Developing Soft and Parallel Programming Skills Using Project- Based Learning

Spring 2020

The Commuters
Alaya Shack, Miguel Romo, Arteen Ghafourikia, Andre Nguyenphuc, Joan Galicia

Planning and Scheduling:

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Note
Arteen Ghafourikia	aghafourikia1@student.gsu.edu	Organize and schedule designated assignments. Complete individual parallel programming skills task and complete the parallel programming basics task.	4 hours	(none)	04/24	Review report for grammatical/ spelling errors
Miguel Romo	mromo1@student.gsu.edu	Create the Planning and Scheduling table. Complete individual parallel programming skills task and complete the parallel programming basics task.	30 mins	(none)	04/24	Review report for grammatical/ spelling errors.
Joan Galicia	Jgalicia2@student.gsu.edu	Upload report to Github. Complete individual parallel programming skills task and complete the	5 hours	Github	04/24	Review report for grammatical/ spelling errors.

		parallel programming basics task.				
Alaya Shack	ashack1@student.gsu.edu	Get the report formatted correctly (fonts, page numbers, and sections). Complete individual parallel programming skills task and complete the parallel programming basics task.	5 hours	(none)	04/24	Review report for grammatical/ spelling errors. 24 hours before the due date, please have the report ready for all team members to review.
Andre Nguyenphuc (Team Coordinator)	anguyenphuc1@student.gsu.edu	Team Coordinator. Edit the video and include a link in the report. Complete individual parallel programming skills task and complete the parallel programming basics task.	3.5 hours	(none)	04/24	Review report for grammatical/ spelling errors.

Parallel Programming Skills Foundation:Alaya Shack

- ➔ **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**

- ◆ First, identify the set of tasks that can run simultaneously or divisions of data that can be processed simultaneously. Then, examine if the data can be decomposed into equal-size chunks, and if it can, a parallel solution can be developed. Then, decide which implementation should be used such as master/worker. With master/worker, examine data and split it up, depending on the number of workers, assign/send workers their data. Next, workers perform tasks on their data and return results to master. To summarize, the basic steps are decomposition, assignment, orchestration, and mapping.
- ◆ One example of a parallel program is pharmaceutical design, in which programs are developed to process large amounts of data related to medicine to further advance and develop different medications. Another example would be a parallel program, with master/worker implementation, designed to calculate pi, by using a circle inscribed a square.

➔ **What is MapReduce?**

- ◆ MapReduce is a programming model and a software framework that is used to process and generate large amounts of data with a parallel, distributed algorithm. It is based on the combinators map and reduces that developed within Google.

➔ **What is map and what is reduce?**

- ◆ Map, or the map function, takes in an input pair, which usually consists of a function and values, and the function is applied to the values and produces intermediate key/value pairs.
- ◆ Reduce, or the reduce function, takes the intermediate key/value pairs from the map and combines these values into a smaller set of output.

➔ **Why MapReduce?**

- ◆ MapReduce is used because it allows engineers to perform simple computations, while using abstraction to hide details about parallelization, data distribution, load balancing, and fault tolerance. Also, MapReduce helps with making data more readable and helps process data quickly , by reducing pairs into smaller sets.

➔ **Show an example for MapReduce.**

- ◆ An example for MapReduce would be Hadoop MapReduce. Another example of MapReduce would be calculating the number of different words and the number times that they appear in a document. The map would have the different words in the text and a 1. Then, the reduce function would combine the occurences of the different words into a single output.

➔ **Explain in your own words how MapReduce model is executed?**

- ◆ The MapReduce model is executed with master/worker implementation. First, the MapReduce library in the user program divides input files into pieces and copies the program on various machines, with one copy being the master and the others are workers. The master assigns a reduce or map task to a worker, and if the worker is assigned a map task, they must create their key/value pairs, which are then buffered in memory. The locations of the buffered pairs are passed to the master, who sends the locations to the reduce workers. Then, the reduce worker performs its tasks on data. The reduce worker passes its work to the user's reduce function. Then, the MapReduce call in the user program goes back to the user code.

➔ **List and describe three examples that are expressed as MapReduce computations.**

- ◆ Count of URL Access Frequency-In this computation, the map function examines logs of web page requests and outputs the URL and a 1 as a key value pair. Then, the reduce function adds the number of occurrences for the same URL and outputs the <URL,total count> pair.
- ◆ Term-Vector per Host-In this computation, there is a term vector that gets the most important words that occur in a document as a list of <word, frequency pairs>. The greater the frequency, the more important a word is. The map function creates a <hostname, term vector> for each document. Then, the reduce function combines the term-vectors corresponding to documents retrieved from the same host and produces a final <hostname, term vector> pair..
- ◆ Distributed Grep- In this computation, the map function outputs a line if it matches a certain pattern. Then, the reduce function just copies the intermediate data to the output.

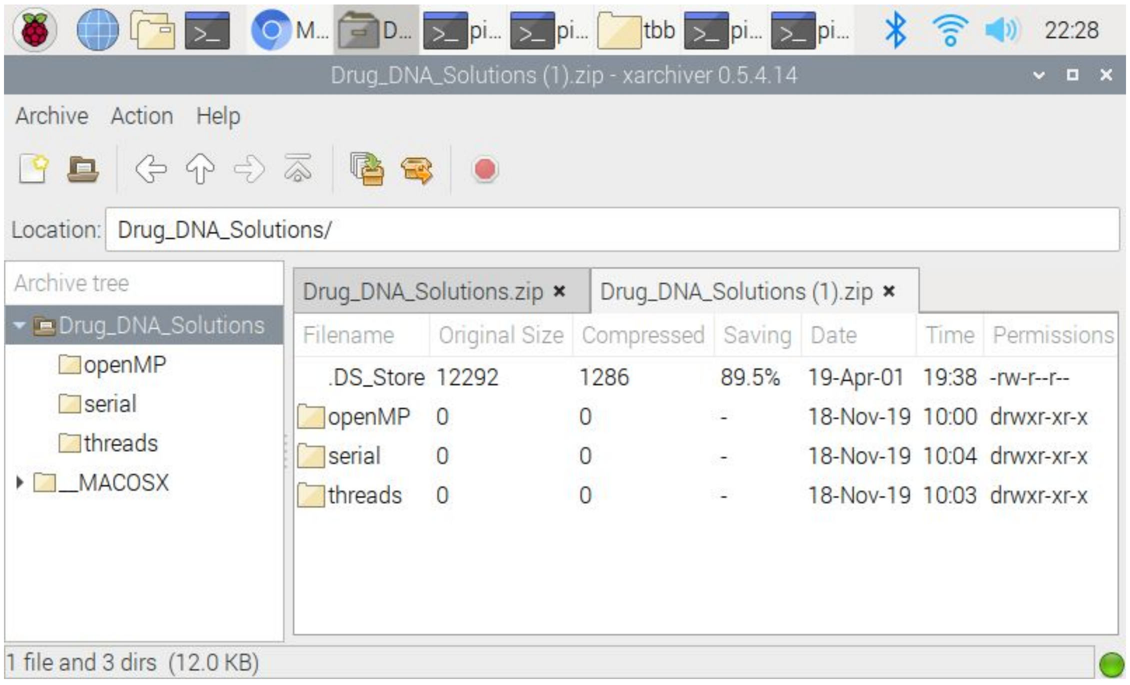
➔ **When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**

- ◆ OpenMP is used when introducing shared memory parallelism to code because of its efficient directive based library. OpenMP can help to parallelize for loops and helps to avoid writing manual thread code.
- ◆ Message Passing Interface(MPI) is used to develop tightly synchronous code and well load balanced such as parallel scientific applications, and MPI is fairly versatile because it can be used to develop almost any parallel code that uses multiple machines.
- ◆ MapReduce(Hadoop) is used when there is a large amount of data such as terabytes, and the user wants to perform operations such as extract, transform, and load. We use MapReduce(Hadoop) because of its high fault tolerance, which refers to the ability of a system to continue operating even with failure in one of its components.

➔ **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**

- ◆ DNA contains instructions for making proteins in our body, and proteins perform specific functions in a person's body. When a drug is designed by pharmaceutical companies, ligands are used to change a protein's shape. With the Drug Design and DNA problem, the goal is to find certain ligands that would be considered good candidates to be used as drugs. The algorithm behind the drug design software is to generate different combinations of ligands to compare against a particular protein. Then, each ligand will get a score based on how well the ligand will fit the desired shape and how well it fits a protein. Based on this information, the highest scoring ligands will be considered for actual production and testing.

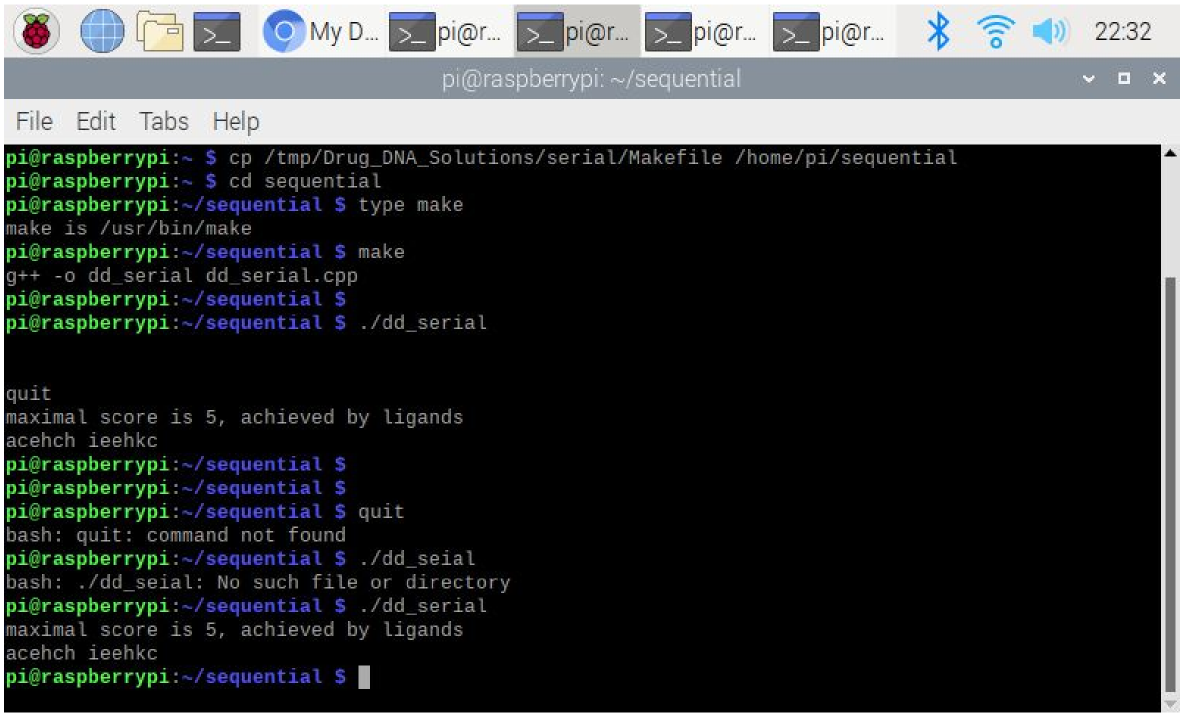
Parallel Programming Basics:Alaya Shack



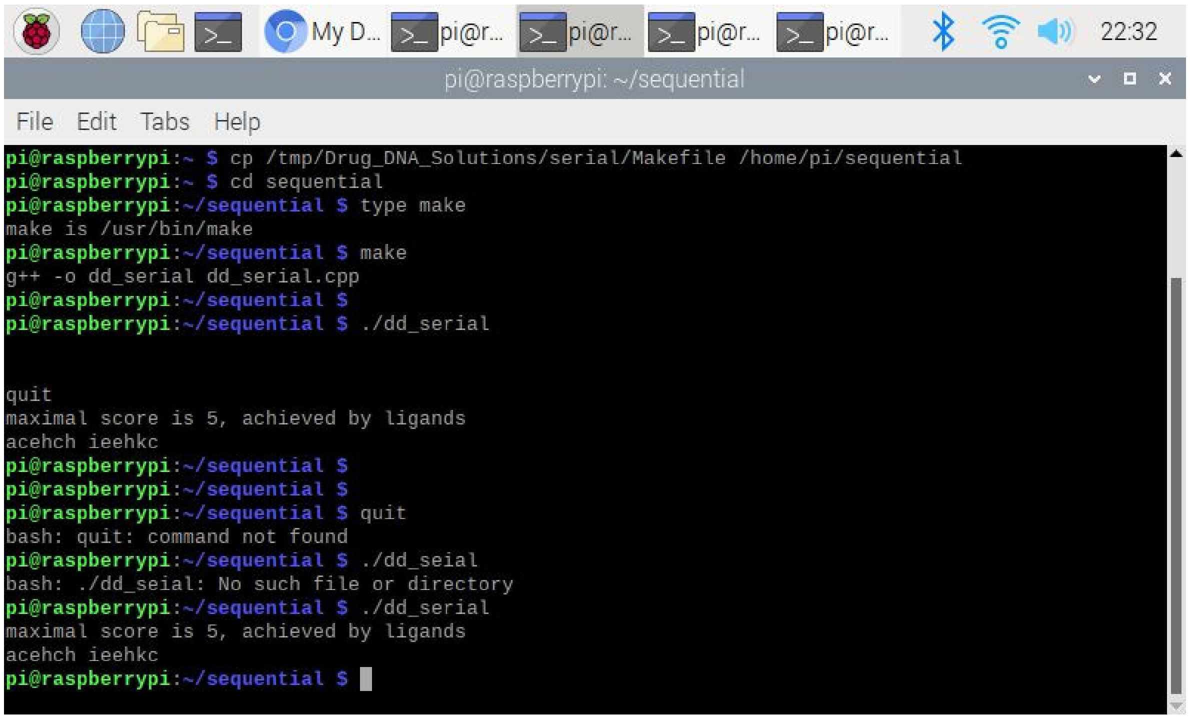
Filename	Original Size	Compressed	Saving	Date	Time	Permissions
.DS_Store	12292	1286	89.5%	19-Apr-01	19:38	-rw-r--r--
openMP	0	0	-	18-Nov-19	10:00	drwxr-xr-x
serial	0	0	-	18-Nov-19	10:04	drwxr-xr-x
threads	0	0	-	18-Nov-19	10:03	drwxr-xr-x

This screenshot is how the downloaded Drug DNA Solutions file looked with the three solutions of openMP, serial, and threads.

