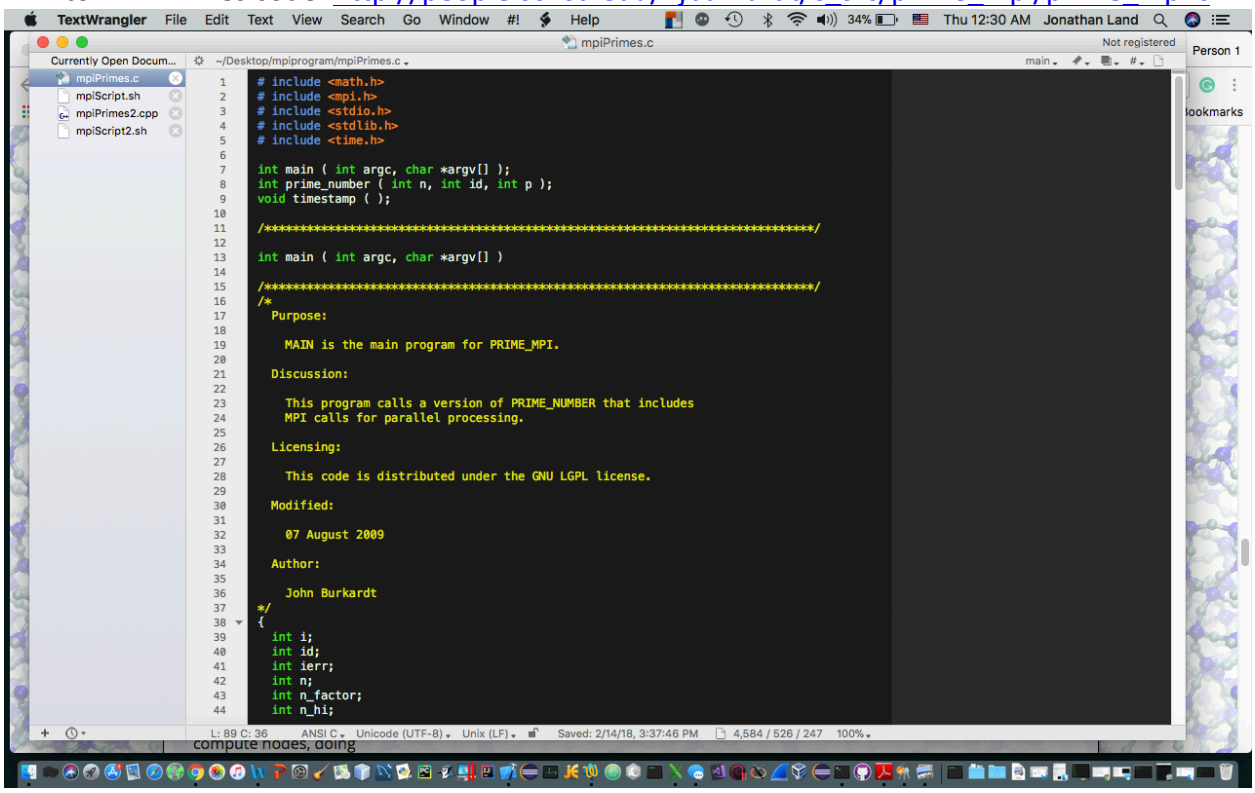
Technical Report: MP3: AWS and Parallel Computing (MPI)

Jonathan Land

Step-by-Step Instructions

1. Find the MPI code you want to use in AWS EC2. In this case, similar to the sample program that was uploaded to UTC Learn by the Professor, our team modified a C program that counts the number of primes between 1 and N, using MPI to carry out the calculation in parallel. Below is a link to this code and a screenshot **(NOTE: Additionally, this code, along with all other source code used in this assignment is listed below, in the last section of this technical report).**

   Link to MPI Primes code: http://people.sc.fsu.edu/~jburkardt/c_src/prime_mpi/prime_mpi.c



2. In order to make sure the code works, it is probably a wise idea to first run your code in Linux/Unix OS before attempting to run the VMs in EC2. To run the code you may need to install mpi, which you can Google to find what command to use to get this on your machine. If you already have MPI, then you will want to save the C file (e.g., mpiPrimes.c), and then compile and run it. First, cd where the file is saved, then type the following command.

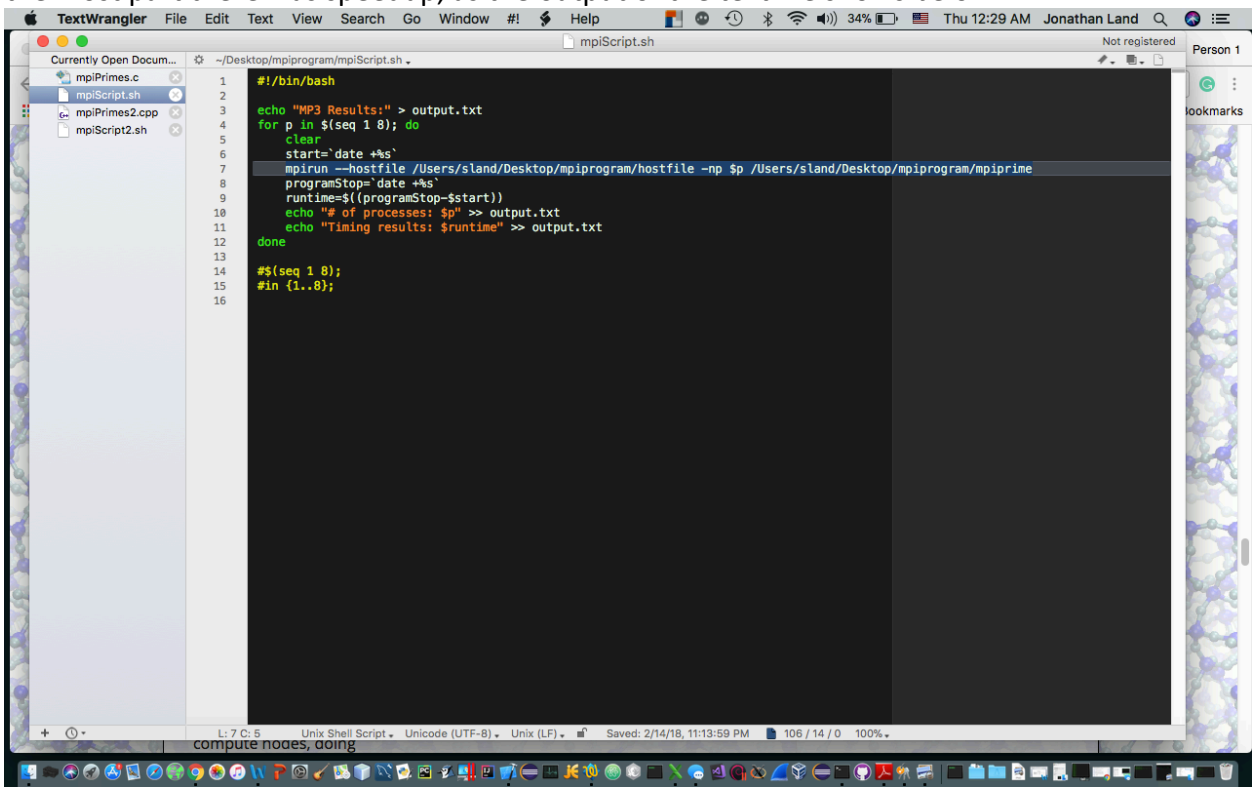
   ```
   mpicc mpiPrimes.c
   ```

   This should produce an executable file, now run the executable: ./a.out.

