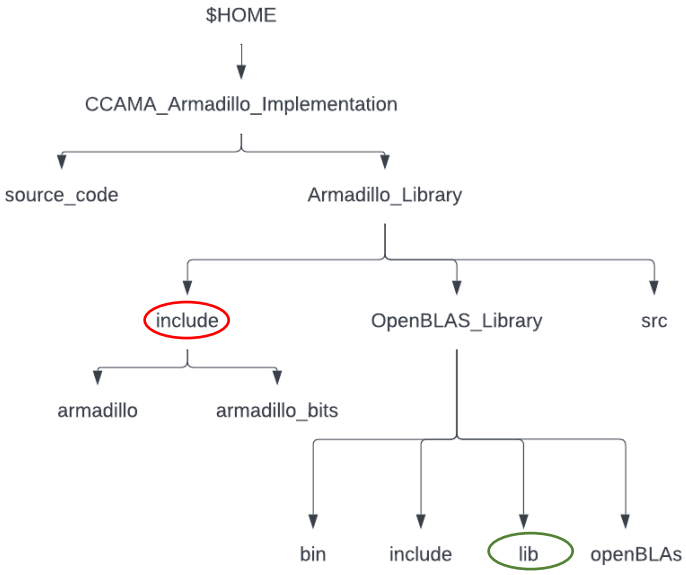

Figure 12 - Directory Tree for Armadillo Setup

Circled in red is the **include** directory that we will tell the compiler to look for the armadillo header files. Circled in green is the **lib** directory that we will tell the compiler to look for to link the Armadillo header files with the OpenBLAS library.

Testing Armadillo with example1.cpp

The armadillo-11.4.2 folder comes with an example to execute to ensure armadillo was installed correctly. Navigate to the **examples** directory inside of the armadillo-11.4.2 directory. In here, there is an example file called example1.cpp. Type the following to view it:

$ cd src/armadillo-11.4.2/examples
$ ll

The below command is used to compile and link example1.cpp. For readability's sake **${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library** will be substituted by **$ARMA_ROOT**. Just remember to actually type the full pathname and not **$ARMA_ROOT.** Also note that this is all one single command. Figure 13 shows the command on a lonestar6 terminal for extra clarification.

$ g++ example1.cpp -o main -std=c++11 -O3 -I$ARMA_ROOT/include -L$ARMA_ROOT/OpenBLAS_Library/lib -Wl,-rpath,$ARMA_ROOT/OpenBLAS_Library/lib -lopenblas

```
login1.ls6(1028)$ g++ example1.cpp -o main -std=c++11 -O3 -I${HOME}/CCAMA_Armadillo_I
mplementation/Armadillo_Library/include -L${HOME}/CCAMA_Armadillo_Implementation/Arma
dillo_Library/OpenBLAS_Library/lib -Wl,-rpath,${HOME}/CCAMA_Armadillo_Implementation/
Armadillo_Library/OpenBLAS_Library/lib -lopenblas
```

Figure 13 - g++ Compiler Command to Compile example1.cpp

Some things to point out in this long compiler command. The compiler we're using is **g++** and the **-o** option allows us to name our executable, in this case main.exe. The **-std=c++11** option tells the compiler to compile against the c++11 standard, needed for OpenMP later. The **-O3** option is compiler optimization. The **-I** option tells the compiler where to find the Armadillo header files. The **-L** option tells the compiler where to find the library to link with. The previous two mentioned commands are for compile time; however, OpenBLAS is a dynamic library and thus must also be linked at runtime. To do this, we use the **-Wl,-rpath,** command to link with the library in the directory specified at runtime. This is the compiler command any time you want to execute code with Armadillo.

To then execute the code type:

$ ./main

If this runs and prints matrices, then Armadillo has been installed correctly.

Running CCAMA with Armadillo - Single core

The first step is to copy all our CCAMA cpp and header files from our local machine to Lonestar6. If using a windows machine, open the command prompt, if using a MAC open the terminal. On your machine, use the **cd** command to navigate to the directory that contains your CCAMA source code. Below is an example of this on my local machine.