Part 1



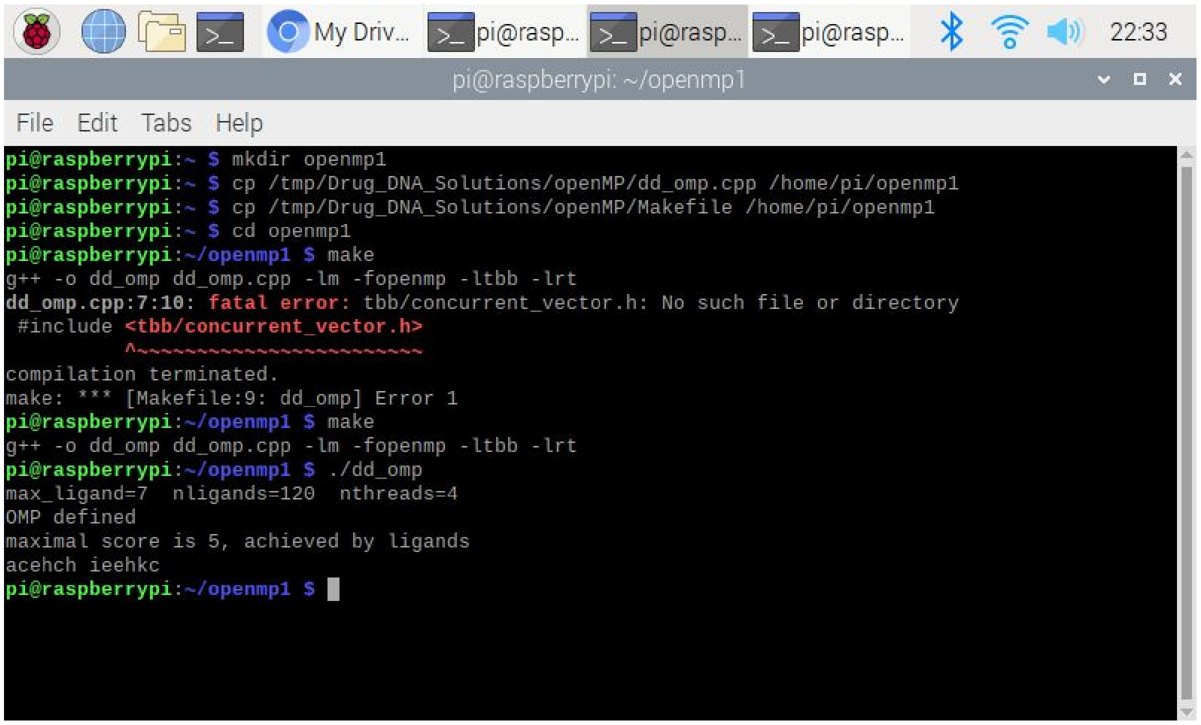
```
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/serial/Makefile /home/pi/sequential
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ type make
make is /usr/bin/make
pi@raspberrypi:~/sequential $ make
g++ -o dd_serial dd_serial.cpp
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $ ./dd_serial


quit
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $ quit
bash: quit: command not found
pi@raspberrypi:~/sequential $ ./dd_seial
bash: ./dd_seial: No such file or directory
pi@raspberrypi:~/sequential $ ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/sequential $
```



```
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/serial/Makefile /home/pi/sequential
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ type make
make is /usr/bin/make
pi@raspberrypi:~/sequential $ make
g++ -o dd_serial dd_serial.cpp
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $ ./dd_serial


quit
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $
pi@raspberrypi:~/sequential $ quit
bash: quit: command not found
pi@raspberrypi:~/sequential $ ./dd_seial
bash: ./dd_seial: No such file or directory
pi@raspberrypi:~/sequential $ ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/sequential $
```

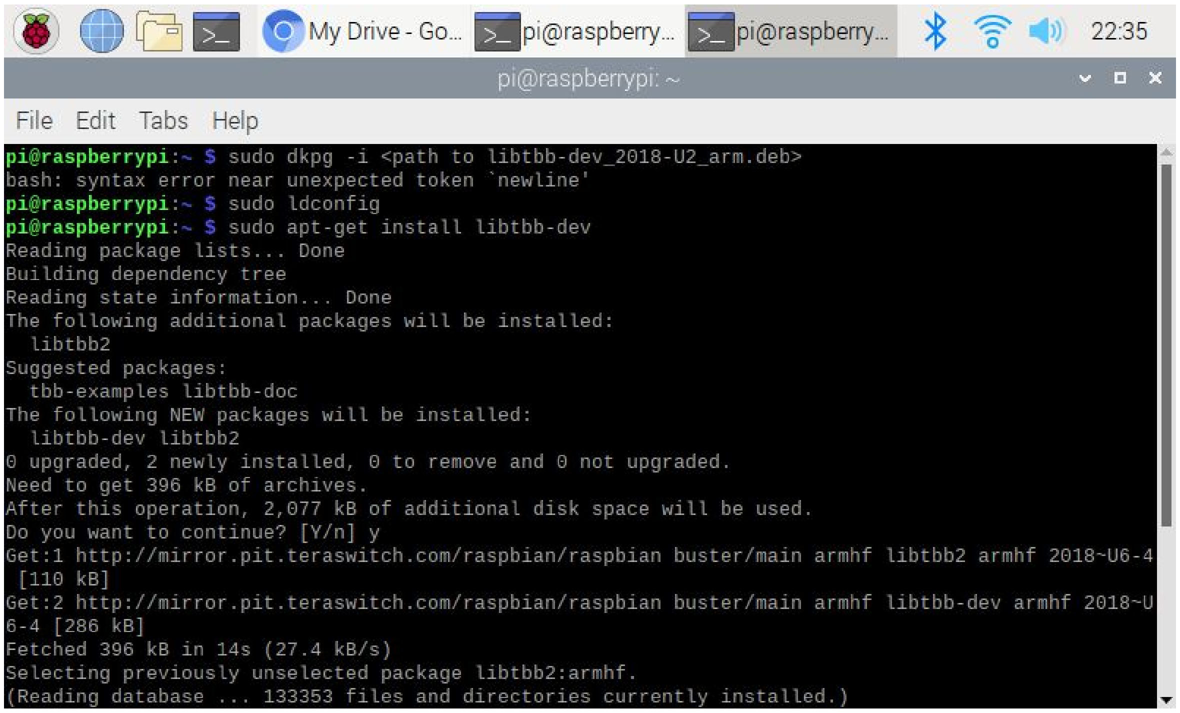
These two screenshots show how I made the directory called sequential, and I copied the Makefile and dd_serial.cpp into that directory. Then, I used cd to change the directory to sequential. I typed make, then the compiling phrase popped up, which was surprising because I thought that I would have to type it in. Then, I ran the program with “./dd_serial”. Then, the output appeared with the best score of 5 shown, and the ligands “acehch” and “ieehkc” that would make good drug candidates.

Part 2

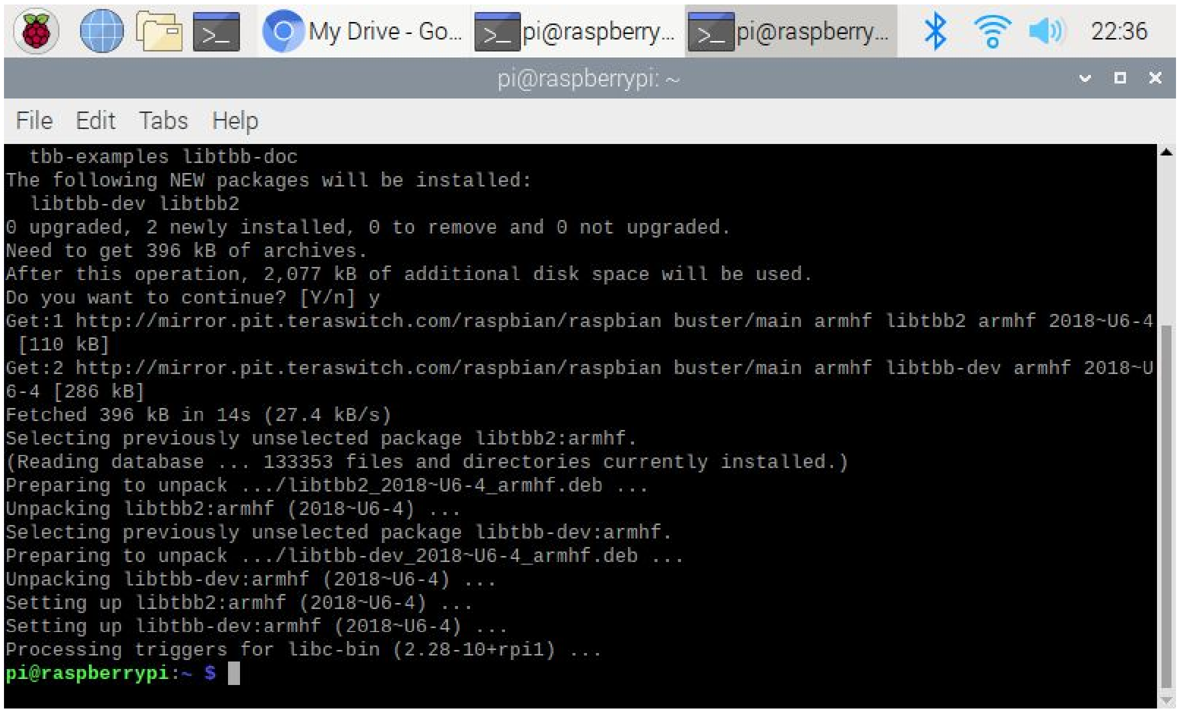


```
pi@raspberrypi:~ $ mkdir openmp1
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/openMP/dd_omp.cpp /home/pi/openmp1
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/openMP/Makefile /home/pi/openmp1
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ make
g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
dd_omp.cpp:7:10: fatal error: tbb/concurrent_vector.h: No such file or directory
 #include <tbb/concurrent_vector.h>
          ^~~~~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
make: *** [Makefile:9: dd_omp] Error 1
pi@raspberrypi:~/openmp1 $ make
g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/openmp1 $ ./dd_omp
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/openmp1 $
```

This screenshot shows how I made the directory openmp1 and copied the appropriate files to that directory. Then, I changed the directory with “cd openmp1” command. I encountered an error with tbb/concurrent_vector.h because the file directory did not exist. Finally, after researching and discussing with my teammates, we realized that we needed to download the tbb library. When I downloaded the file, I was able to use make and the compiler phrase appeared with no error. I then ran the program, and the output displayed the maximum length of a ligand(max_ligand), and the number of ligands generated(nligands), number of threads(nthreads), and the phrase OMP defined. The output was the same as the serial solution. This solution was not as slow as the serial, but it was still slower than expected.