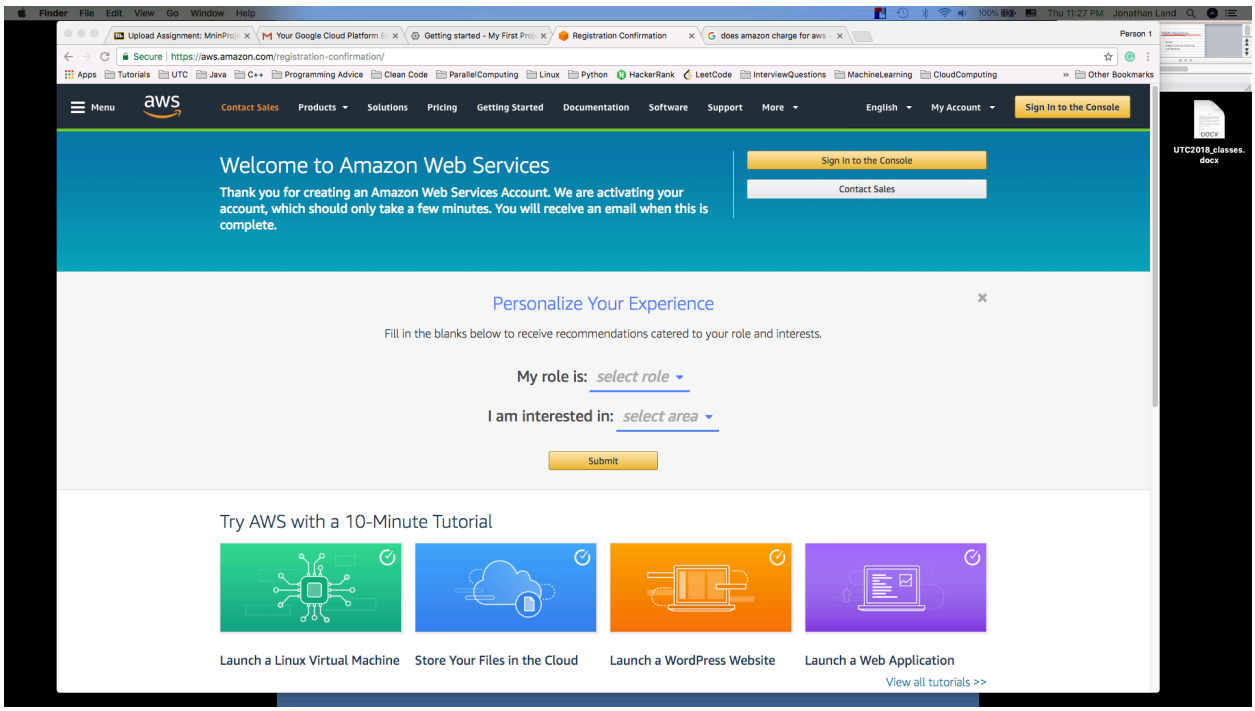
3. After making sure your code is running properly on your machine, create a script file and save it in the same folder as you did your mpiPrimes.c program **(Note: Save in .sh format, as shown below).** Also create a hostfile with the following statement in it and save it to your mpi folder: `localhost slots = 25`.

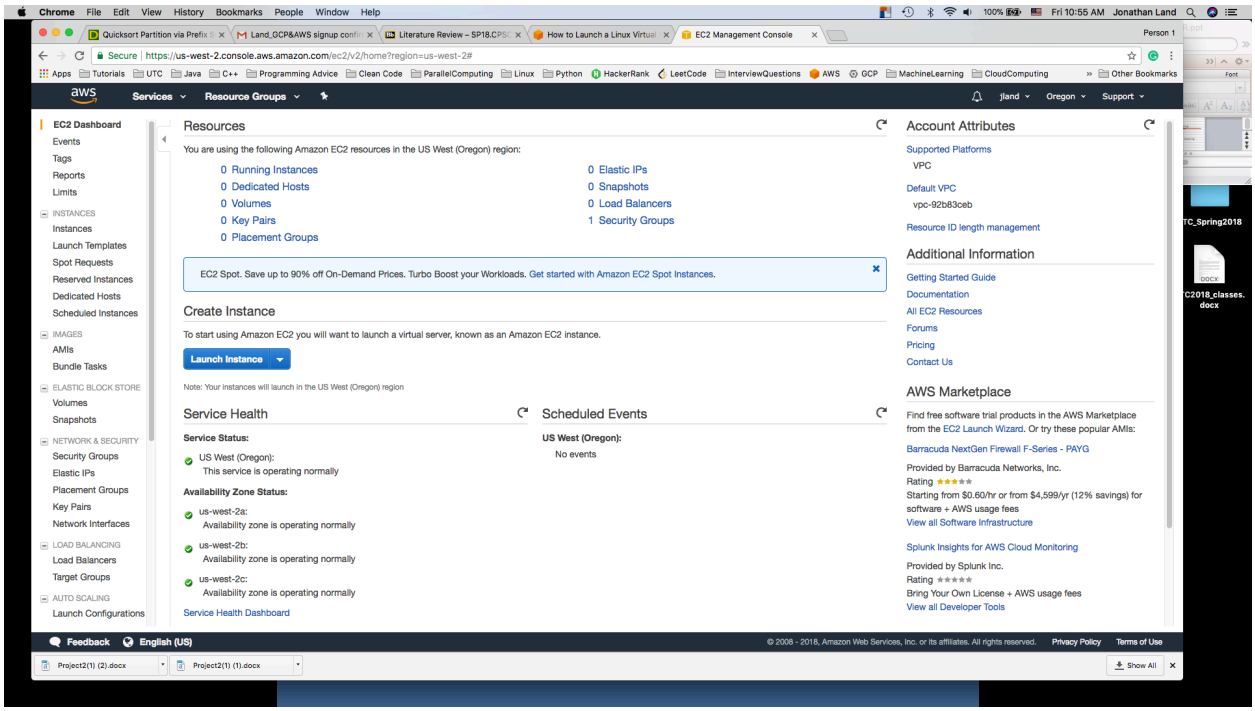
   The logic behind this idea is as follows: Since, in this project, we were supposed to run the code on multiple VMs, the script file and hostfile were created in order to loop through the different IP addresses of the VMs in EC2 that we create (our instances). You can create this script file and run it in on your machine to test this first. This can be run as many times you designate in the loop, as shown in the first screenshot below. That said, after pushing these files to EC2, you will have to *change* both the script file and the hostfile using vi, or whatever text you like, when you ssh into EC2 to include the nodes and IP addresses (this will be discussed in further detail later). If you change this before, the code will not run properly, because it will not find what you are telling the program to look for.

