Developing Soft and Parallel Programming Skills Using Project-Based Learning

Justin Choi, Adam Henrie, Jean-Pierre Sacha, Titilayo Shonuyi, Shanza Siddiqi

Spring 2019
Group Name: Qubits

Planning and Scheduling

Team name: qubits

Name	Email	Task	Duration	Dependency	Due Date	Note
Justin Choi	jchoi34@student.gsu.edu	Report and help with parallel questions	3 hours	Task 3 and Task 4	4/14/19	Finish report a day before the due date
Adam Henrie	ahenrie1@student.gsu.edu	Video, parallel programming	4 hours	Video	4/12/19	Please go over how the programming works
JP Sacha	jsacha2@student.gsu.edu	Parallel questions and report	4 hours	Task 3 and Task 4	4/14/19	Assist with questions and report
Titilayo Shonuyi	tshonuyi1@student.gsu.edu	Coordinator and help with parallel programming	4 hours	None	4/12/19	Please remember to submit the report by Monday midnight
Shanza Siddiqi	Ssiddiqi2@student.gsu.edu	Create To do/In Progress/Done columns and cards	3 hours	None	4/7/19	Create new cards on github and provide screenshots of the tasks

Parallel Programming Skills

Foundation

1. **(15p) What are the basic steps (show all steps) in building a parallel program? Show at least one example.**
   To build a parallel program, you must first identify sets of tasks and/or partitions of data that can be processed concurrently. If the data can be split evenly into parts, a parallel solution is possible, although sometimes it is impossible to do so.

   An example for building a parallel program would be taking a very large array and being able to evenly divide it into multiple smaller arrays while being able to independently compute the processing without any communication or dependency between the tasks. Using the master/worker implementation technique, the master initializes the array and divides it among the available workers. It then receives the results from each worker. The worker receives its portion of the array from the master, processes the array, and then returns the results back to the master. The master/worker model implements static load balancing since all of the tasks are performing the same amount of work across several identical machines.

2. **(5p) What is MapReduce?** A Hadoop method developed by Google that combines map and reduce clauses to process large quantities of data

3. **(10p) What is map and what is reduce?** A map takes a set of data and assigns a given function to each value, while a reduce joins the set while performing some binary operation

4. **(5p) Why MapReduce?** The combination of map and reduce allows data to be processed in parallel and the processed output to be reduced, possibly simplifying the data

5. **(5p) Show an example for MapReduce.** Checking a database of millions of people's age to indicate if they are eligible to use their parent's health insurance. If 26 or older, they are no longer eligible.

6. **(10p) Explain in your own words how MapReduce model is executed?** MapReduce is executed in two phases, map and reduce. During the first phase, an input pair is taken by the map which produces intermediate key and value pairs. The intermediate values are then paired with the intermediate key they are associated with by the MapReduce library which is then passed to the reduce

function, entering the second phase. The reduce phase accepts the intermediate key along with the values associated with that key and combines the values together to form a smaller dataset.

7. **(6p) List and describe three examples that are expressed as MapReduce computations.**
   Distributed grep: The map function emits a line if it matches a given pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.

   Count of URL Access Frequency: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

   Reverse Web-Link Graph: The map function outputs <target, source> pairs for each link to a target URL found in a page named "source". The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>.

8. **(6p) When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?**
   OpenMP is used when you want to use shared memory parallelism for your code in order to split a task among multiple threads. OpenMP focuses on shared memory paradigms, e.g. multi-core processor. A classic example would be a for loop being parallelized so multiple threads are handling a portion of the iterations of the loop.

   MPI is used mainly for developing parallel scientific applications because scientific applications have code that is tightly synchronous and well load balanced. MPI focuses on message passing paradigms which are a non-homogenous distributed system, e.g. multiple machines.

   Hadoop MapReduce is used if you have a vast amount of data, such as terabytes, and you want to extract, transform and load the data. This is due to its ability to run an operation for each element of data and reduce the results.

