I chose the Mycoplasma Genitalium because it’s the smallest free-living organism that is capable of independent growth and replication. Prof. Kristensen was kind enough to inform me of this, which is beneficial because it limits the amount of information I’m waiting on the HPC for.

To parse through its information for blast I downloaded the faa.gz format. Once again Prof. Kristensen was kind enough to inform the class that this is the easiest format to use for blast search. However all the proteins are saved onto a single file which is not ideal for multithreading on the cluster.

To parallelize the files I used what was recommended in the assignment guide and split the faa.gz file into multiple files containing 10 proteins each using a perl script. These files were saved as “files” with a number between 1-52 and a file extension of .txt tied to the end of it. An example file name would be “files1.txt”. With multiple files, my program is now parallelized and able to run chunks at a time individually through the high performance computer. To call the perl script locally I used the call “perl faaSeperator.pl” ‘protein.faa.gz’ file.

My program “FinalProject.job” ran on neon using 4 cores to rotate through my files and blast search each protein located inside of the separate “files”. To run my “FinalProject.job” on the neon cluster I compressed all my files that were locally separated on a linux machine along with the FinalProject.job that had been written locally into a folder. I sent it to the neon cluster using the command “scp –P 40 jmewert\_1.tar.gz neon.hpc.uiowa.edu: .”. Then unzipped the compressed file on neon using the command “tar xvzf jmewert\_1.tar.gz” which uncompressed it into my private directory. To send the actually call to neon I used the command “qsub –t 1:52 FinalProject.job” into neon.

My FinalProject.job defined the blast search that occurred on neon. It used refseq\_protein database as that’s where the file format of my proteins came from. The faa.gz file was found in ncbi under assembly for proteins in reseq database. Therefore refseq seemed like the logical database to use. I used the commands “qseqid sseqid evalue length bitscore stitle” to get Query Seq-ID, Subject Seq-ID, e-Value, Alignment Length, and subject title in that order output form the blast search. So I got 6 columns on my outputs that are named “Output1.txt”-“Output52.txt”. Neon didn’t have enough time to run through all my outputs.

After these blast searches were output, I used the perl file outputParserV2.pl to parse through it and organize the information into separate arrays with the indexes being the main feature keeping the files together. From here I used a search function for the largest bit score found in a blast search for each protein in the file. I saved that index which could be used to access the rest of the matching information for that specific blast search. I used bit score as it’s a normalized version of the raw score so it can be compared to other blast searches. The Bit score represented by S’ is calculated by  S’ = (λ × S − lnK)/ ln2. The raw score is S, K is a constant, and lamda is the gumble distribution constant. The higher the bit score the better the match between the two proteins in the blast search. Unfortunately outputParserV2.pl only does a file at a time. To pull the file off of neon to be parsed and broken down I used “scp neon.hpc.uiowa.edu:/Users/jmewert/Output1.txt .”.

To call the outputParserV2.pl I used the call “perl outputParserV2.pl Output1.txt”. After calling this the true output of my code is printed to the screen. It has the highest bit scores for each individual protein along with the sequence IDs of the blast search and the name of the protein match with the highest bit score.

I included an example output of my program as finaloutexample00.txt.