   The following screenshots are examples of utilizing the script file, which uses mpirun and other MPI methods to run the C code in parallel multiple times, along with printing the timing results to a text file. In terms of our tests, the timing results were sometimes erratic, but for the most part there was speedup, as the output of the text file shows below.



4. After making sure your program and script file work, create your instances on EC2. Create an account and sign in to your console.





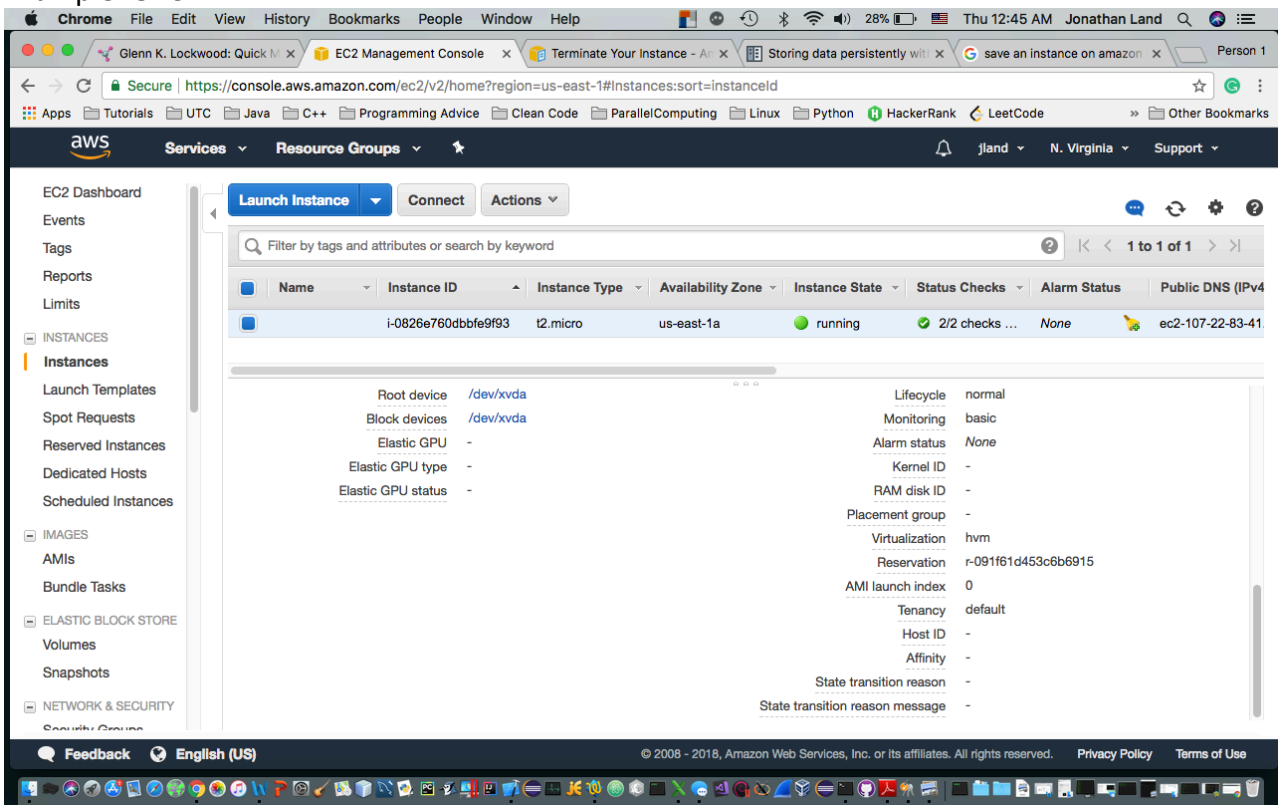
5. As you can see, there are no VMs in EC2 now so you will have to create them. In order to do this, go to the following website and follow the instructions step-by-step, depending on your OS, to create your first VM. After this you can create a copy/image of the one and go from there, but you will need this first one to be your main VM.
   Link to "Launch a Linux Virtual Machine with Amazon EC2": https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/

6. When you are done, you can then ssh into EC2 like this, using the following command: `ssh -i ~/.ssh/MyKeyPair.pem ec2-user@<Specific IP Address goes here>` (NOTE: this is the main IP that wil be used – you will create other IPs to loop through).

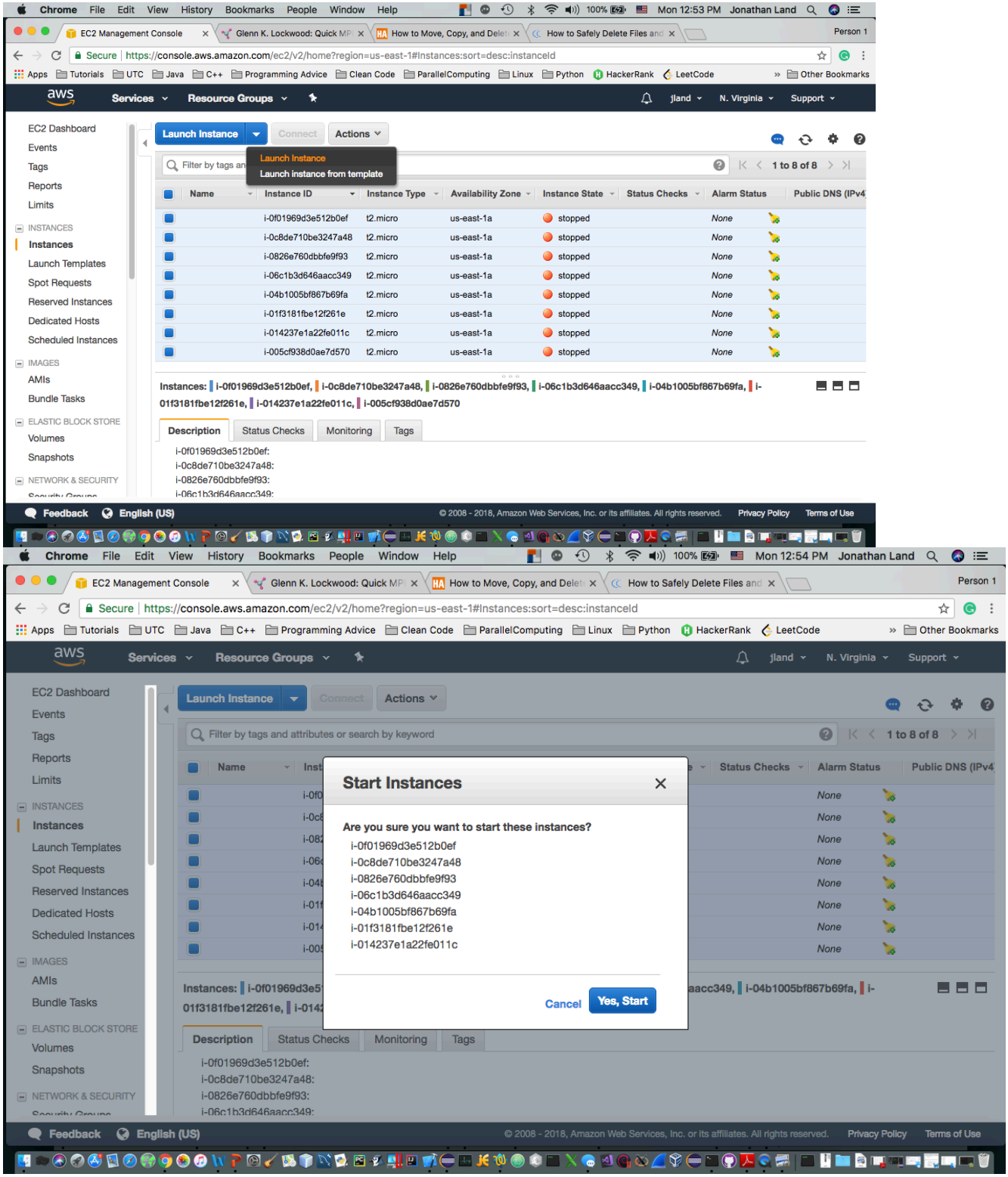
7. Now we know we can gain access to EC2 and we have created our first instance called `MyKeyPair` (or whatever else you called it). Practice exiting out of EC2 and ssh backing into it. Each time you will use that same ssh command you have saved in Linux command cache.