9. **(14p) In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**

A drug design and DNA problem is a problem of pharmaceutical companies designing the medicine people use. Due to the fact that DNA houses the blueprints of making proteins in people's bodies, pharmaceutical companies have to figure out how to change a protein's shape because the shape of a protein determines what type of function it performs within the human body. This is done by using ligands to change the shape of a protein. A drug design software that accomplishes these goals goes through several steps. It first generates several ligands to try and mix and match with a specified protein. Then a score is given for each ligand that specifies how well it bonds with the protein and how well it will change the shape of the protein into a desired shape. Lastly, it chooses the ligand with the best score to be synthesized and tested.

Parallel Programming Basics - Drug Design and DNA in Parallel

Compilation process for Project 5 including inserting new TBB library via terminal

Compilation process via the 2019 experimental version of TBB
Only one that I could get to work.

Link for 2019 experimental version
https://packages.debian.org/experimental/armhf/libtbb-dev/download

Next make command was use to compile the OpenMP code

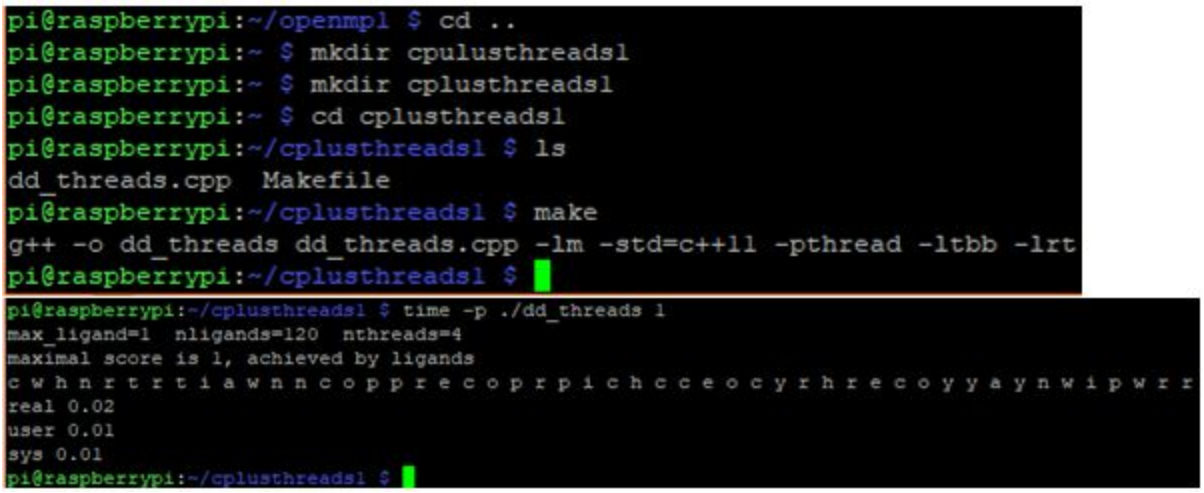
```
pi@raspberrypi:~/Documents $ sudo dpkg -i libtbb-dev_2019_U5-1_expl_armhf.deb
(Reading database ... 142390 files and directories currently installed.)
Preparing to unpack libtbb-dev_2019_U5-1_expl_armhf.deb ...
Unpacking libtbb-dev:armhf (2019~U5-1~expl) over (2018~U6-4) ...
dpkg: dependency problems prevent configuration of libtbb-dev:armhf:
 libtbb-dev:armhf depends on libtbb2 (= 2019~U5-1~expl); however:
  Version of libtbb2:armhf on system is 4.3~20150611-2.

dpkg: error processing package libtbb-dev:armhf (--install):
 dependency problems - leaving unconfigured
Errors were encountered while processing:
 libtbb-dev:armhf
pi@raspberrypi:~/Documents $ cd ..
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ ls
dd_omp.cpp  Makefile
pi@raspberrypi:~/openmp1 $ make
g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/openmp1 $
```