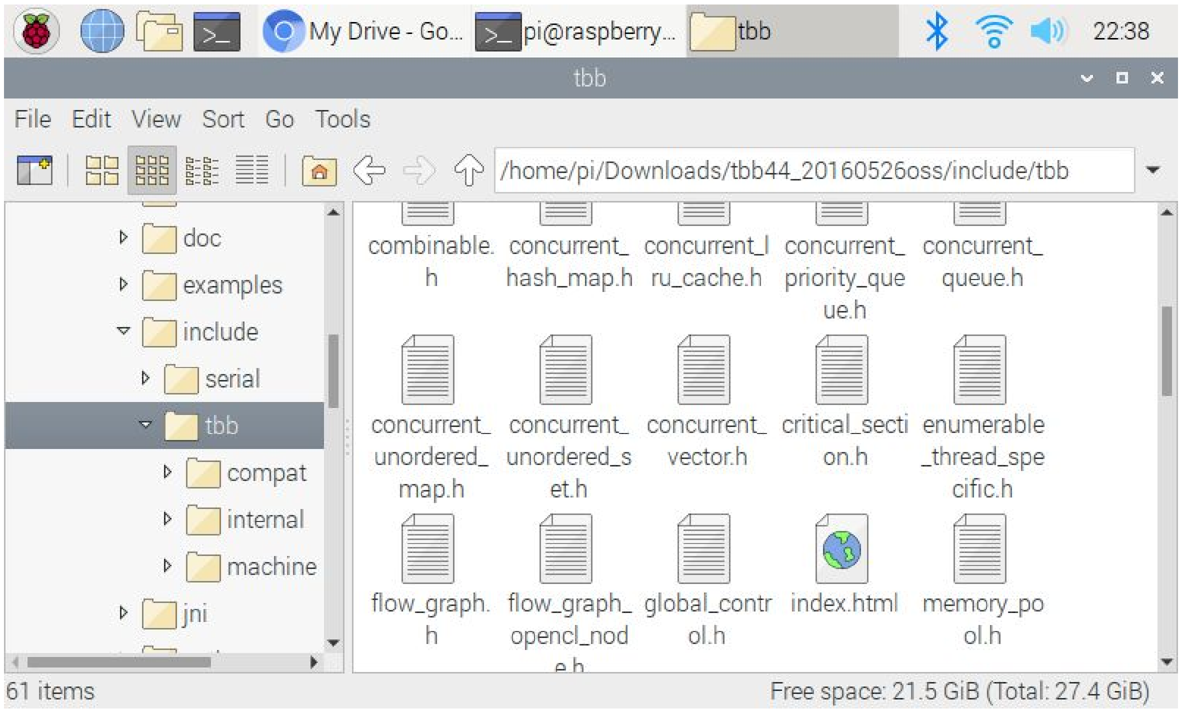


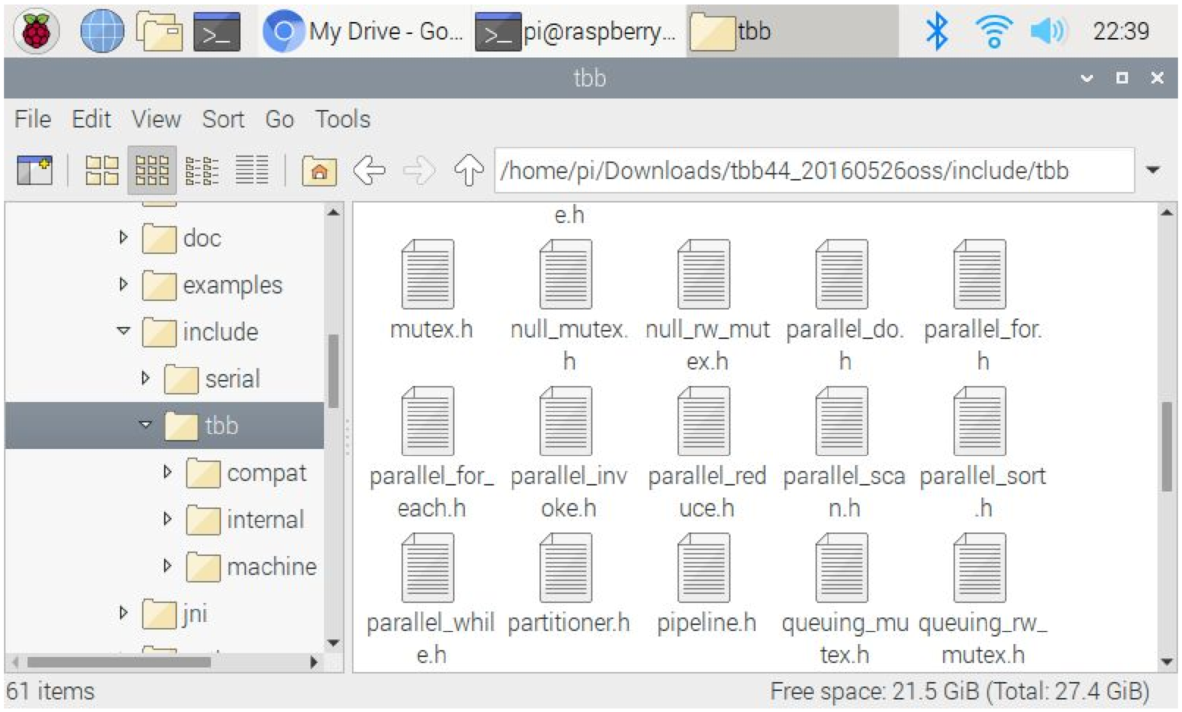
```
pi@raspberrypi:~ $ sudo dkpg -i <path to libtbb-dev_2018-U2_arm.deb>
bash: syntax error near unexpected token `newline'
pi@raspberrypi:~ $ sudo ldconfig
pi@raspberrypi:~ $ sudo apt-get install libtbb-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libtbb2
Suggested packages:
  tbb-examples libtbb-doc
The following NEW packages will be installed:
  libtbb-dev libtbb2
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 396 kB of archives.
After this operation, 2,077 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtbb2 armhf 2018~U6-4
 [110 kB]
Get:2 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtbb-dev armhf 2018~U
6-4 [286 kB]
Fetched 396 kB in 14s (27.4 kB/s)
Selecting previously unselected package libtbb2:armhf.
(Reading database ... 133353 files and directories currently installed.)
```



```
  tbb-examples libtbb-doc
The following NEW packages will be installed:
  libtbb-dev libtbb2
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 396 kB of archives.
After this operation, 2,077 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtbb2 armhf 2018~U6-4
 [110 kB]
Get:2 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtbb-dev armhf 2018~U
6-4 [286 kB]
Fetched 396 kB in 14s (27.4 kB/s)
Selecting previously unselected package libtbb2:armhf.
(Reading database ... 133353 files and directories currently installed.)
Preparing to unpack .../libtbb2_2018~U6-4_armhf.deb ...
Unpacking libtbb2:armhf (2018~U6-4) ...
Selecting previously unselected package libtbb-dev:armhf.
Preparing to unpack .../libtbb-dev_2018~U6-4_armhf.deb ...
Unpacking libtbb-dev:armhf (2018~U6-4) ...
Setting up libtbb2:armhf (2018~U6-4) ...
Setting up libtbb-dev:armhf (2018~U6-4) ...
Processing triggers for libc-bin (2.28-10+rpi1) ...
pi@raspberrypi:~ $
```

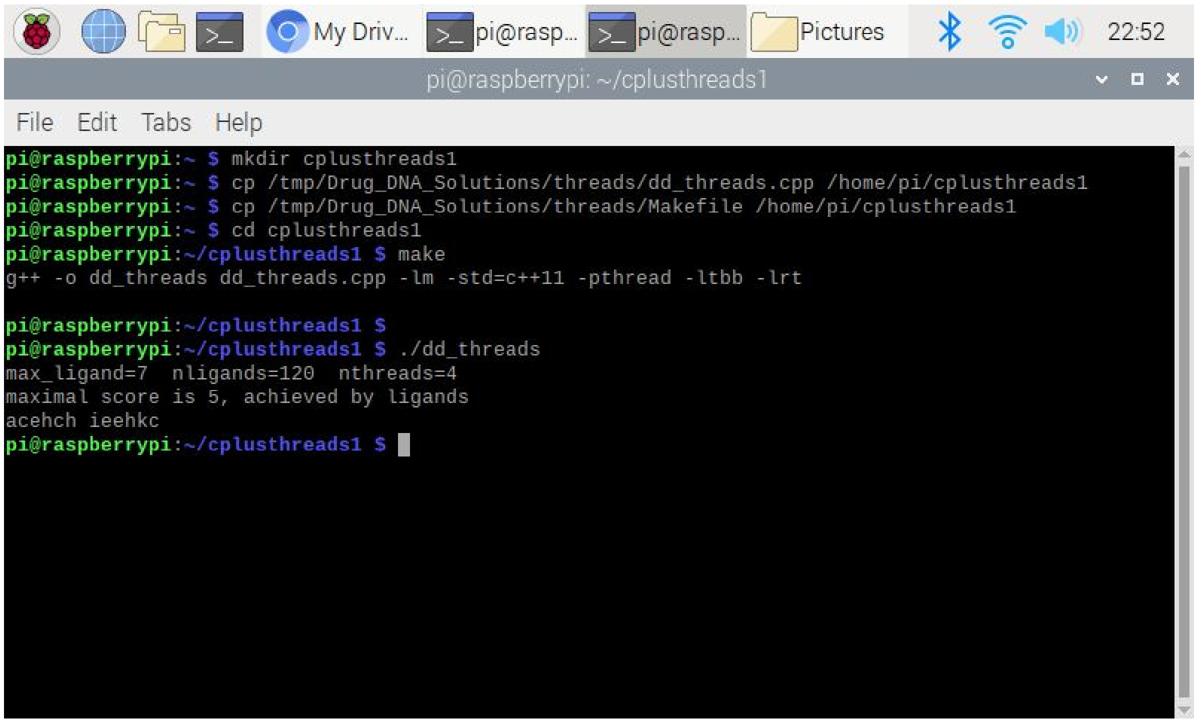
These two screenshots show how I downloaded the appropriate library, so that the openmp solution would work. The first two lines were mistakes, and did not properly download the file. However, the third line with "sudo apt -get install libtbb-dev" installed the library on the Pi.





These two screenshots show the files that I needed from tbb in order to correctly compile dd_openmp, which were concurrent_vector.h and parallel_sort.h.

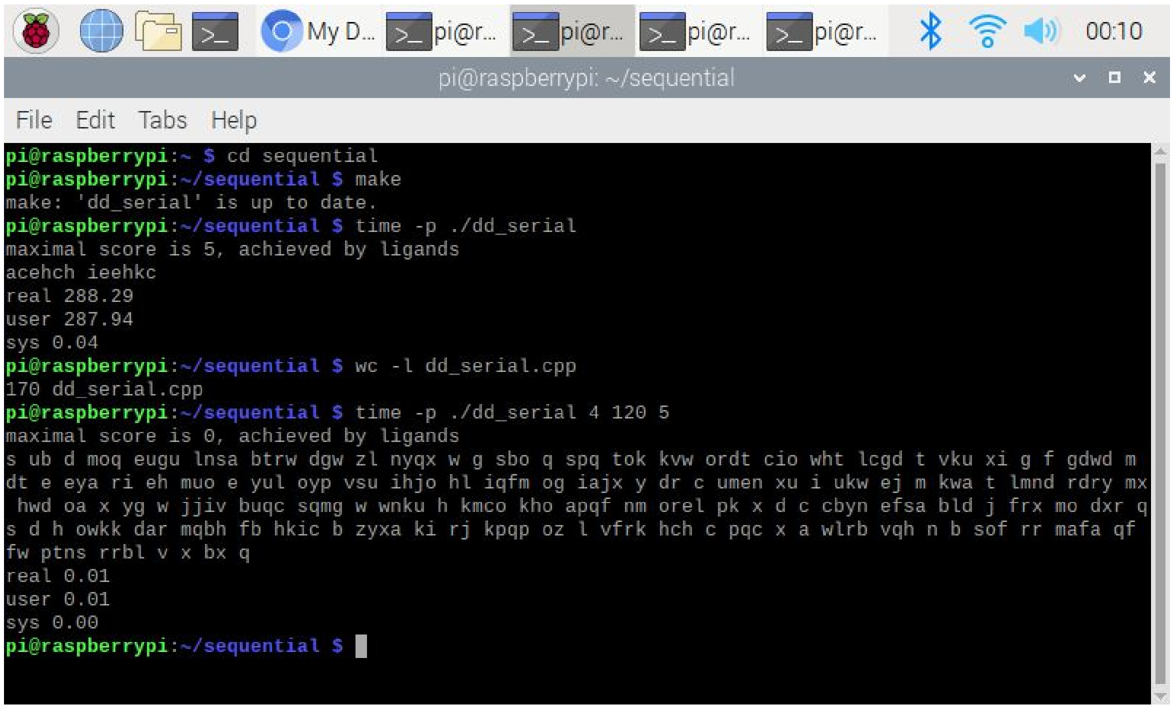
Part 3



```
pi@raspberrypi:~ $ mkdir cplusthreads1
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/threads/dd_threads.cpp /home/pi/cplusthreads1
pi@raspberrypi:~ $ cp /tmp/Drug_DNA_Solutions/threads/Makefile /home/pi/cplusthreads1
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ make
g++ -o dd_threads dd_threads.cpp -lm -std=c++11 -pthread -ltbb -lrt

pi@raspberrypi:~/cplusthreads1 $
pi@raspberrypi:~/cplusthreads1 $ ./dd_threads
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/cplusthreads1 $
```

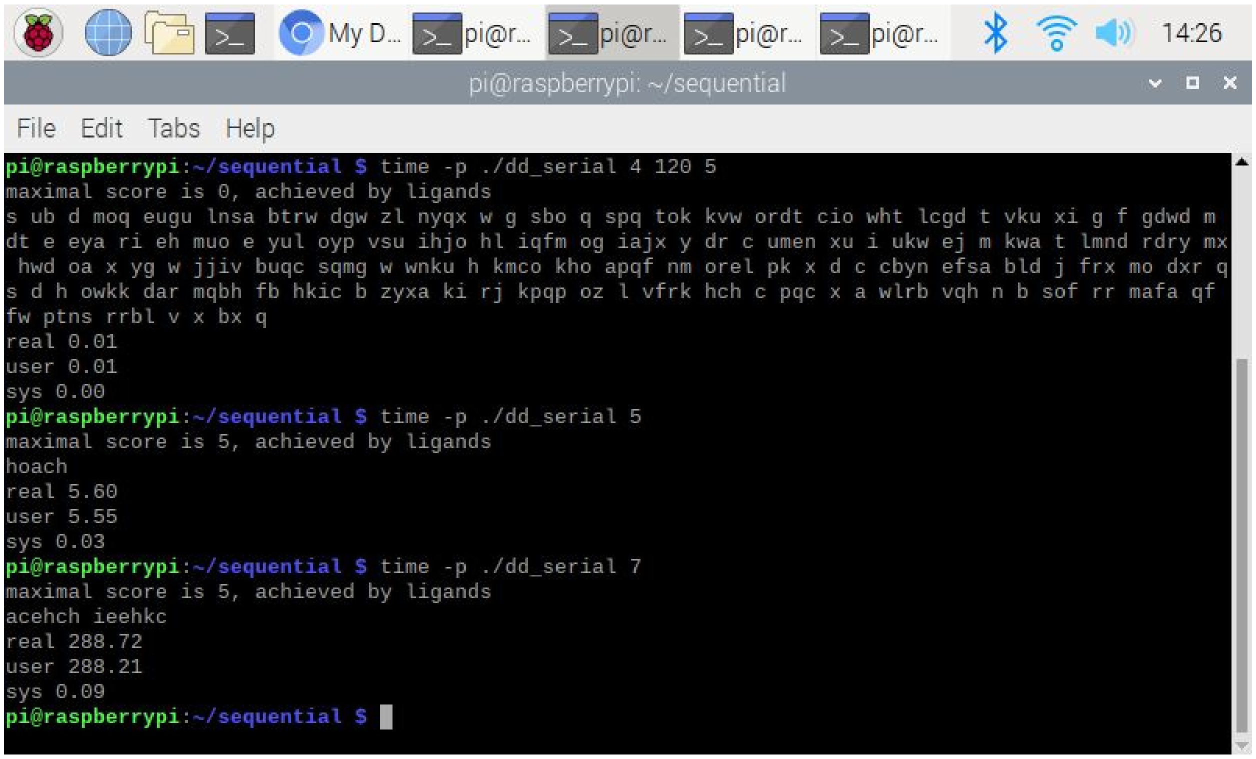
This screenshot shows how I made the directory cplusthreads1 and copied the appropriate files into it. Then, I used "cd cplusthreads1" to change the directory, and then I typed make, which made the compiler phrase appear. I ran the program with "./dd_threads", and the same output from openmp1 was displayed minus the OMP Defined.