8. You now need to create multiple VMs with different IP addresses for each. To do this first run the first instance. Go to "Launch Instance" and click "Launch Instance." After it starts running, now create an image of the first VM. Click on "Actions" and create a copy/image and add this to the VM pool.

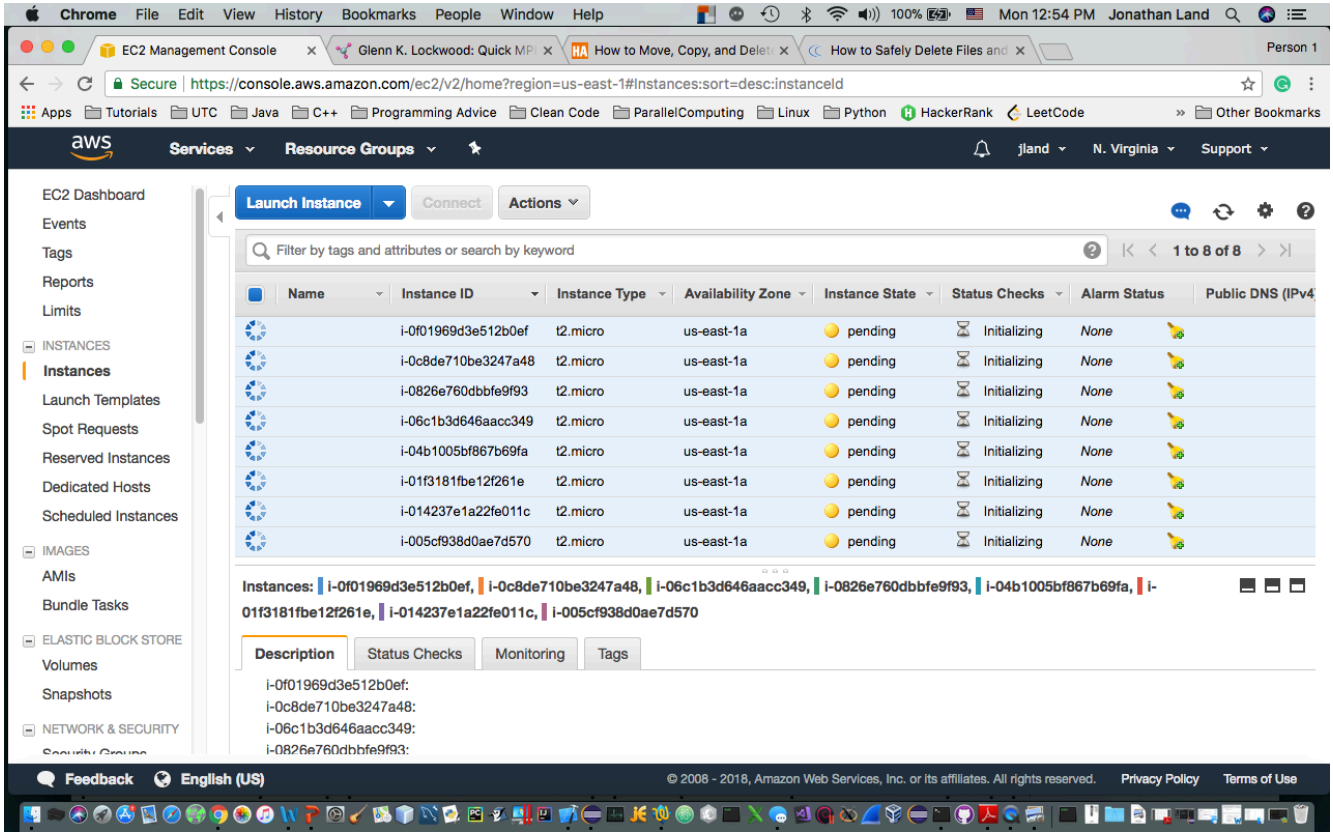
   Example: One VM



   Example: After adding 8 VMs and running them (run all instances [click on "Launch Instance" – you change the Instance IDs]).



It may take a few minutes to launch all instances.



Alright, so you now know your code works in Linux/Unix, you have created multiple VMs in EC2, and you can exit out of EC2 and gain access again using the ssh command listed above, which is saved in the command cache of your machine. Now, you need to push all the files to EC2 and modify them to run your code in EC2.

9. Push your files from the host computer to EC2.
    a. Make sure you are logged out of EC2. After logging out, you are now going to ssh back in, but using the `sftp` command to push your files. So, use the following command:
    `sftp -i ~/.ssh/MyKeyPair.pem ec2-user@<Main IP address here>`

    b. Now push the files, using the `put` command. In the example below, you can see where I first used `ls` to see all files I needed to transfer, then I used `put` to transfer those files. Each time, the OS tells you whether or not those files have successfully loaded to EC2 or not, which is helpful.

```
put mpscript.sh
put mpiprime
put hostfile
put mpiPrimes.c
put output.txt
```

c. After all of your files are transferred, type “quit”.

d. Now ssh back in (the normal way without the sftp).
```
ssh -i ~/.ssh/MyKeyPair.pem <key pair here>
```

e. Now type in `ls` to make sure your files are there…they should be.

f. And to double-check that the correct code is within those files, you can `cat` each one.