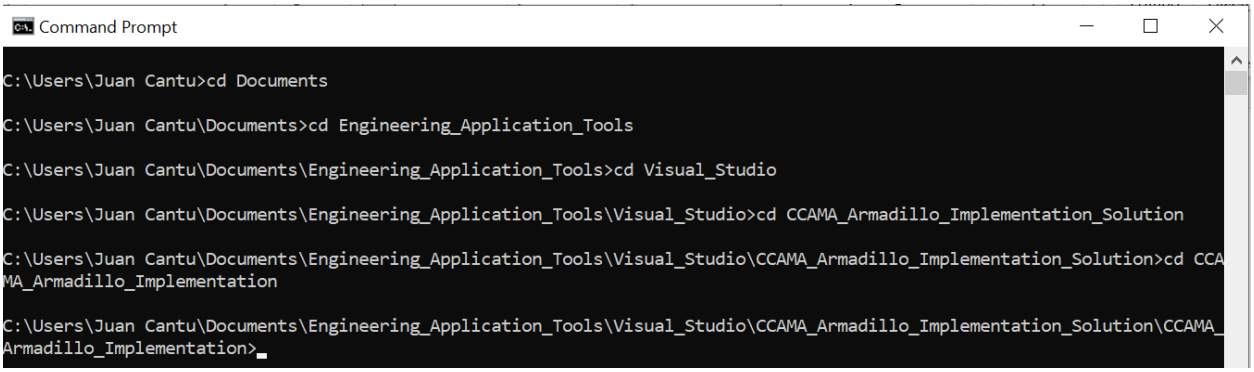

Figure 14 - Using cd to Navigate to CCAMA Files On Local Machine Using Windows Command Prompt

Next, we will copy all our CCAMA files into the **source_code** directory in Lonestar6 by typing the following. **Be sure to replace my TACC username with yours and be sure to note that this is all one long command**.

$ scp Options.h Output.h Lyap.h Lyap.cpp CCAMA.h CCAMA.cpp run_CCAMA.cpp jcantu@ls6.tacc.utexas.edu:/${HOME}/CCAMA_Armadillo_Implementation/source_code

TACC will then ask for a password and number token before completing the copy. If the copy was successful, you should see each file in your **source_code** directory on Lonestar6. On Lonestar6, assuming you're in the **CCAMA_Armadillo_Implementation** directory, navigate to the **source_code** directory using **cd source_code** command. Type **ll** command when in the source_code directory; this will list all files in the directory. Below is what your Lonestar6 screen should look like.

```
login2.ls6(545)$ pwd
/home1/09097/jcantu00/CCAMA_Armadillo_Implementation/source_code
login2.ls6(546)$ ll
total 36
-rw------- 1 jcantu00 G-824208 10078 Nov 30 11:30 CCAMA.cpp
-rw------- 1 jcantu00 G-824208   173 Nov 30 11:30 CCAMA.h
-rw------- 1 jcantu00 G-824208   580 Nov 30 11:30 Lyap.cpp
-rw------- 1 jcantu00 G-824208   100 Nov 30 11:30 Lyap.h
-rw------- 1 jcantu00 G-824208   671 Nov 30 11:30 Options.h
-rw------- 1 jcantu00 G-824208   767 Nov 30 11:30 Output.h
-rw------- 1 jcantu00 G-824208  2865 Nov 30 11:30 run_CCAMA.cpp
login2.ls6(547)$
```

Figure 15 - Lonestar6 source_code Directory

The Windows Command Prompt or MAC Terminal or not needed anymore, so you can exit out of them.

To compile the CCAMA project, you must be in the **source_code** directory. Reminder that this is one command and that **$ARMA_ROOT** should be replaced by **${HOME}/CCAMA_Armadillo_Implementation/Armadillo_Library**:

$ g++ Lyap.cpp CCAMA.cpp run_CCAMA.cpp -o main -std=c++11 -O3 -I$ARMA_ROOT/include -L$ARMA_ROOT/OpenBLAS_Library/lib -Wl,-rpath,$ARMA_ROOT/OpenBLAS_Library/lib -lopenblas

The figure below is the compile command on my Lonestar6 terminal for further clarification.

```
login1.ls6(1011)$ g++ Lyap.cpp CCAMA.cpp run_CCAMA.cpp -o main -std=c++11 -O3 -I${HO
ME}/CCAMA_Armadillo_Implementation/Armadillo_Library/include -L${HOME}/CCAMA_Armadil
lo_Implementation/Armadillo_Library/OpenBLAS_Library/lib -Wl,-rpath,${HOME}/CCAMA_Ar
madillo_Implementation/Armadillo_Library/OpenBLAS_Library/lib -lopenblas
```

Figure 16 - Compile Command for CCAMA

The to run the code type:

$ ./main

Note that normally you will not be executing code in your **source_code** directory. You would only compile and then copy the executable to your **$SCRATCH** directory and submit a job there using the executable. We only executed the code here to ensure everything was installed correctly.