Compilation and run of serial version for ligands.

```
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ ls
dd_serial.cpp  Makefile
pi@raspberrypi:~/sequential $ make
g++ -o dd_serial dd_serial.cpp
pi@raspberrypi:~/sequential $ ls
dd_serial  dd_serial.cpp  Makefile
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 143.67
user 143.61
sys 0.01
pi@raspberrypi:~/sequential $
```

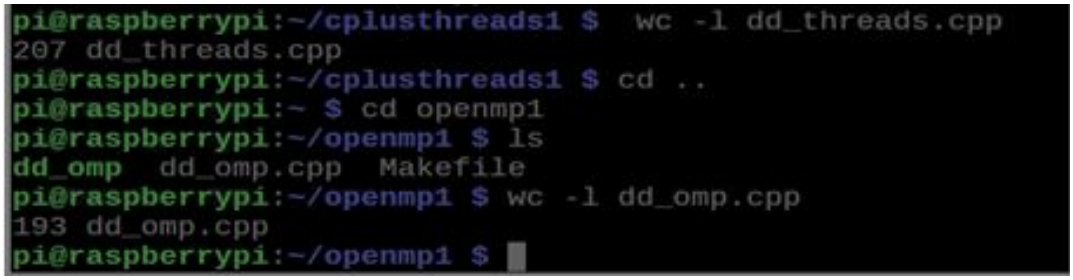
Compilation and run of first Threads code in cplusthreads1 directory.



Questions for last project DNA with ligand comparisons

1. The omp and pThreads implementations are both similar in overall speed in regards to ligand comparisons.

2. We can see that the pThreads implementation is only 14 more words in the code which really equates to only a couple of lines of code.
Pthreads = 207
OpenMP = 193



3. Using a max ligand match length of 4 and number of processing threads as 5. Threads time was 0.21 and OMP code time was 0.21

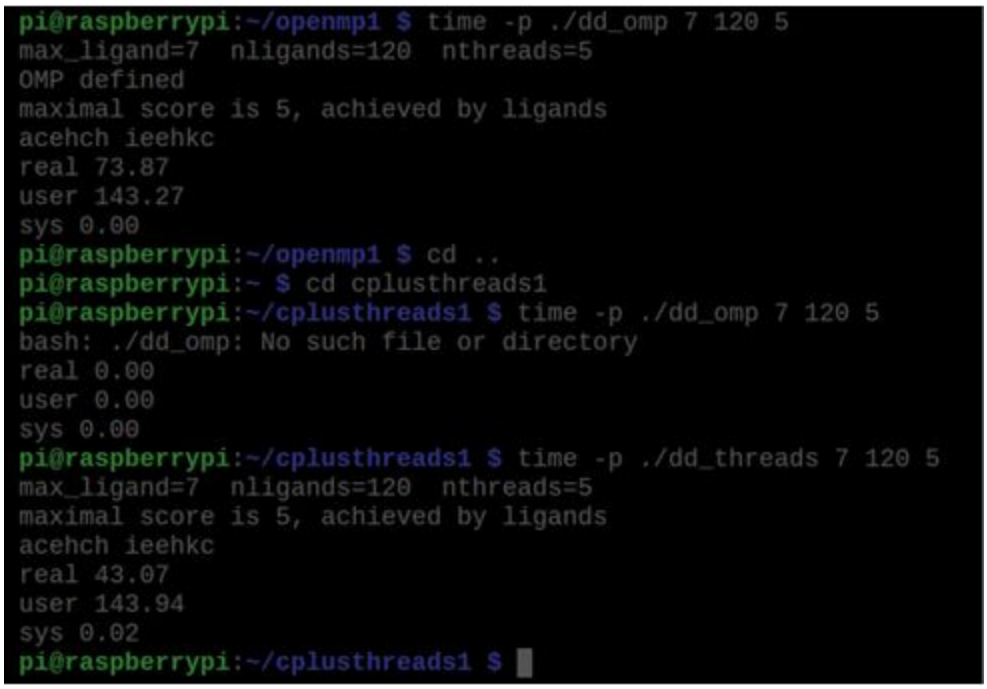
```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 5
max_ligand=4  nligands=120  nthreads=5
maximal score is 3, achieved by ligands
hch ihjo orel rdry hkic wht cio ordt
real 0.21
user 0.61
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ cd ..
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 5
max_ligand=4  nligands=120  nthreads=5
OMP defined
maximal score is 3, achieved by ligands
wht rdry ihjo hkic orel ordt hch cio
real 0.21
user 0.60
sys 0.03
pi@raspberrypi:~/openmp1 $
```