10. The issue you need to take care of now is changing the hostfile to include the IP addresses of your VMs so that when the script file runs, it will be looping through those particular VMs. You do not need to even designate how many, it will just continually add them and run them. To do this, you need to change your hostfile and script file, because right now it does not have the other IP addresses and contains the data that only runs on your host computer. In order to do this, complete the following:

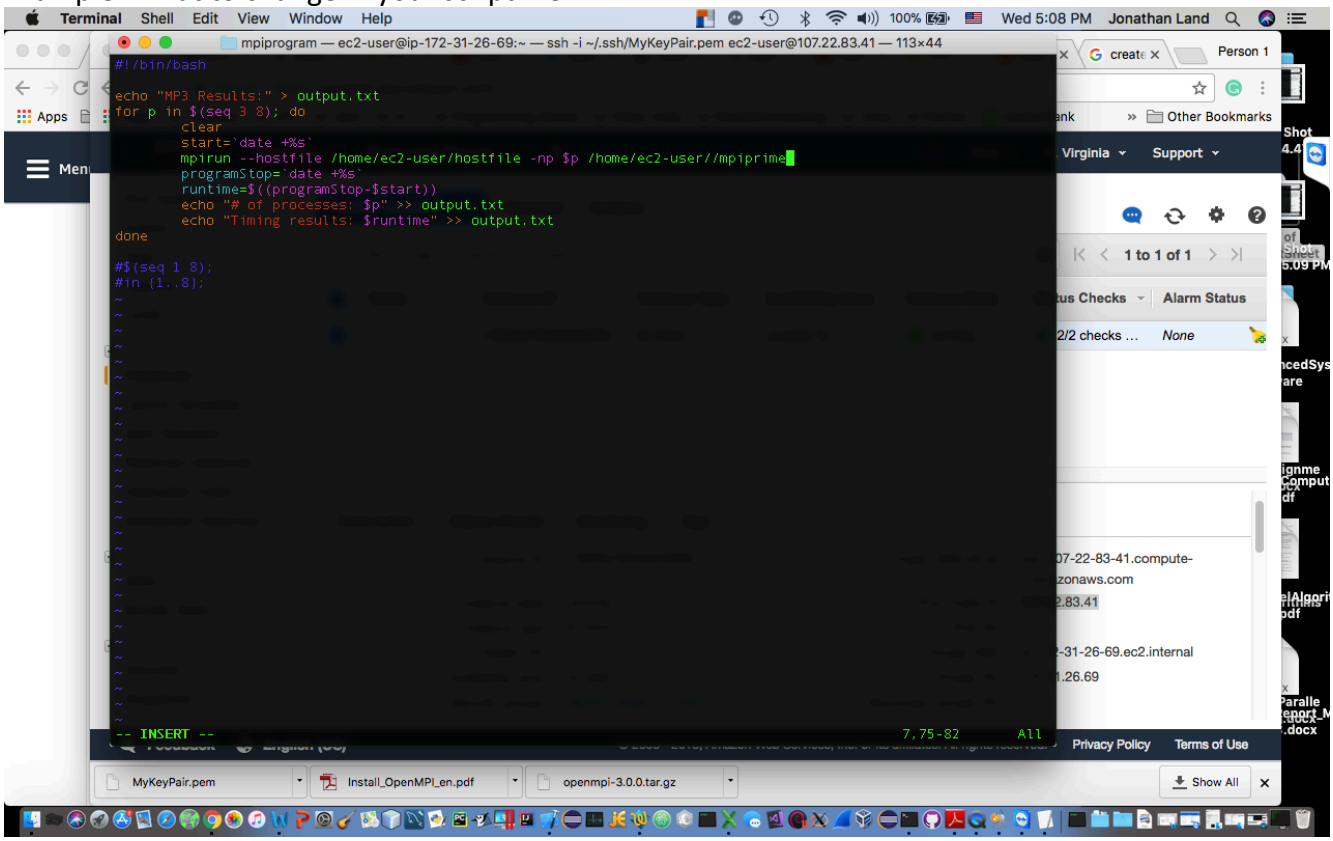
    a. *Changing the host file:* Go back to your AWS instances. You will need to copy and past *each* of the IP addresses *and* instance IDs for the particular VMs *into* the hostfile. To do this, you will have to edit the file in EC2. Open up a text editor. I used vi, but you can use whichever you want. To open the hostfile in vi, type `vi hostfile`. To edit in vi, type i to insert code. Now, type the name and IP address of each of your VMs into the file. When done inserting, save the file using `:wq` to write, quit and go back to command line. See the example below of how to include name and IP address. Obviously your IP addresses will differ from those below.

```
<Instance Id name here> <IP address here>
<Instance Id name here> 122.223.344.34
<Instance Id name here> 564.43.45.4454
<Instance Id name here> 111.22.344.455
<Instance Id name here> 775.34.44.4455
<Instance Id name here> 44.55.44.445.66
<Instance Id name here> 45.50.123.44.62
<Instance Id name here> 44.55.44.445.66
```

    b. *Changing the script file:* You now need to change the script file to link with your EC2 instances, instead of your home folder on your host computer. I am using vi again do this. Since my script file is called, mpiScript.sh, then you can open this for editing in vi using, `vi mpiScript.sh`. You now want to delete and change the part that is associated with the mpirun command in your code (use the commands above for vi to save and exit after you are done or it will not save what you do).

(**NOTE:** Change the host file name to what you saved it as. In the description above, "hostfile" is my host file and "mpiprime" is the name of the executable file that is processing all the IP address information from the hostfile to make it run).