Also note that run_CCAMA produces two csv files, Xout and Zout.

Running CCAMA with Multiple Cores Using SLURM to Submit Jobs

According to the Lonestar6 documentation, jobs should be submitted in the **$SCRATCH** directory and only compiling and file organization should happen in the **$HOME** directory.

To submit a job, we will need to create a slurm file. I recommend creating a template file that you can copy and paste and modify to the scenario. The template slurm file I created was named **RUN_CCAMA_N_exN_THREAD_exT.slurm** and it was created in my **source_code** directory. Below is the file. # denotes a comment except for if SBATCH follows it.

```
#!/bin/bash

#-----------------------------------------------------------------------------------------------------------
#       This is a template slurm file for use with CCAMA
#
#       --replace all exN with the integer N equals in run_CCAMA in the file name aswell
#       --replace all exT with the number of threads being used in the file name aswell
#       --by default the queue is set to deveolpment but you can change to whichever queue
#       --by default the run time is 2 hrs but that can be changed aswell
#       --replace emailaddress.edu with your email
#
#       --Be sure to place this file along with your executable in your SCRATCH directory
#-----------------------------------------------------------------------------------------------------------

#SBATCH -J CCAMA_N_exN_exT                      # Job name
#SBATCH -o CCAMA_N_exN_exT.o%j                  # std output file
#SBATCH -e CCAMA_N_exN_exT.e%j                  # std error file
#SBATCH -p development                          # queue name
#SBATCH -N 1                                    # Number of nodes
#SBATCH -n 1                                    # Number of mpi tasks
#SBATCH -t 02:00:00                             # Run time
#SBATCH --mail-type=all                         # Send email at start and end of job
#SBATCH --mail-user=emailaddress.edu            # Email address

module list
pwd
date

#uncomment this for multithreading or leave it commented if using 1 core
#export OMP_NUM_THREADs=exT

#executes the program
./main_N_exN_THREAD_exT
~
```

Figure 17 - Template Slurm File

The comments I put tell you exactly what to do. In essence replace, “exN” with the value of N and replace “exT” with the number of threads. Do this anywhere those two appear including the file name as it’s a template. To create this type

$ touch RUN_CCAMA_N_exN_THREAD_exT.slurm

And to edit the file type

$ vim RUN_CCAMA_N_exN_THREAD_exT.slurm

To edit type **i** and copy figure 17 exactly. To save press the escape key and type **:w** followed by enter key and then to exit type **:q** followed by enter key.

Before we can compile Armadillo programs on multiple cores, we have to change its configuration. Navigate to the **armadillo_bits** directory in the **include** directory. In here there is a config.hpp file that we will need to modify. If you're in the **source_code** directory type.

$ cd ..
$ cd Armadillo_Library/include/armadillo_bits
$ vim config.hpp

Scroll down until you see the section of code shown in figure 18. Note you can only use arrow keys to scroll down.

```
#if !defined(ARMA_OPENMP_THREADS)
  #define ARMA_OPENMP_THREADS 8
#endif
//// The maximum number of threads to use for OpenMP based parallelisation;
//// it must be an integer that is at least 1.
```

Figure 18 - OpenMP Thread Count Variable in config.hpp

The ARMA_OPENMP_THREADS value defines the max number of threads available for OpenMP to use. From this constraint, OpenMP determines the optimal number of threads to run on. Thus, we can simply set it to 128, but I recommend setting it to the number of threads you plan on allocating to Lonestar6 to ensure accurate date. For example, if we submit a job using 4 threads then set ARMA_OPENMP_THREADS to 4. For this example we will use 2 threads so set the value to 2. To edit and save follow the same procedure as before. This file will have to be edited in this manner every time you want to compile with a new number of threads.

For large input size it's also good to uncomment DEFINE ARMA_64BIT_WORD as shown below.

```
#if !defined(ARMA_64BIT_WORD)
  #define ARMA_64BIT_WORD
//// Uncomment the above line if you req
//// Note that ARMA_64BIT_WORD is automa
```

Figure 19 - ARMA_64BIT_WORD Variable in config.hpp

Now navigate back to **source_code** directory and to compile using openMP type the following. Once again this is one command, and you must replace **$ARMA_ROOT.**

$ g++ Lyap.cpp CCAMA.cpp run_CCAMA.cpp -o main_N_10_THREAD_2 -std=c++11 -O3 -I$ARMA_ROOT/include -L$ARMA_ROOT/OpenBLAS_Library/lib -Wl,-rpath,$ARMA_ROOT/OpenBLAS_Library/lib -lopenblas -fopenmp

Notice how I changed the name of the executable to main_N_10_THREAD_2. This is an example using 2 threads. This is how I organized my files and I recommend you do so as the slurm file is set up to execute a file in that format. Also notice that -fopenmp is how we compile using openMP.