4. Using the number of threads as 4 and the max ligand length of 7 we get OMP time of 80.83 and a Threads time of 42.46.

Max threads 4

```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 80.83
user 143.23
sys 0.02
pi@raspberrypi:~/openmp1 $ cd ..
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 42.46
user 143.83
sys 0.03
pi@raspberrypi:~/cplusthreads1 $
```

Max threads 5 w/ligand 7

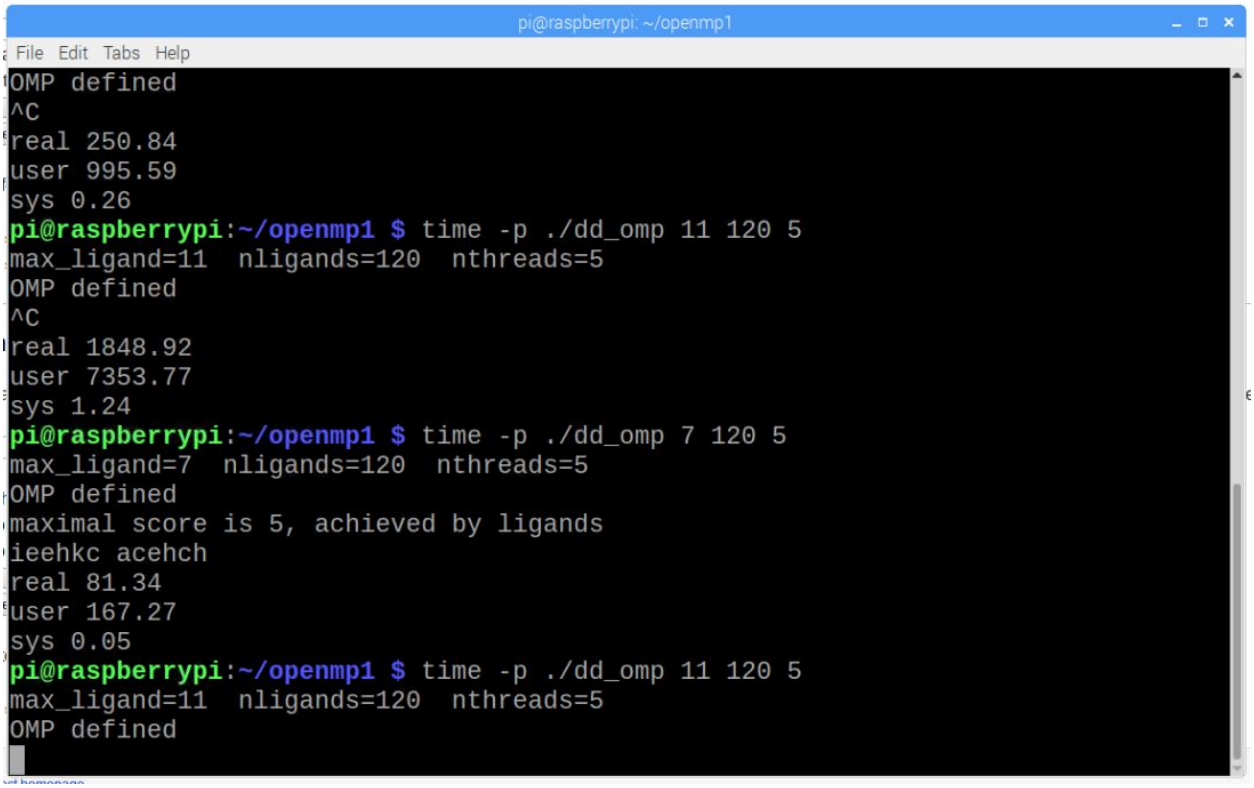


Using the number of threads as 5 and the max ligand length of 7 we get OMP real time of 73.87 and a Threads code real time of 43.07.