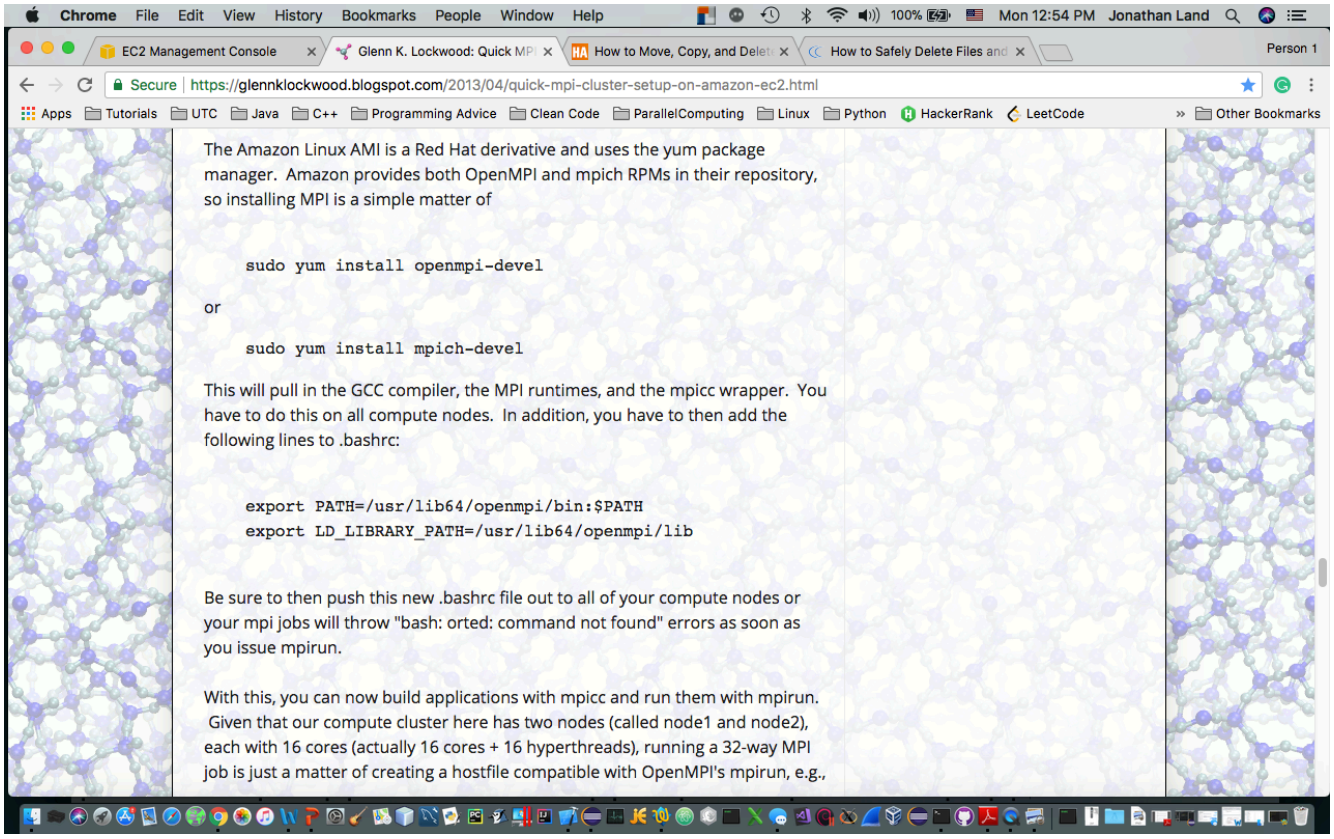
Example: What to change in your script file



11. Now that your host file and script file are changed, you are almost ready to run your code in EC2 VMs. If you ran your code now as it is, you will likely get the following error message. The following explains how to fix this.

    a. The first error message you may encounter is, "mpirun command not found." This is because you have not install MPI onto the VM, so we have to do this.

        i. To install MPI on run your code, go the following website: https://glennklockwood.blogspot.com/2013/04/quick-mpi-cluster-setup-on-amazon-ec2.html
        ii. Scroll down until you get to the following section. Copy and past each piece of code from this section into the command line. This will install MPI and run the code on all nodes.



The Amazon Linux AMI is a Red Hat derivative and uses the yum package manager. Amazon provides both OpenMPI and mpich RPMs in their repository, so installing MPI is a simple matter of

```
sudo yum install openmpi-devel
```

or

```
sudo yum install mpich-devel
```

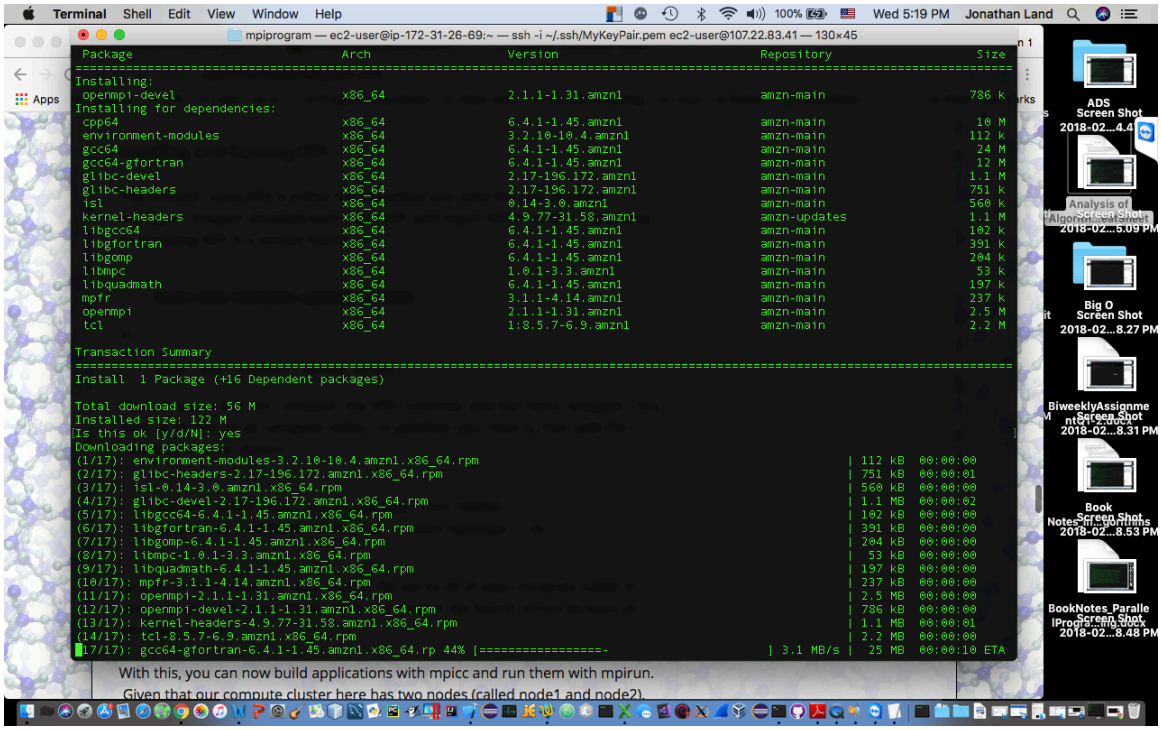
This will pull in the GCC compiler, the MPI runtimes, and the mpicc wrapper. You have to do this on all compute nodes. In addition, you have to then add the following lines to .bashrc:

```
export PATH=/usr/lib64/openmpi/bin:$PATH
export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib
```

Be sure to then push this new .bashrc file out to all of your compute nodes or your mpi jobs will throw "bash: orted: command not found" errors as soon as you issue mpirun.