Measure Run-Time

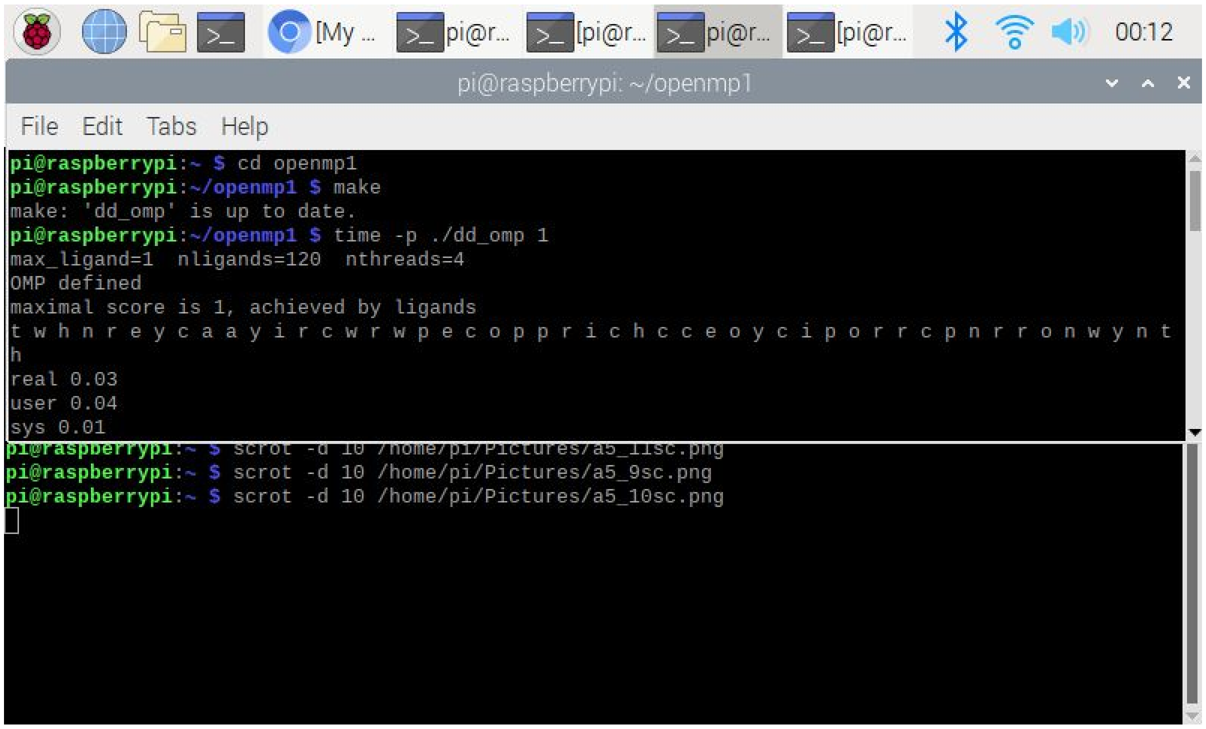


```
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ make
make: 'dd_serial' is up to date.
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 288.29
user 287.94
sys 0.04
pi@raspberrypi:~/sequential $ wc -l dd_serial.cpp
170 dd_serial.cpp
pi@raspberrypi:~/sequential $ time -p ./dd_serial 4 120 5
maximal score is 0, achieved by ligands
s ub d moq eugu lnsa btrw dgw zl nyqx w g sbo q spq tok kvw ordt cio wht lcgd t vku xi g f gdwd m
dt e eya ri eh muo e yul oyp vsu ihjo hl iqfm og iajx y dr c umen xu i ukw ej m kwa t lmnd rdry mx
 hwd oa x yg w jjiv buqc sqmg w wnku h kmco kho apqf nm orel pk x d c cbyn efsa bld j frx mo dxr q
s d h owkk dar mqbh fb hkic b zyxa ki rj kpqp oz l vfrk hch c pqc x a wlrb vqh n b sof rr mafa qf
fw ptns rrbl v x bx q
real 0.01
user 0.01
sys 0.00
pi@raspberrypi:~/sequential $
```

These screenshots show the runtime of dd_serial.cpp, and the number of lines in dd_serial.cpp.



```
pi@raspberrypi:~/sequential $ time -p ./dd_serial 4 120 5
maximal score is 0, achieved by ligands
s ub d moq eugu lnsa btrw dgw zl nyqx w g sbo q spq tok kvw ordt cio wht lcgd t vku xi g f gdwd m
dt e eya ri eh muo e yul oyp vsu ihjo hl iqfm og iajx y dr c umen xu i ukw ej m kwa t lmnd rdry mx
 hwd oa x yg w jjiv buqc sqmg w wnku h kmco kho apqf nm orel pk x d c cbyn efsa bld j frx mo dxr q
s d h owkk dar mqbh fb hkic b zyxa ki rj kpqp oz l vfrk hch c pqc x a wlrb vqh n b sof rr mafa qf
fw ptns rrbl v x bx q
real 0.01
user 0.01
sys 0.00
pi@raspberrypi:~/sequential $ time -p ./dd_serial 5
maximal score is 5, achieved by ligands
hoach
real 5.60
user 5.55
sys 0.03
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 288.72
user 288.21
sys 0.09
pi@raspberrypi:~/sequential $
```

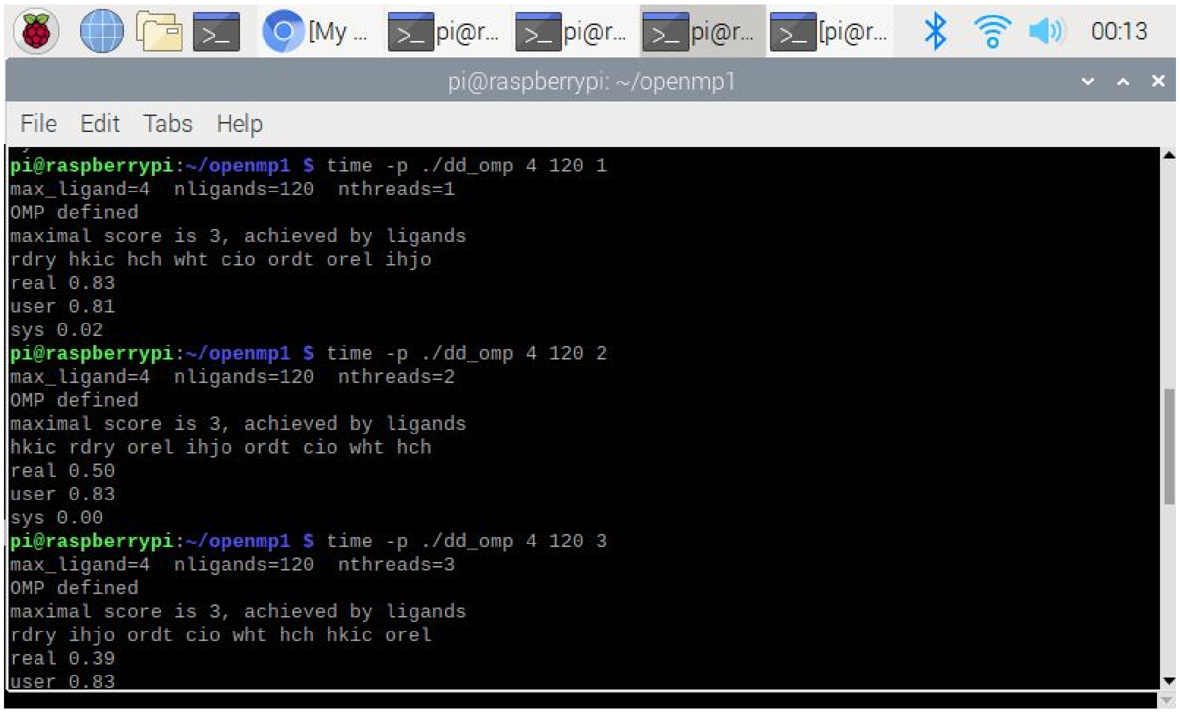


```
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ make
make: 'dd_omp' is up to date.
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1  nligands=120  nthreads=4
OMP defined
maximal score is 1, achieved by ligands
t w h n r e y c a a y i r c w r w p e c o p p r i c h c c e o y c i p o r r c p n r r o n w y n t
h
real 0.03
user 0.04
sys 0.01
pi@raspberrypi:~ $ scrot -d 10 /home/pi/Pictures/a5_11sc.png
pi@raspberrypi:~ $ scrot -d 10 /home/pi/Pictures/a5_9sc.png
pi@raspberrypi:~ $ scrot -d 10 /home/pi/Pictures/a5_10sc.png
```

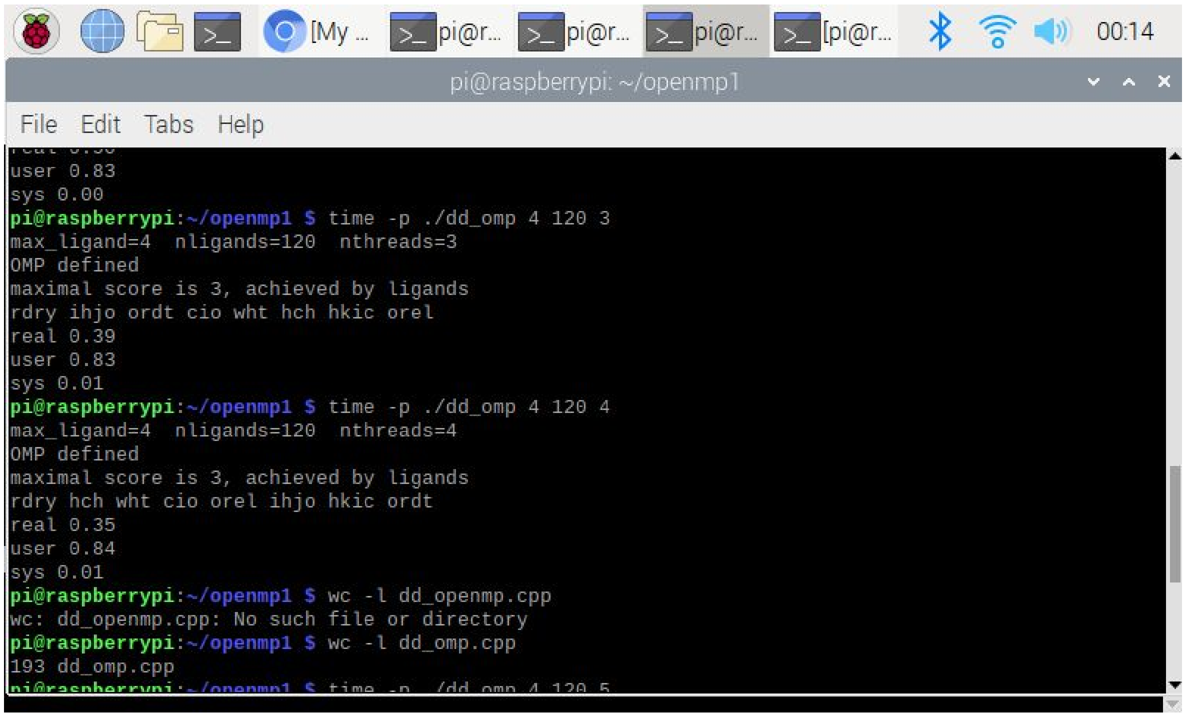
This screenshot shows the runtime of dd_omp with max ligand being 1.