Next copy the executable and the template slurm file to the **$SCRATCH** directory.

$ cp main_N_10_THREAD_2 RUN_CCAMA_N_exN_THREAD_exT.slurm ${SCRATCH}

Now change directories to the **$SCRATCH** directory.

$ cds

In this directory you may organize the files as you see fit so long as the slurm file corresponding to the executable its submitting and the executable are in the same directory. Now we must edit the template slum file we compiled to have it submit the executable we generated.

$ vim RUN_CCAMA_N_exN_THREAD_exT.slurm

Edit it to match figure 20 except use your email and not mine. The export OMP_NUM_THREADS command tells it how many threads to allocate. For a single core job this line would not be needed.

```
#!/bin/bash

#SBATCH -J CCAMA_N_10_2                          # Job name
#SBATCH -o CCAMA_N_10_2.o%j                      # std output file
#SBATCH -e CCAMA_N_10_2.e%j                      # std error file
#SBATCH -p development                           # queue name
#SBATCH -N 1                                     # Number of nodes
#SBATCH -n 1                                     # Number of mpi tasks
#SBATCH -t 02:00:00                              # Run time
#SBATCH --mail-type=all                          # Send email at start and end of job
#SBATCH --mail-user=jac170004@utdallas.edu       # Email address

module list
pwd
date

export OMP_NUM_THREADS=2

./main_N_10_THREAD_2
~
```

Figure 20 - slurm File for N=10 and 2 Threads

Note that you don't need to delete the comments giving instructions, so long as you replace all the instances of exN and exT with 10 and 2 respectively for this case. Edit, save and quit as usual. Now we must change the file name.

$ mv RUN_CCAMA_N_exN_THREAD_exT.slurm RUN_CCAMA_N_10_THREAD_2.slurm

Now we can submit the job

$ sbatch RUN_CCAMA_N_10_THREAD_2.slurm

To check job status

$ squeue -u [tacc username goes here]

You should get an email for when the job starts and is finished. Once it has finished an output and error file will be generated. Your directory should now look like the figure below.

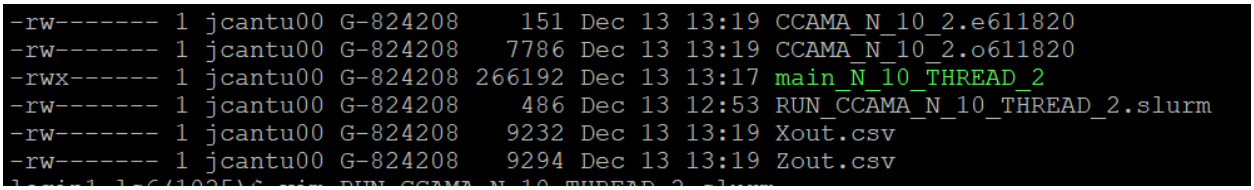

Figure 21 - Scratch directory after job completed

Here we can see we have the output and error files that were generated. The executable and slurm files we copied over are here as well and lastly the csv files the run_CCAMA generates are here as well.

I strongly recommend creating a directory structure in **$SCRATCH** directory to organize slurms and executables. I organized mine by thread count and then by N size. To return to the **$HOME** directory

$cdh

In essence to submit a multithreaded job:

1. Edit the config.hpp file to change ARMA_OPENMP_THREADS to your desire thread count
2. Compile CCAMA project using the -fopenmp and name the executable in the format main_N_exN_THREAD_exT
3. Copy the executable and the template slurm file, RUN_CCAMA_N_exN_THREAD_exT.slurm, to the **$SCRATCH** directory
4. Edit the slurm file to match your executable and rename it
5. Submit the job using the sbatch command

YouTube Videos for Extra Help:

Debugging using gdb

https://www.youtube.com/watch?v=Dq8l1_-QgAc&t=127s

https://www.youtube.com/watch?v=3T3ZDquDDVg

Installing Armadillo on Windows for Microsoft Visual Studio – Follow this guide but tailor it towards Armadillo

https://www.youtube.com/watch?v=or1dAmUO8k0

Installing OpenBLAS – This is the video I referenced

https://www.youtube.com/watch?v=85hm_kbwOJs