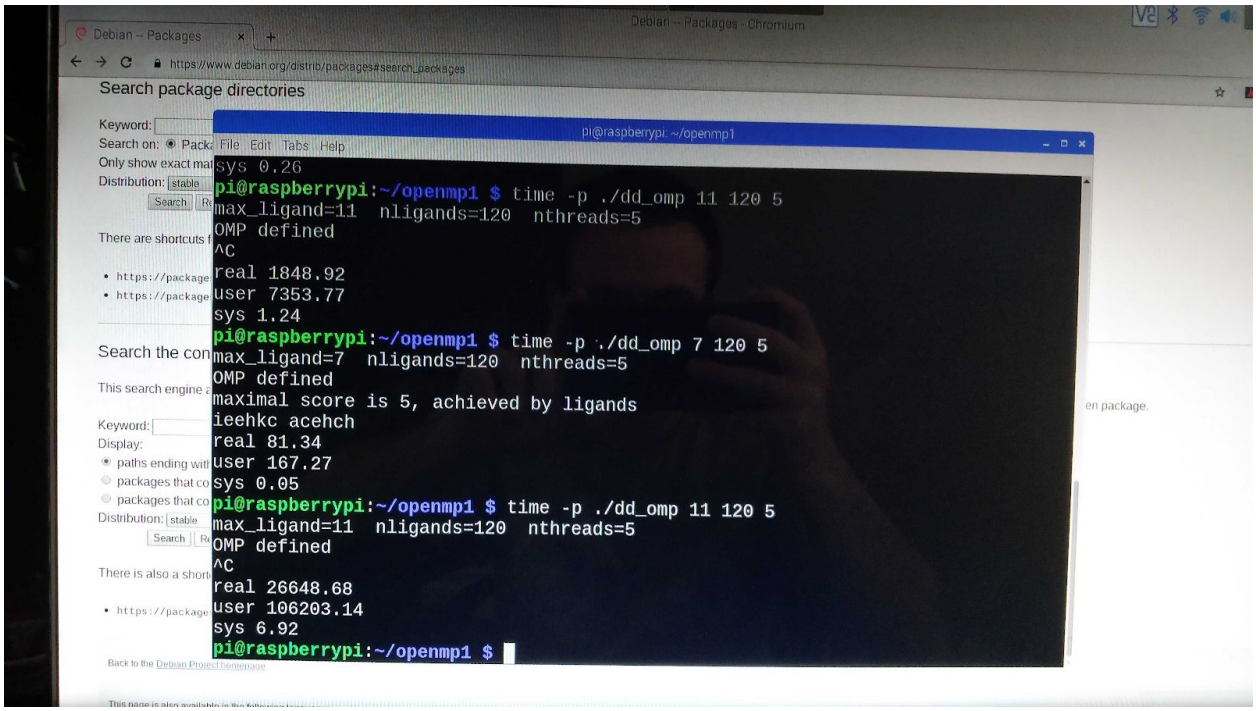
Tables comparing times

Implementation	Time (s) real
dd_serial	143.67
dd_omp 1 ligand	0.03
dd_threads 1 ligand	0.02

Implementation	Time(s) 2 ligands Threads	Time(s) 3 ligands Threads	Time(s) 4 ligands Threads
dd_omp	0.02	0.06	0.25
dd_threads	0.02	0.06	0.21



Omp implementation time here after I cancelled it....