Implementation	**Time(s)**
dd_serial	288.29
dd_omp	0.03
dd_threads	0.02

Implementation	**Time(s) 2 Threads**	**Time(s) 3 Threads**	**Time (s) 4 Threads**
dd_omp	.50	.39	.35
dd_threads	.43	.30	.26

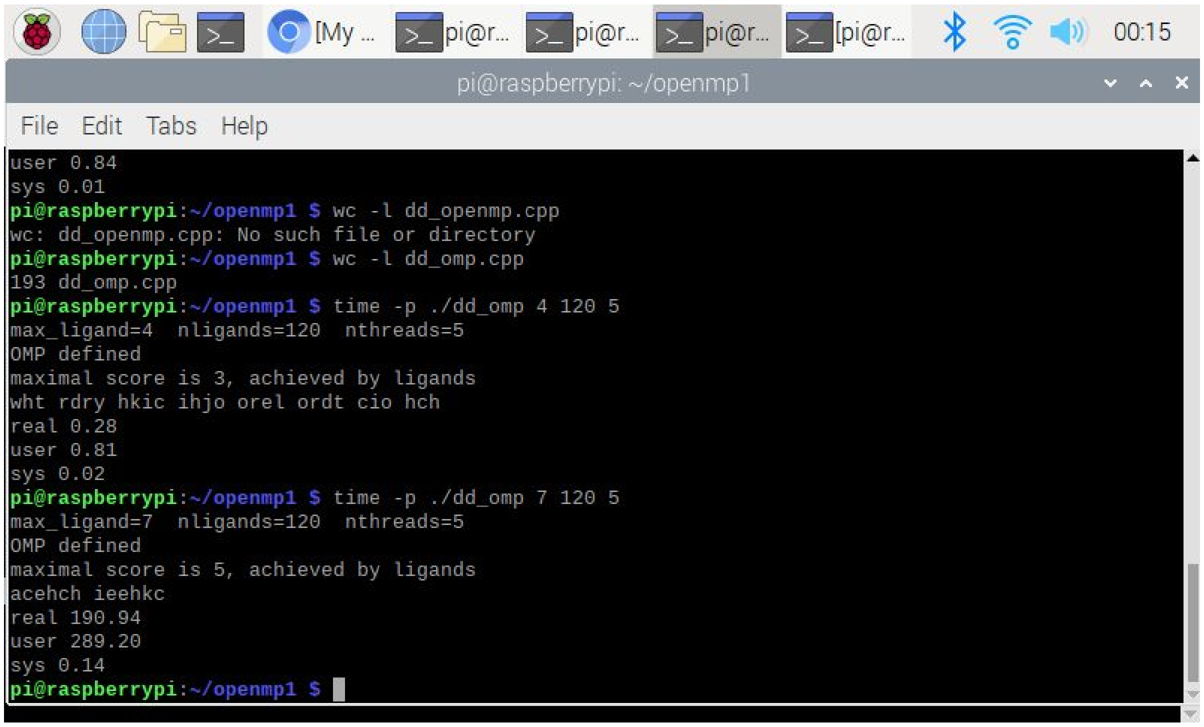


```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 1
max_ligand=4  nligands=120  nthreads=1
OMP defined
maximal score is 3, achieved by ligands
rdry hkic hch wht cio ordt orel ihjo
real 0.83
user 0.81
sys 0.02
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 2
max_ligand=4  nligands=120  nthreads=2
OMP defined
maximal score is 3, achieved by ligands
hkic rdry orel ihjo ordt cio wht hch
real 0.50
user 0.83
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 3
max_ligand=4  nligands=120  nthreads=3
OMP defined
maximal score is 3, achieved by ligands
rdry ihjo ordt cio wht hch hkic orel
real 0.39
user 0.83
```



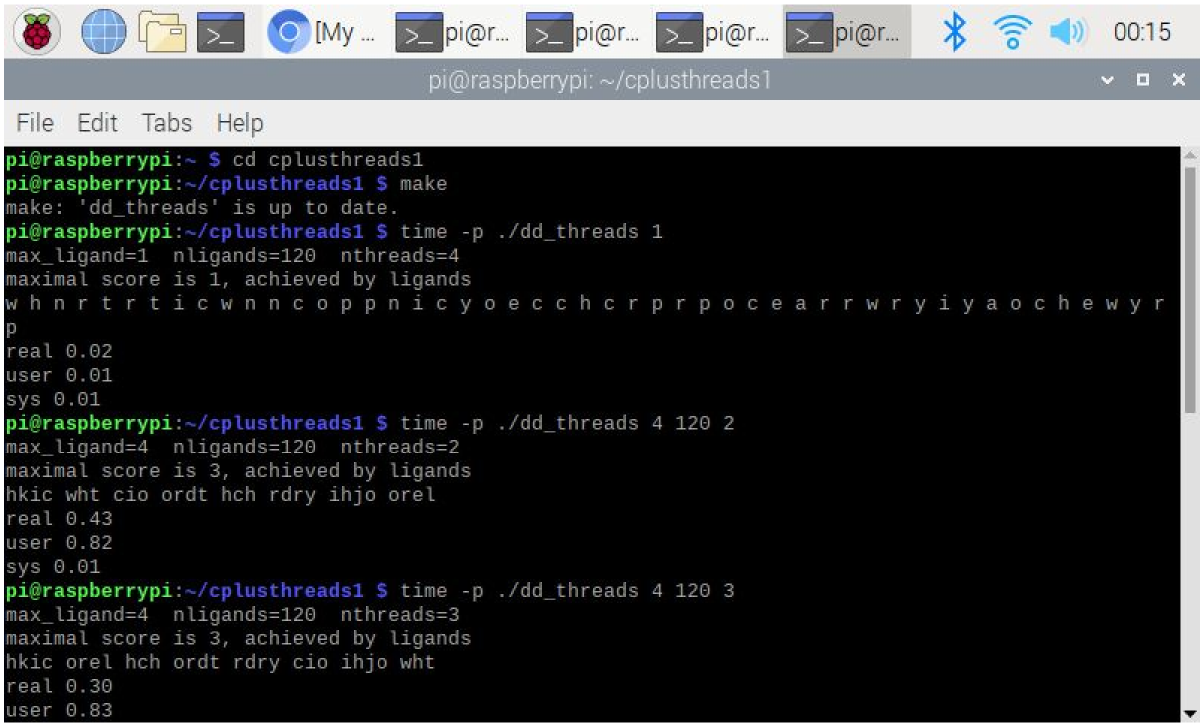
```
user 0.83
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 3
max_ligand=4  nligands=120  nthreads=3
OMP defined
maximal score is 3, achieved by ligands
rdry ihjo ordt cio wht hch hkic orel
real 0.39
user 0.83
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 4
max_ligand=4  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
rdry hch wht cio orel ihjo hkic ordt
real 0.35
user 0.84
sys 0.01
pi@raspberrypi:~/openmp1 $ wc -l dd_openmp.cpp
wc: dd_openmp.cpp: No such file or directory
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
193 dd_omp.cpp
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 5
```

These screenshots show how I made the max_ligand=4, nthreads=120, and nthreads=2 and the various runtimes. Also, the number of lines in dd_openmp1 are displayed.

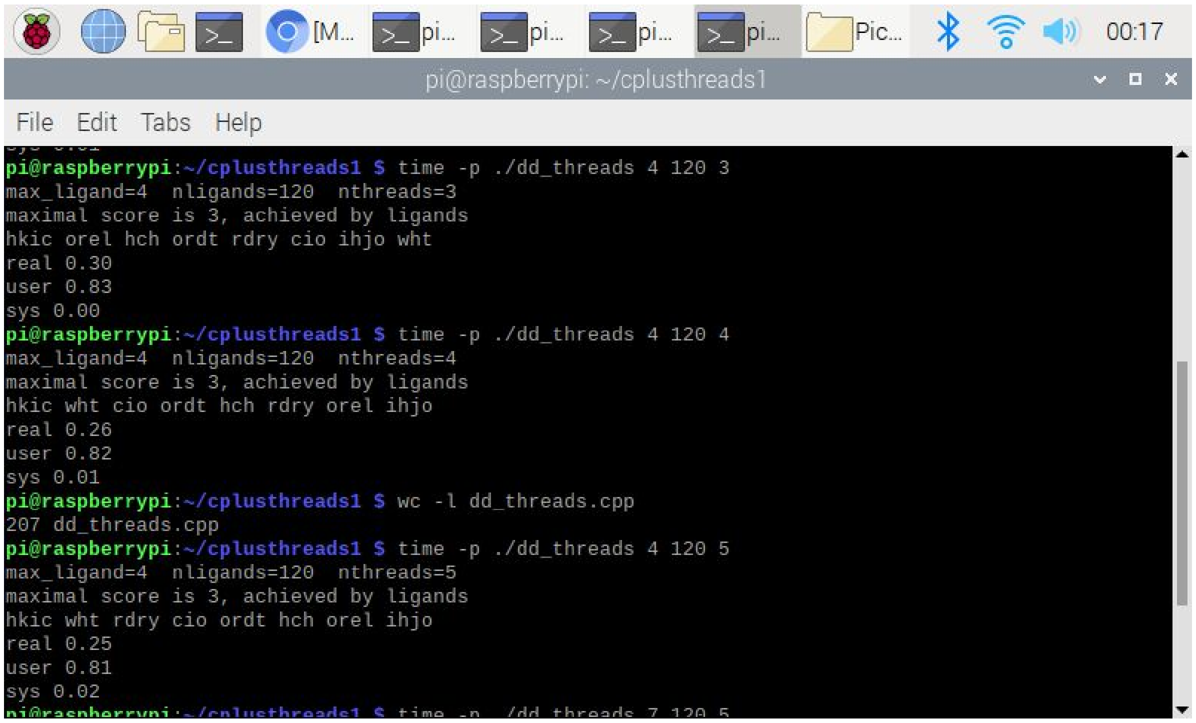


```
user 0.84
sys 0.01
pi@raspberrypi:~/openmp1 $ wc -l dd_openmp.cpp
wc: dd_openmp.cpp: No such file or directory
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
193 dd_omp.cpp
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 5
max_ligand=4  nligands=120  nthreads=5
OMP defined
maximal score is 3, achieved by ligands
wht rdry hkic ihjo orel ordt cio hch
real 0.28
user 0.81
sys 0.02
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 190.94
user 289.20
sys 0.14
pi@raspberrypi:~/openmp1 $ 
```

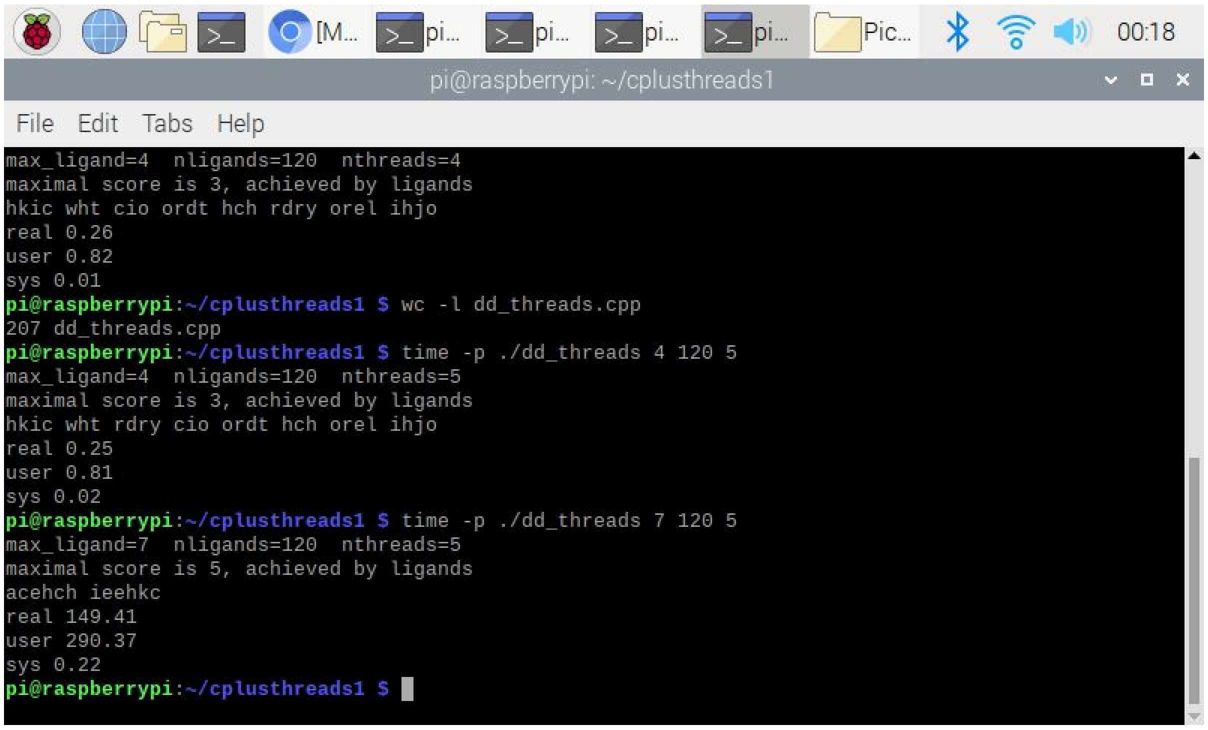
This screenshot shows how I changed the number of threads to 5, and the max_ligand changed to 7.



```
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ make
make: 'dd_threads' is up to date.
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1
max_ligand=1  nligands=120  nthreads=4
maximal score is 1, achieved by ligands
w h n r t r t i c w n n c o p p n i c y o e c c h c r p r p o c e a r r w r y i y a o c h e w y r
p
real 0.02
user 0.01
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 2
max_ligand=4  nligands=120  nthreads=2
maximal score is 3, achieved by ligands
hkic wht cio ordt hch rdry ihjo orel
real 0.43
user 0.82
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 3
max_ligand=4  nligands=120  nthreads=3
maximal score is 3, achieved by ligands
hkic orel hch ordt rdry cio ihjo wht
real 0.30
user 0.83
```



```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 3
max_ligand=4  nligands=120  nthreads=3
maximal score is 3, achieved by ligands
hkic orel hch ordt rdry cio ihjo wht
real 0.30
user 0.83
sys 0.00
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 4
max_ligand=4  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
hkic wht cio ordt hch rdry orel ihjo
real 0.26
user 0.82
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
207 dd_threads.cpp
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 5
max_ligand=4  nligands=120  nthreads=5
maximal score is 3, achieved by ligands
hkic wht rdry cio ordt hch orel ihjo
real 0.25
user 0.81
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7 120 5
```



```
max_ligand=4  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
hkic wht cio ordt hch rdry orel ihjo
real 0.26
user 0.82
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
207 dd_threads.cpp
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 5
max_ligand=4  nligands=120  nthreads=5
maximal score is 3, achieved by ligands
hkic wht rdry cio ordt hch orel ihjo
real 0.25
user 0.81
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7 120 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 149.41
user 290.37
sys 0.22
pi@raspberrypi:~/cplusthreads1 $
```

These three screenshots show the various runtimes of dd_threads. Some of the modifications include changing max_ligand=4, nthreads=120, and nthreads=2. Also, the number of lines in dd_threads is displayed. I also changed the number of threads to 5 and the max ligand to 7.