With this, you can now build applications with mpicc and run them with mpirun. Given that our compute cluster here has two nodes (called node1 and node2), each with 16 cores (actually 16 cores + 16 hyperthreads), running a 32-way MPI job is just a matter of creating a hostfile compatible with OpenMPI's mpirun, e.g.,

Examples of MPI installing



12. Now try to run your code. First, compile your program by typing the following: `mpicc mpiPrimes.c`. Now that an executable has been created, run the script file which you will use from now on to run your program by typing: `./myScript.sh` (or whatever you named this).

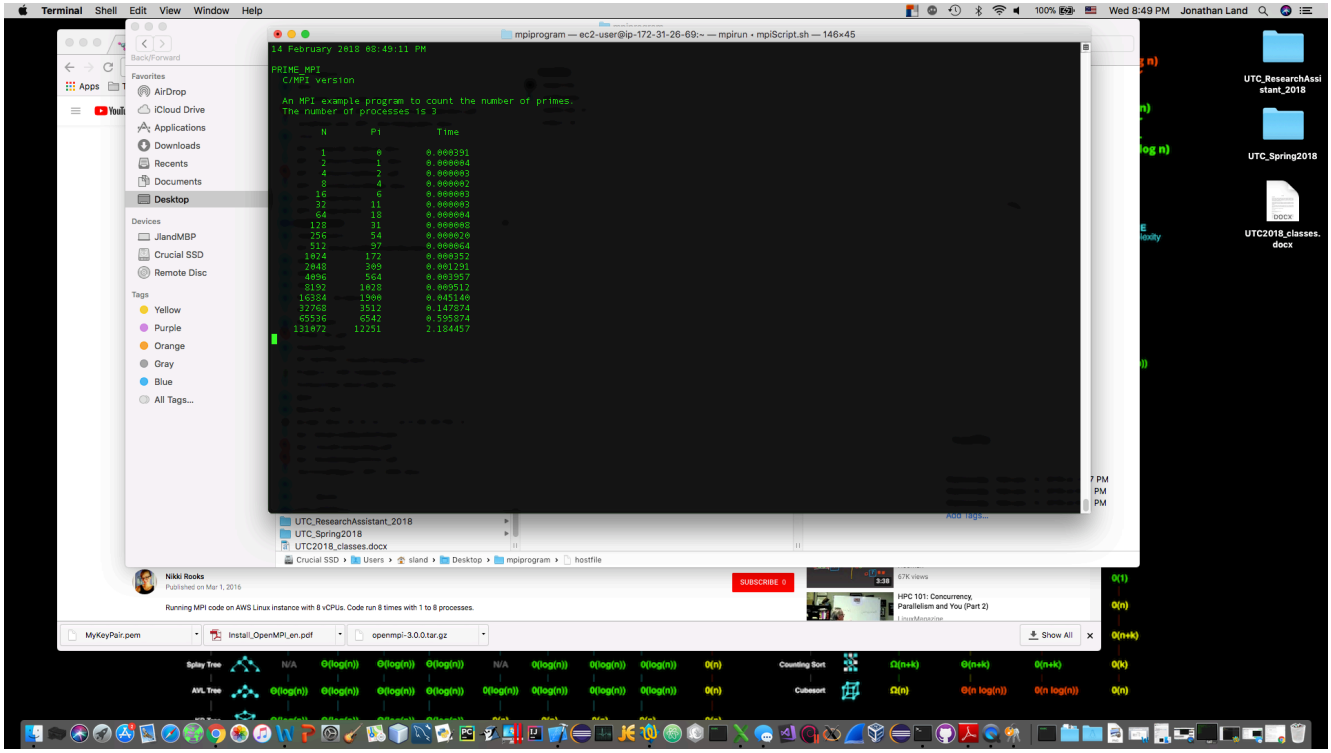
    a. When this runs, the system will likely produce this error message, “Exec format error”

    b. The reason this erred out is because when you compile the program and run the script, it is creating an executable called `./a.out`, but we specified that our executable be called, `mpiprime` in our script file, so we have to change this to match up. In order to do this we are just going to replace the previous `mpiprime` executable with the new `./a.out` while also changing the name of `./a.out` to `mpiprime`. To do so, type the following command:

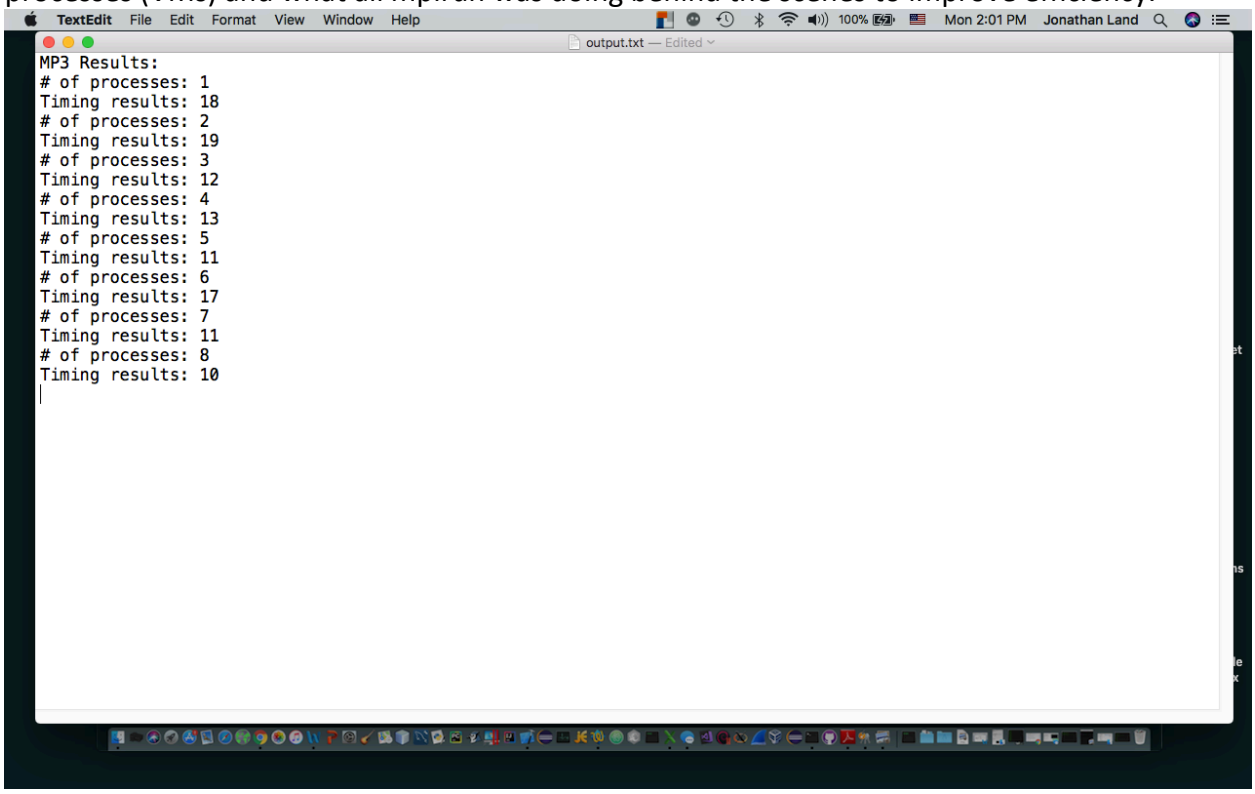
    ```
    mv ./a.out mpiprime
    ```

    c. Type `ls` to make sure the `mpiprime` file is still there, but `./a.out` is gone.