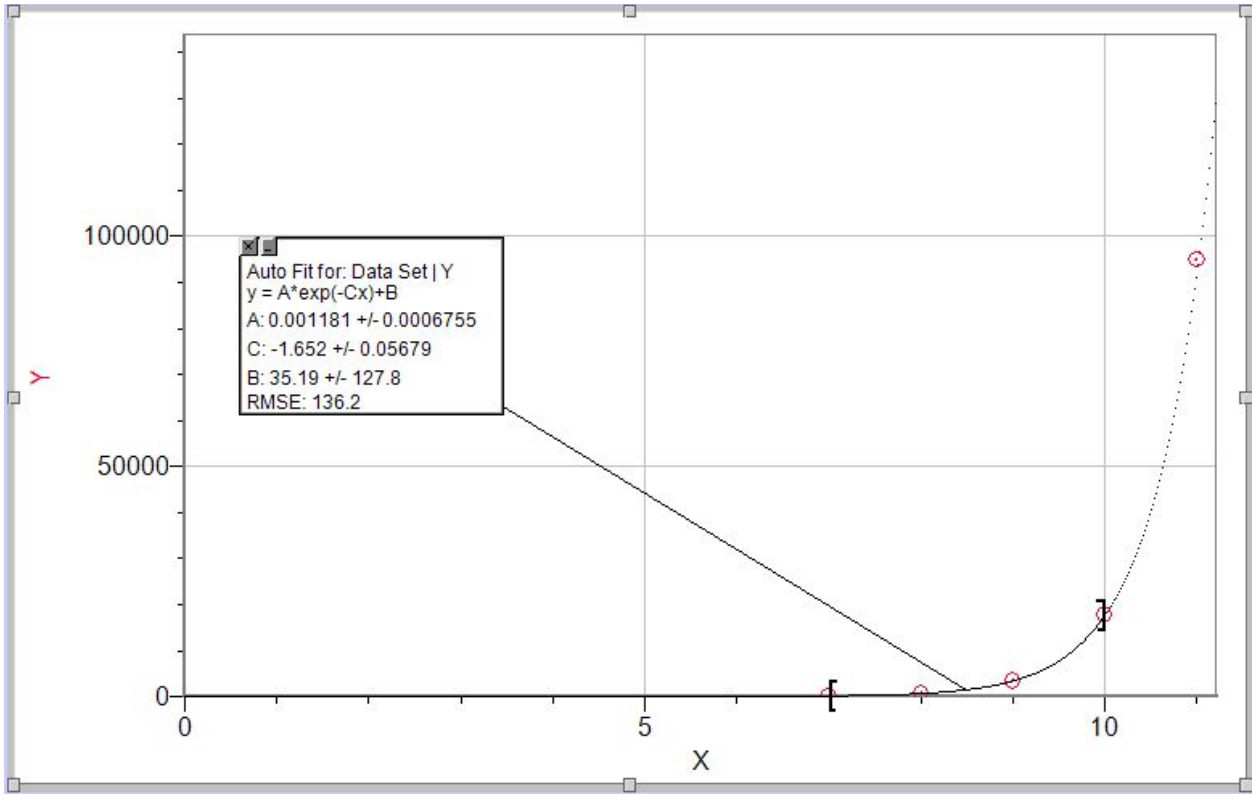


7.4 hours

Time for input of max 10 ligands was 17697 seconds. 4.91 hours.