<u>Discussion Questions</u>

1. **Which approach is the fastest?**
    a. The C++11 threads solution is the fastest.
2. **Determine the number of lines in each file (use wc -l).How does the C++11 implementation compare to the OpenMP implementations?**
    a. The number of lines in the serial file are 170. The number of lines in the openmp file are 193. The number of lines in the threads file are 207. C++11 has 14 more lines than the openmp file
3. **Increase the number of threads to 5 threads.What is the run time for each?**
    a.

Implementation	Time(s)
dd_openmp	.28
dd_threads	.25
dd_serial	N/A

4. **Increase the maximum ligand length to 7, and rerun each program.What is the run time for each?**
    a.

Implementation	Time(s)
dd_openmp	190.94
dd_threads	149.41
dd_serial	288.92

<u>Parallel Programming Skills Foundation:Arteen Ghafourikia</u>

➔ **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**
  ◆ The first thing you want to do is to identify the sets of tasks that can be run concurrently. Then you look at the data and decompose them into equal-sized tasks. If there are no dependencies in the computation you want to do, you then

choose an implementation technique such as the master/worker technique. This method implements a static load balancing to prevent processors doing nothing. The task is then mapped to a processor.

- ◆ An example would be a master/worker implementation to calculate pi.

➔ **What is MapReduce?**

- ◆ It is a programming model that implements generating and processing big data sets with a parallel and shared algorithm on a cluster.

➔ **What is map and what is reduce?**

- ◆ Map takes the data and creates intermediate key value pairs.
- ◆ Reduce takes the intermediate key value pairs from the Map and combines them to produce the results.

➔ **Why MapReduce?**

- ◆ The framework is much faster because they can process huge amounts of data parallel with no communication with each other.

➔ **Show an example for MapReduce.**

- ◆ MapReduce is used with Hadoop to use large amounts of data to map and reduce to give the general data.

➔ **Explain in your own words how the MapReduce model is executed?**

- ◆ You execute by mapping and reducing which is done by creating key value pairs and taking those pairs and combining them to produce the designated results.

➔ **List and describe three examples that are expressed as MapReduce computations.**

- ◆ **Distributed Grep**- The map function displays a line if it is the same as the pattern. The reduce function copies the given data to the output.
- ◆ **Count of URL Access Frequency**- The map function examines the web page's requests and outputs. The reduce function sums up all the values for the given URL and displays a URL total count pair.
- ◆ **Reverse Web-Link Graph-** The map function outputs target source pairs for every link to a URL that is named "source". The reduce function merges all the source URLs of a marked URL and displays the pair: <target,list(source)>.

➔ **When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**

- ◆ **OpenMp-** You use OpenMp when you want to implement shared memory parallelism and it is useful because it breaks your loops in different threads to save a lot of time.
- ◆ **MPI-** MPI is a distributed memory parallel model typically used for scientific applications because of its balance and ability to tightly synchronize the code. It is useful for running code that uses multiple machines.
- ◆ **MapReduce(Hadoop)-** You use Hadoop when you have a lot of data and you want to extract, transform, and load which makes it great for scientific applications because it condenses the code for complex problems.

- ➔ **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**
  - ◆ Drug Design and DNA problem is when they generate ligands to find a good sequence to be used as a drug. They do this by creating different combinations of ligands and comparing them. They can then alter and change the sequence until they get the shape that works the best. This is useful when creating the best fitted drug for a certain problem. The simplified process is to generate ligands for a protein, then you get the score for each ligand and see which ones are the highest scored ligands.

<u>Parallel Programming Basics: Arteen Ghafourikia</u>

(The screenshots are below)

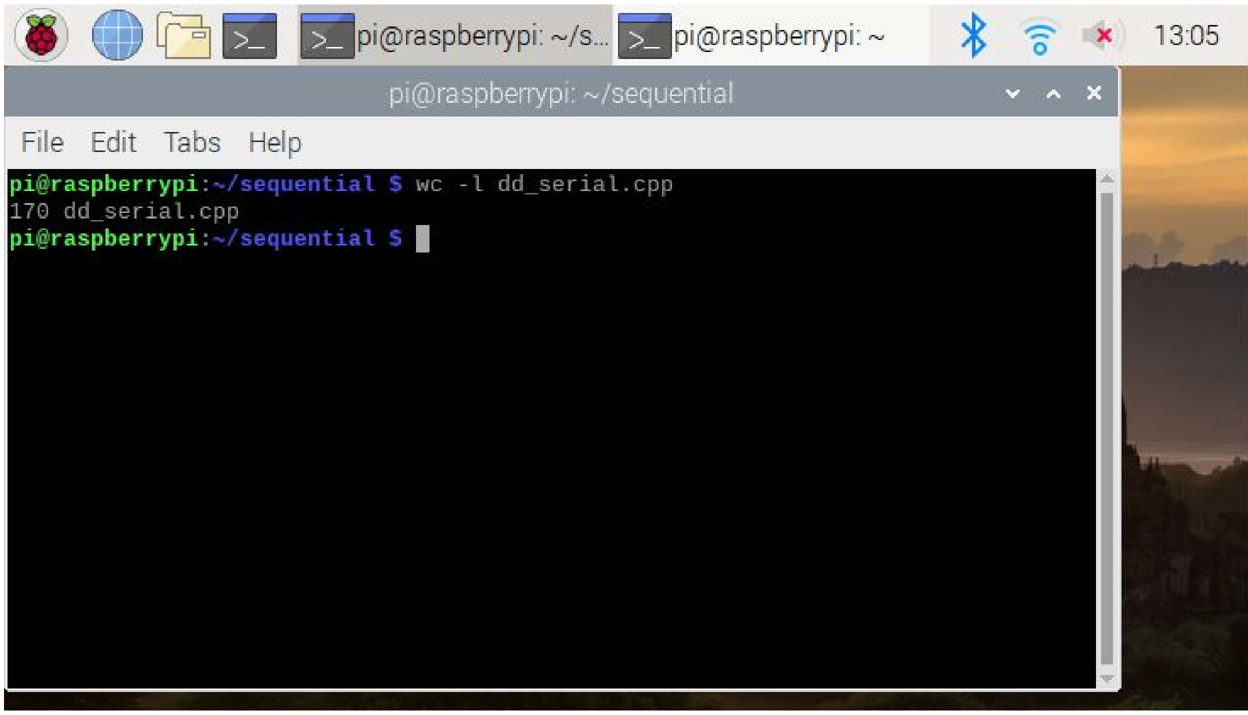
<u>Measure Run-Time</u>

Implementation 5 ligands	**Time (s)**
dd_serial	151.63
dd_omp	4.32
dd_threads	4.30

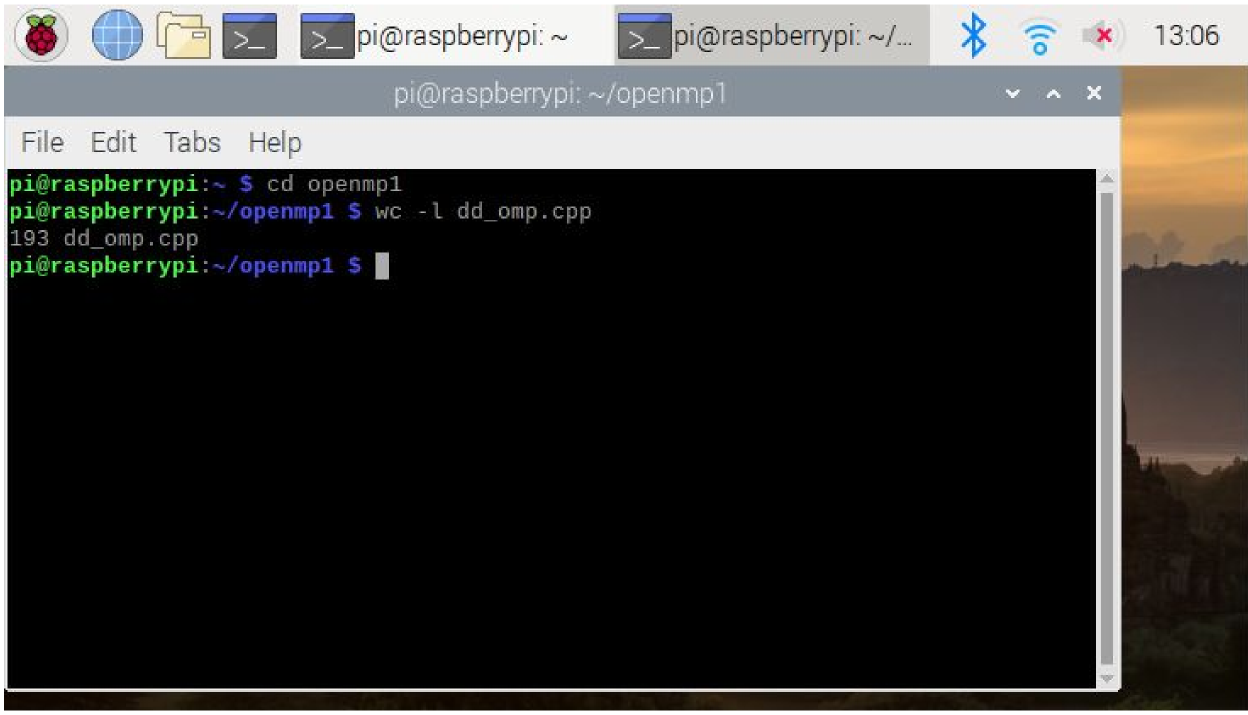
Implementation 5 ligands	**Time (s) 2 Threads**	**Time (s) 3 Threads**	**Time (s) 4 Threads**
dd_omp	3.27	2.38	2.03
dd_threads	2.77	1.75	1.38

<u>Discussion Questions</u>

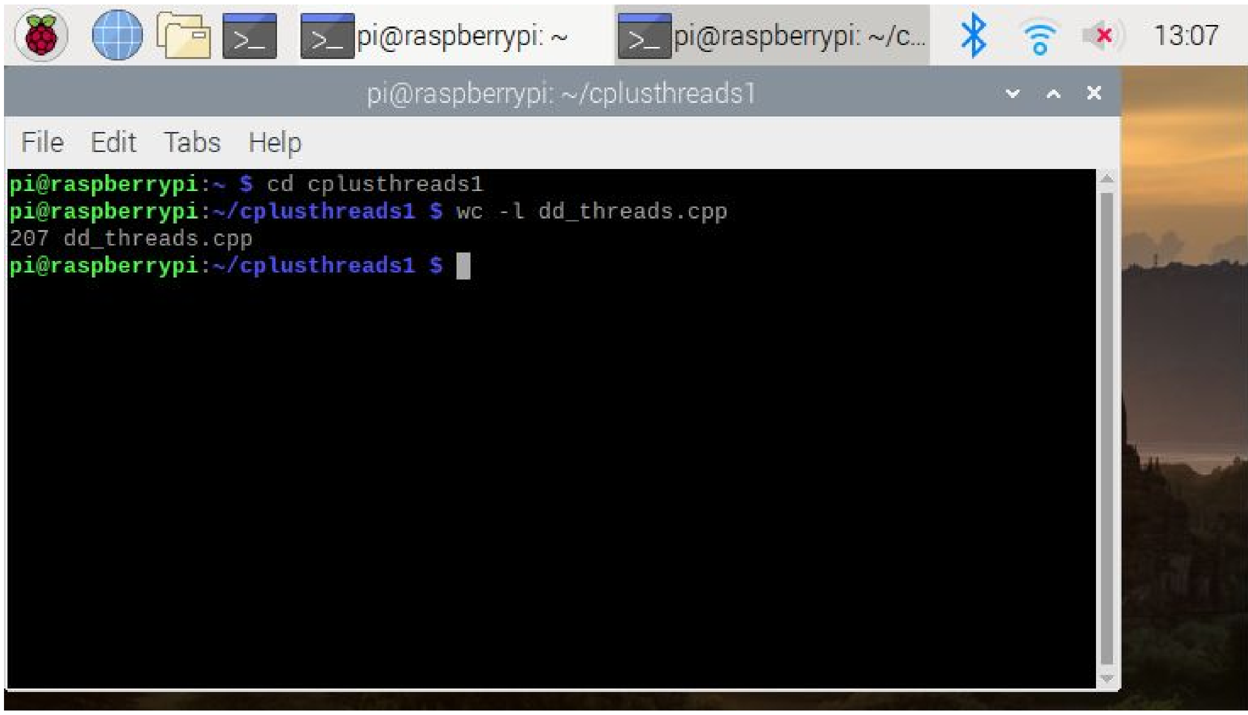
1. **Which approach is the fastest?**
   a. dd_threads is the fastest.
2. **Determine the number of lines in each file (use wc-1). How does the C++11 implementation compare to the OpenMP implementations?**
   a. Serial-170 lines
   b. OpenMp-193 lines
   c. Threads-207 lines
      i. There are 14 more lines in the C++11 implementation than in the openmp implementation.



```
pi@raspberrypi:~/sequential $ wc -l dd_serial.cpp
170 dd_serial.cpp
pi@raspberrypi:~/sequential $
```



```
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
193 dd_omp.cpp
pi@raspberrypi:~/openmp1 $
```



```
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
207 dd_threads.cpp
pi@raspberrypi:~/cplusthreads1 $ 
```

I used wc -l to find the number of lines.