13. Now run `./myScript.sh`. The program should start and should go through each VM keeping track of the timings, which are put in the output.txt file (See screencast for complete running of the program).



In this timing example, the timing decreased from 18 to 10 seconds, depending on the # of processes (VMs) and what all mpirun was doing behind the scenes to improve efficiency.



Source Code

Parallel MPI code: mpiPrimes.c

```
# include <math.h>
# include <mpi.h>
# include <stdio.h>
# include <stdlib.h>
# include <time.h>

int main ( int argc, char *argv[] );
int prime_number ( int n, int id, int p );
void timestamp ( );

/******************************************************************************/

int main ( int argc, char *argv[] )

/******************************************************************************/
/*
  Purpose:

    MAIN is the main program for PRIME_MPI.

  Discussion:

    This program calls a version of PRIME_NUMBER that includes
    MPI calls for parallel processing.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    07 August 2009

  Author:

    John Burkardt
*/
{
  int i;
  int id;
  int ierr;
  int n;
  int n_factor;
  int n_hi;
```

```
  int n_lo;
  int p;
  int primes;
  int primes_part;
  double wtime;

  n_lo = 1;
  n_hi = 262144;
  n_factor = 2;
/*
  Initialize MPI.
*/
  ierr = MPI_Init ( &argc, &argv );
/*
  Get the number of processes.
*/
  ierr = MPI_Comm_size ( MPI_COMM_WORLD, &p );
/*
  Determine this processes's rank.
*/
  ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &id );

  if ( id == 0 )
  {
    timestamp ( );
    printf ( "\n" );
    printf ( "PRIME_MPI\n" );
    printf ( "  C/MPI version\n" );
    printf ( "\n" );
    printf ( "  An MPI example program to count the number of primes.\n" );
    printf ( "  The number of processes is %d\n", p );
    printf ( "\n" );
    printf ( "         N        Pi          Time\n" );
    printf ( "\n" );
  }

  n = n_lo;

  while ( n <= n_hi )
  {
    if ( id == 0 )
    {
      wtime = MPI_Wtime ( );
    }
    ierr = MPI_Bcast ( &n, 1, MPI_INT, 0, MPI_COMM_WORLD );

    primes_part = prime_number ( n, id, p );
```

```
    ierr = MPI_Reduce ( &primes_part, &primes, 1, MPI_INT, MPI_SUM, 0,
      MPI_COMM_WORLD );

    if ( id == 0 )
    {
      wtime = MPI_Wtime ( ) - wtime;
      printf ( "  %8d  %8d  %14f\n", n, primes, wtime );
    }
    n = n * n_factor;
  }
/*
  Terminate MPI.
*/
  ierr = MPI_Finalize ( );
/*
  Terminate.
*/
  if ( id == 0 )
  {
    printf ( "\n");
    printf ( "PRIME_MPI - Master process:\n");
    printf ( "  Normal end of execution.\n");
    printf ( "\n" );
    timestamp ( );
  }

  return 0;
}
/******************************************************************************/

int prime_number ( int n, int id, int p )

/******************************************************************************/
/*
  Purpose:

    PRIME_NUMBER returns the number of primes between 1 and N.

  Discussion:

    In order to divide the work up evenly among P processors, processor
    ID starts at 2+ID and skips by P.

    A naive algorithm is used.

    Mathematica can return the number of primes less than or equal to N
    by the command PrimePi[N].
```

```
                N  PRIME_NUMBER

                1           0
               10           4
              100          25
            1,000         168
           10,000       1,229
          100,000       9,592
        1,000,000      78,498
       10,000,000     664,579
      100,000,000   5,761,455
    1,000,000,000  50,847,534

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    21 May 2009

  Author:

    John Burkardt

  Parameters:

    Input, int N, the maximum number to check.

    Input, int ID, the ID of this process,
    between 0 and P-1.

    Input, int P, the number of processes.

    Output, int PRIME_NUMBER, the number of prime numbers up to N.
*/
{
  int i;
  int j;
  int prime;
  int total;

  total = 0;

  for ( i = 2 + id; i <= n; i = i + p )
  {
    prime = 1;
    for ( j = 2; j < i; j++ )
```

```
    {
      if ( ( i % j ) == 0 )
      {
        prime = 0;
        break;
      }
    }
    total = total + prime;
  }
  return total;
}
/******************************************************************************/

void timestamp ( void )

/******************************************************************************/
/*
  Purpose:

    TIMESTAMP prints the current YMDHMS date as a time stamp.

  Example:

    31 May 2001 09:45:54 AM

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    24 September 2003

  Author:

    John Burkardt

  Parameters:

    None
*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;
```

```
  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  printf ( "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```

Script file: mpiScript.sh (host computer)

```
#!/bin/bash
echo "MP3 Results:" > output.txt
for p in $(seq 1 8); do
     clear
     start=`date +%s`
     mpirun --hostfile /Users/developerland/Desktop/mpiprogram/hostfile -np
$p /Users/developerland/Desktop/mpiprogram/mpiprime
     programStop=`date +%s`
     runtime=$((programStop-$start))
     echo "# of processes: $p" >> output.txt
     echo "Timing results: $runtime" >> output.txt
done
```

Script file: mpiScript.sh (ssh or VM/EC2 side)

```
#!/bin/bash
echo "MP3 Results:" > output.txt
for p in $(seq 1 8); do
     clear
     start=`date +%s`
     mpirun --hostfile /home/ec2-user/hostfile -np $p /home/ec2-
user/mpiprime
     programStop=`date +%s`
     runtime=$((programStop-$start))
     echo "# of processes: $p" >> output.txt
     echo "Timing results: $runtime" >> output.txt
done
```

Host file: hostfile (host computer)

```
localhost slots = 25
```

Host file: hostfile (ssh or VM/EC2 side)

```
localhost slots = 25
<Instance Id name here> <IP address here>

main <Main VM IP address>
node2 <VM IP address>
node3 <VM IP address>
nodd4 <VM IP address>
node5 <VM IP address>
node6 <VM IP address>
node7 <VM IP address>
node7 <VM IP address>
```

Output.txt

```
MP3 Results:
# of processes: 1
Timing results: 18
# of processes: 2
Timing results: 19
# of processes: 3
Timing results: 12
# of processes: 4
Timing results: 13
# of processes: 5
Timing results: 11
# of processes: 6
Timing results: 17
# of processes: 7
Timing results: 11
# of processes: 8
Timing results: 10
```