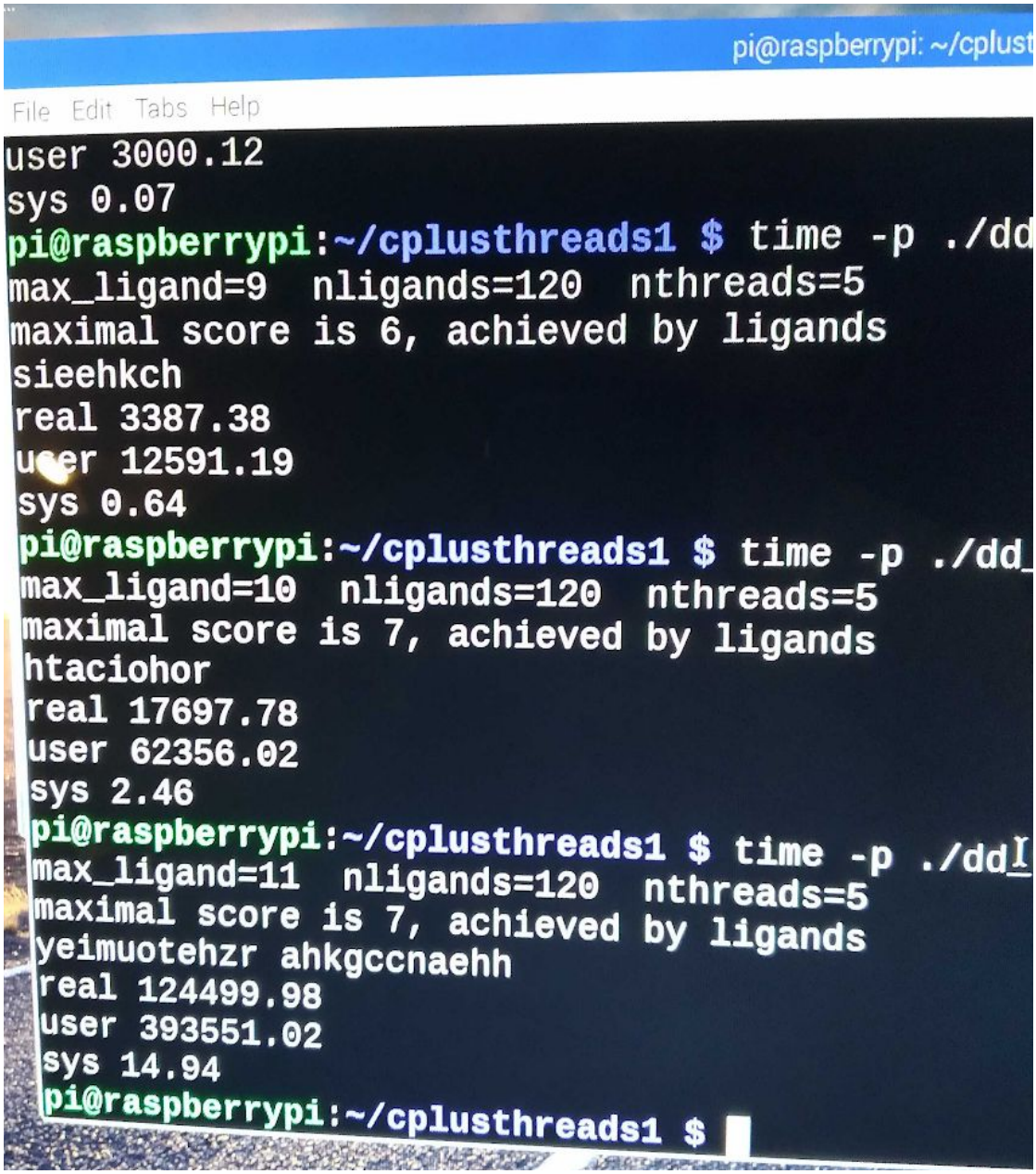
I had it run for 17 hours for 11 ligands and I ended up ctrl-c the program.

Using logger pro I have been able to use points from 7max ligands up to 10 to model the eventual point of 11 ligands.