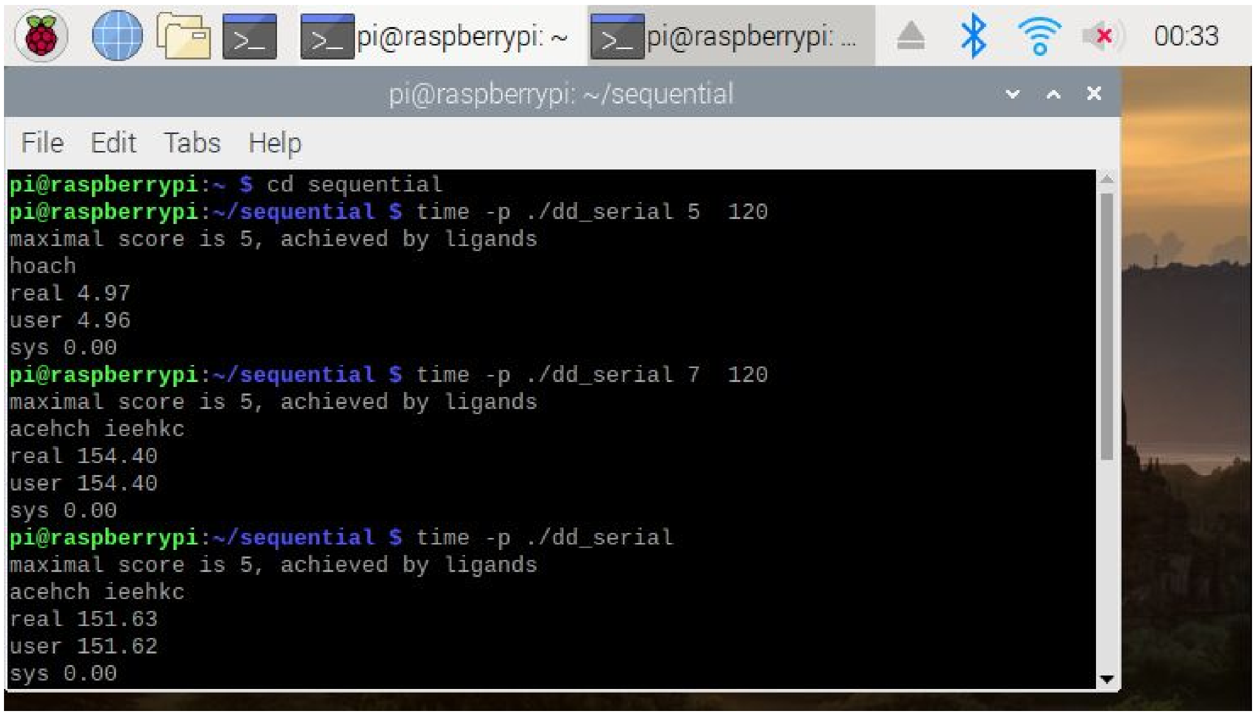
3. **Increase the number of threads to 5 threads. What is the run time for each?**
   (The screenshots are below)

Implementation	**Time (s) 5 Threads**
dd_serial	N/A
dd_omp	1.71
dd_threads	1.43

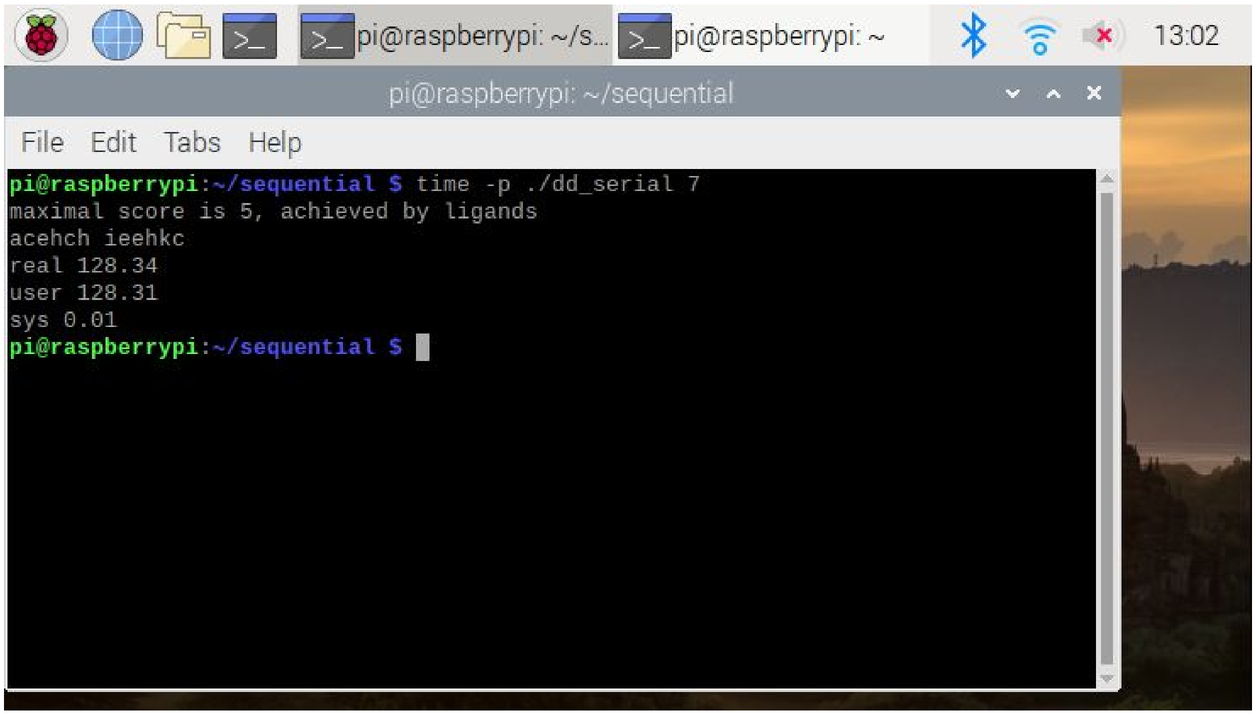
4.Increase the maximum ligand length to 7, and rerun each program.What is the run time for each?

Implementation (7 Ligands)	**Time (s)**
dd_serial	128.34

dd_omp	101.96
dd_threads	74.63

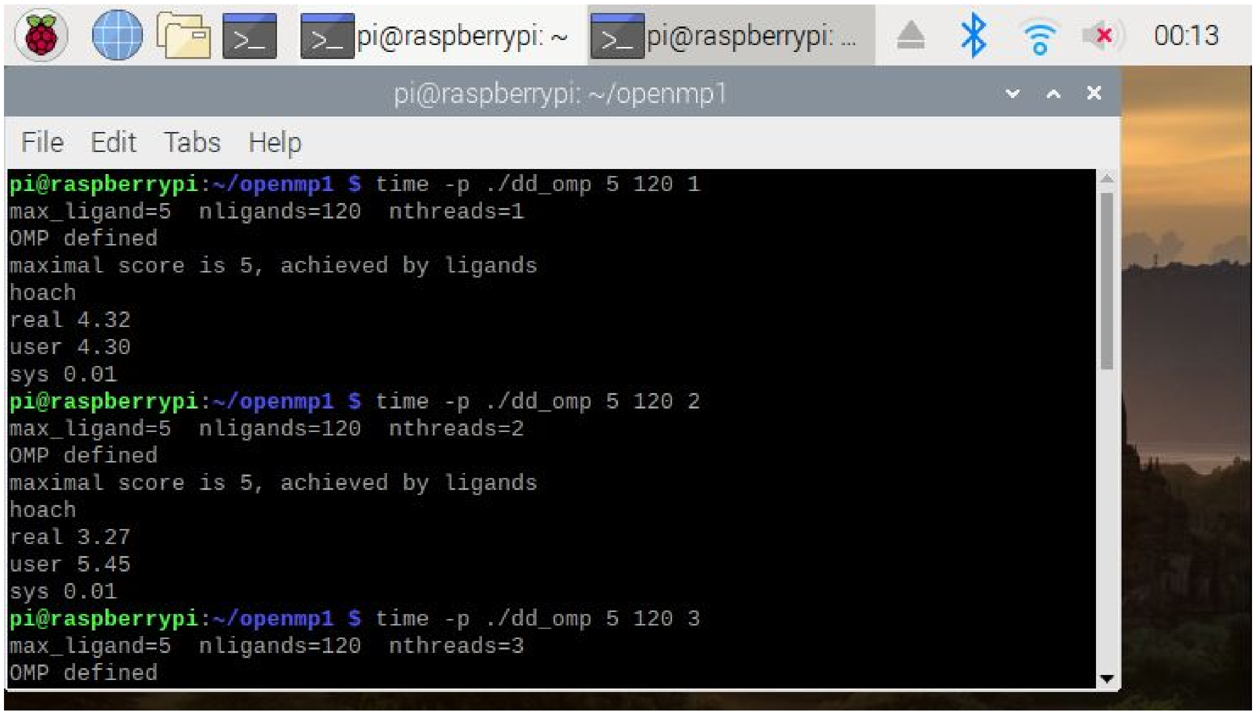


```
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ time -p ./dd_serial 5  120
maximal score is 5, achieved by ligands
hoach
real 4.97
user 4.96
sys 0.00
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7  120
maximal score is 5, achieved by ligands
acehch ieehkc
real 154.40
user 154.40
sys 0.00
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 151.63
user 151.62
sys 0.00
```

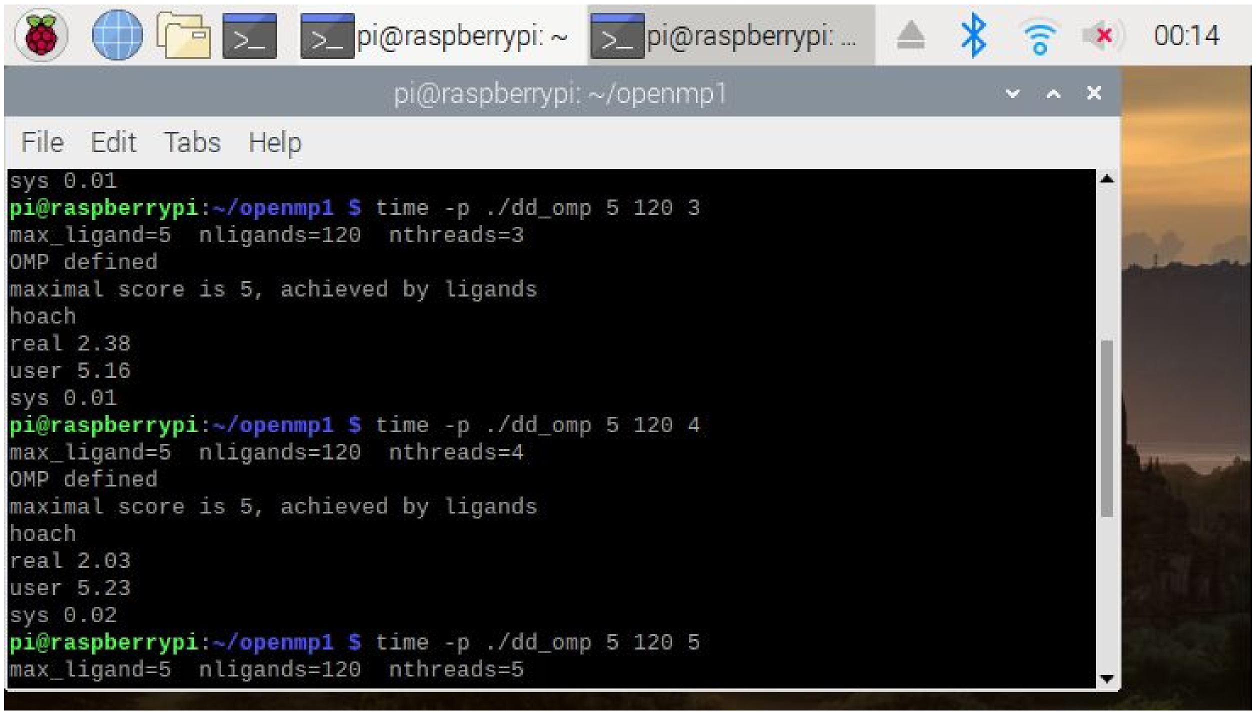


```
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 128.34
user 128.31
sys 0.01
pi@raspberrypi:~/sequential $
```

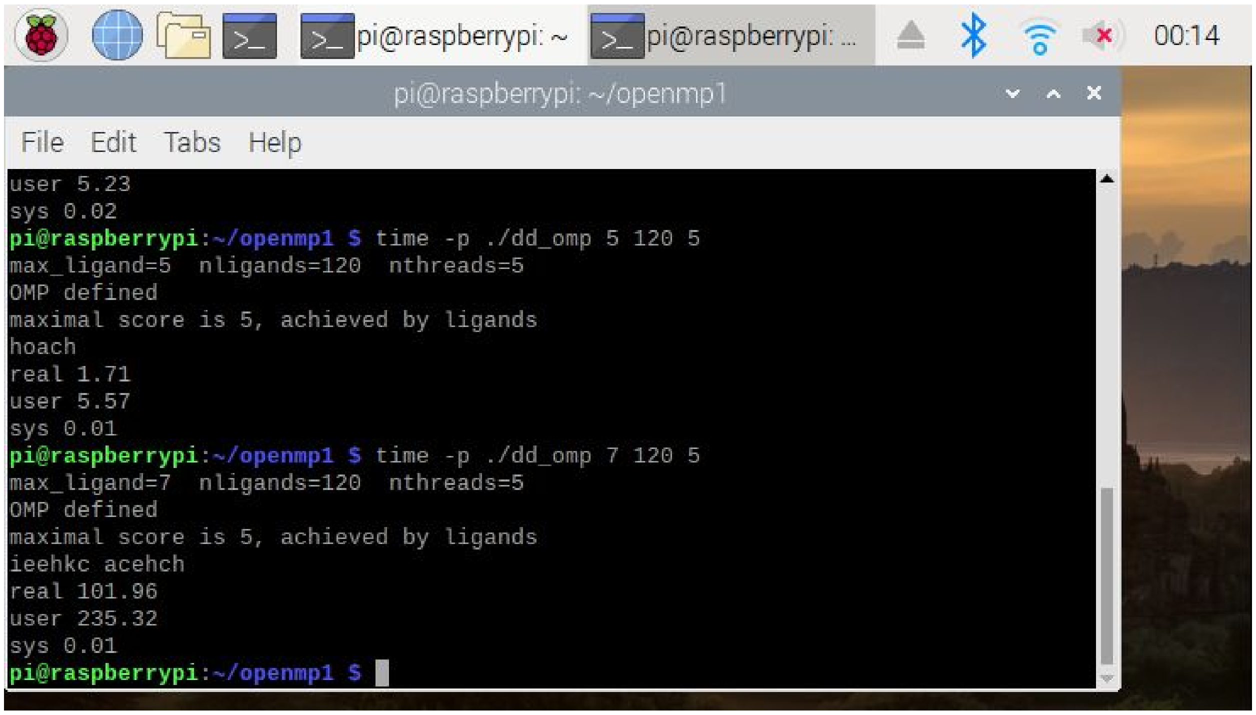
After I downloaded the required TBB library and created the directories. I ran the serial program and it took the longest to load because it computed the information sequentially. The best candidates were "acehch" and "ieehkc". I also played around with different numbers I could run the program with and I noticed that when I increased the number to 7, the program ran faster. However the maximal score was still 5.



```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 1
max_ligand=5  nligands=120  nthreads=1
OMP defined
maximal score is 5, achieved by ligands
hoach
real 4.32
user 4.30
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 2
max_ligand=5  nligands=120  nthreads=2
OMP defined
maximal score is 5, achieved by ligands
hoach
real 3.27
user 5.45
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 3
max_ligand=5  nligands=120  nthreads=3
OMP defined
```