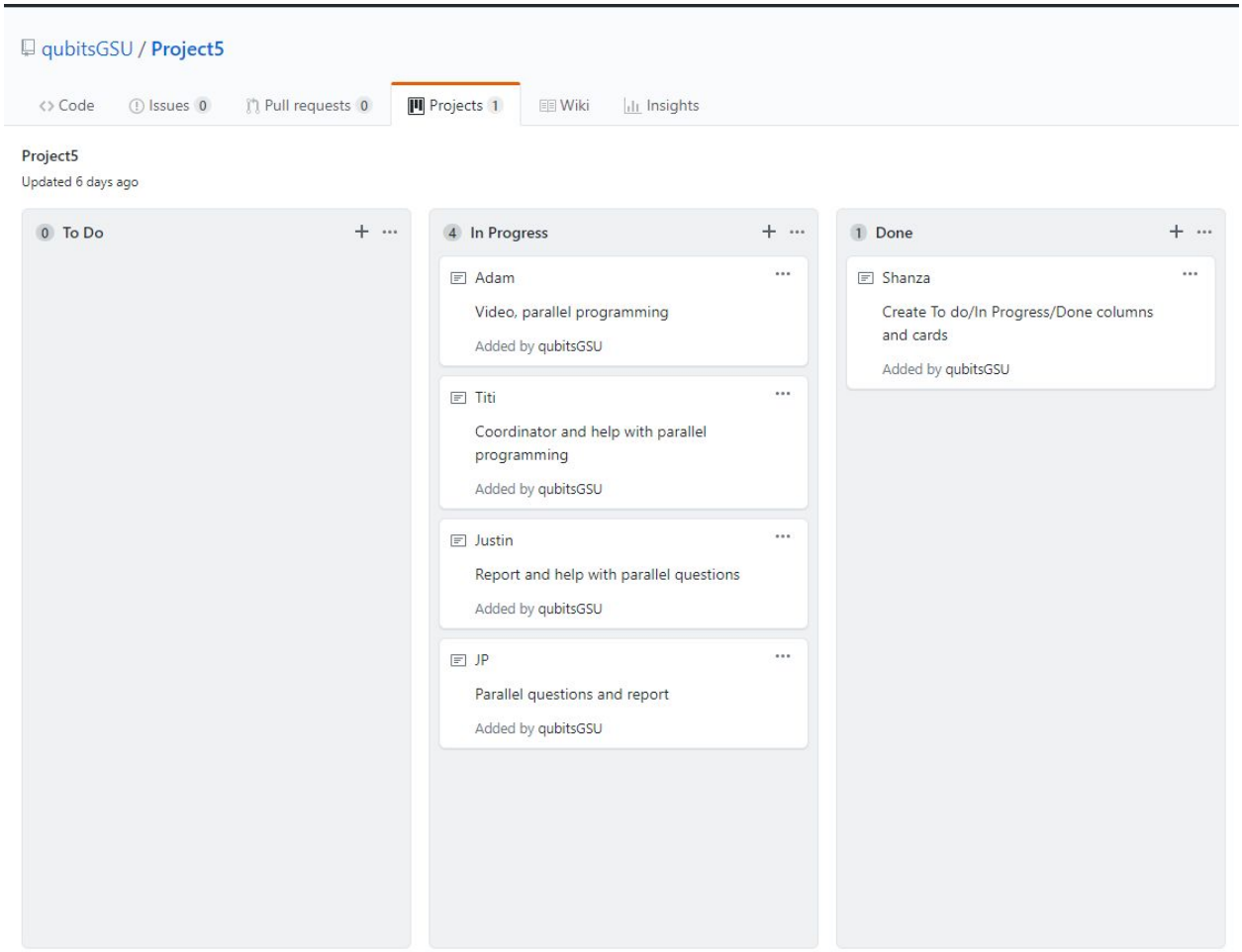
95,000 seconds. Which translates to 26.3 hours. Fingers crossed.
I performed an exponential fit equation to achieve this.

	Data Set	
	X	Y
1	7	82
2	8	791
3	9	3387
4	10	17697
5	11	95000


pi@raspberrypi: ~/cplust
File Edit Tabs Help
user 3000.12
sys 0.07
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd
max_ligand=9 nligands=120 nthreads=5
maximal score is 6, achieved by ligands
sieehkch
real 3387.38
user 12591.19
sys 0.64
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_
max_ligand=10 nligands=120 nthreads=5
maximal score is 7, achieved by ligands
htaciohor
real 17697.78
user 62356.02
sys 2.46
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd
max_ligand=11 nligands=120 nthreads=5
maximal score is 7, achieved by ligands
yeimuotehzr ahkgccnaehh
real 124499.98
user 393551.02
sys 14.94
pi@raspberrypi:~/cplusthreads1 $

Appendix:

GitHub



Important Links:

- **Slack:** computerorganizespr19.slack.com
- **GitHub:** https://github.com/qubitsGSU
- **YouTube:** https://www.youtube.com/watch?v=ZWBHoP2wuPA