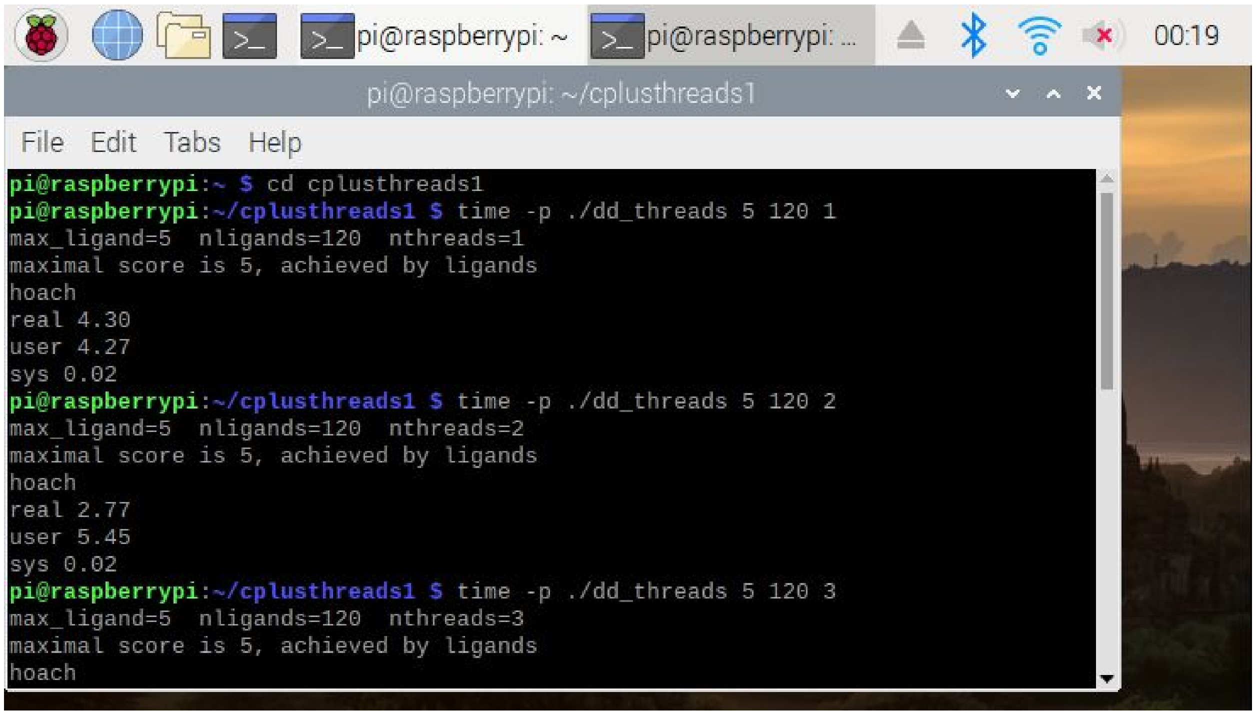


```
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 3
max_ligand=5  nligands=120  nthreads=3
OMP defined
maximal score is 5, achieved by ligands
hoach
real 2.38
user 5.16
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 4
max_ligand=5  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
hoach
real 2.03
user 5.23
sys 0.02
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 5
max_ligand=5  nligands=120  nthreads=5
```

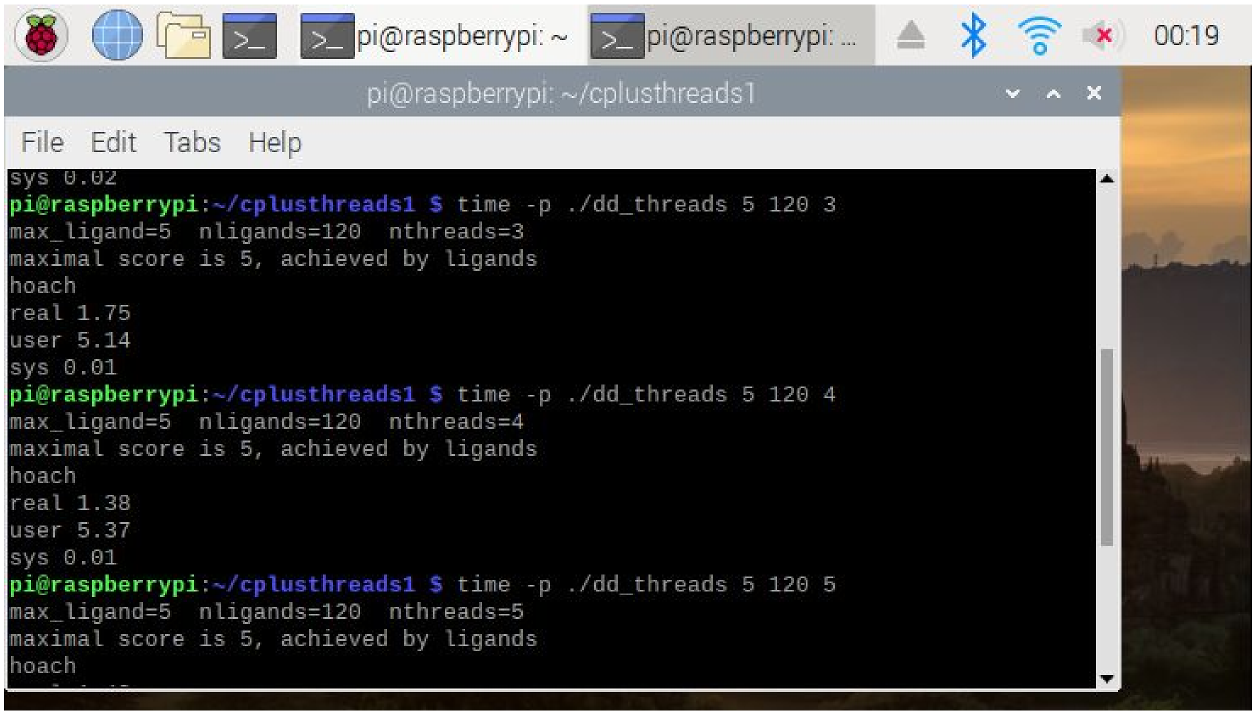


```
user 5.23
sys 0.02
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 5
max_ligand=5  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
hoach
real 1.71
user 5.57
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 101.96
user 235.32
sys 0.01
pi@raspberrypi:~/openmp1 $
```

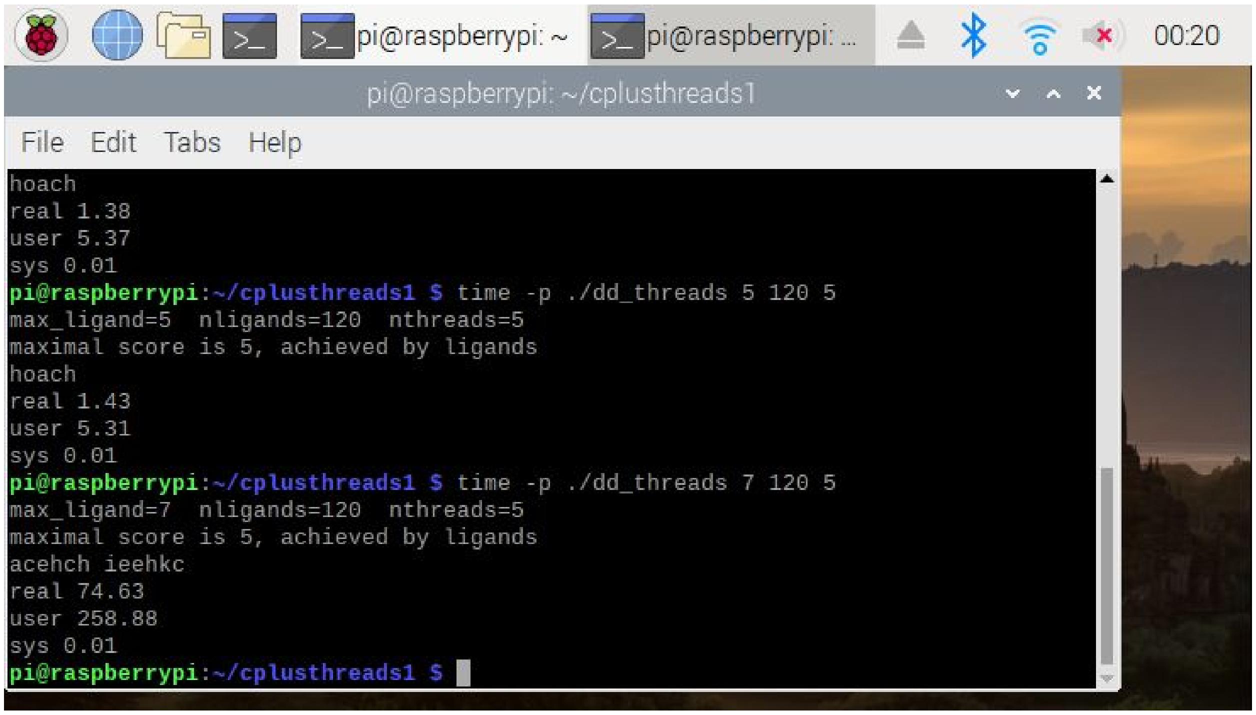
In the OpenMp implementation we parallelize and use shared-memory multithreading which is just a little bit slower than the C++11 Threads implementation and faster than the sequential implementation. In the screenshots above I ran the program with 1-5 threads and one with 7 ligands. I concluded when you increase the number of threads, the time it finishes is much faster. Also when I increased the number of ligands, it took longer to run.



```
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 1
max_ligand=5  nligands=120  nthreads=1
maximal score is 5, achieved by ligands
hoach
real 4.30
user 4.27
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 2
max_ligand=5  nligands=120  nthreads=2
maximal score is 5, achieved by ligands
hoach
real 2.77
user 5.45
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 3
max_ligand=5  nligands=120  nthreads=3
maximal score is 5, achieved by ligands
hoach
```



```
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 3
max_ligand=5  nligands=120  nthreads=3
maximal score is 5, achieved by ligands
hoach
real 1.75
user 5.14
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 4
max_ligand=5  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
hoach
real 1.38
user 5.37
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 5
max_ligand=5  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
hoach
```



```
hoach
real 1.38
user 5.37
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 5
max_ligand=5  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
hoach
real 1.43
user 5.31
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7 120 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 74.63
user 258.88
sys 0.01
pi@raspberrypi:~/cplusthreads1 $
```

In the C++11 Threads Solution we are using C++11 threads which create and manage their own threads using a master-worker parallel programming pattern. The C++11 Threads Solution is also the fastest of the 3 different implementations. In the screenshots above I ran the program with 1-5 threads and one with 7 ligands. I concluded when you increase the number of threads, it finishes faster. It took longer to run when I ran it with 7 ligands.

<u>Parallel Programming Skills Foundation: Joan Galicia</u>

- ➔ **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**
  - ◆ The basic steps in building a parallel program is identifying the set of tasks that can run concurrently, decomposing the data into tasks, assigning the tasks to processes where load is balanced, communication and synchronization of data to perform a task, and the mapping of that task to a processor.
  - ◆ Example of a parallel program is drug processing.
- ➔ **What is MapReduce?**
  - ◆ A model derived from map and reduce combinators where it was developed within google as a mechanism for processing large amounts of raw data. Where it allows google engineers to perform simple computations while hiding the details of parallelization.
- ➔ **What is map and what is reduce?**

- Map takes as input a function and a sequences of values and applies the function to each value in the sequence
- Reduce combines all the elements of a sequence using a binary operation

➔ **Why MapReduce?**
- MapReduce combines the map and reduce functions together to take an input pair and produce a set of intermediate key pairs to then group them together all intermediate values associated with the same intermediate key and merges it together to form a smaller set of values

➔ **Show an example for MapReduce.**
- An example of MapReduce is large scale data analysis where it is the process of applying data analysis techniques to large amounts of data. It uses specialized algorithms to analyze and present information in a more significant form to the users.

➔ **Explain in your own words, how the MapReduce model is executed?**
- The MapReduce model executes by taking input data and collecting them into a set of data pairs where they can then be combined and converted into a smaller set of data pairs.

➔ **List and describe three examples that are expressed as MapReduce computations.**
- Distributed Grep- Finds a intermediate data and copies it to the output
- Count of URL Access Frequency- This example processes logs of web page requests and outputs a URL where all the requests are added together and emit a URL total count pair.
- Inverted Index- stores a mapping of content, such as words or numbers, to its locations in a document or a set of documents.

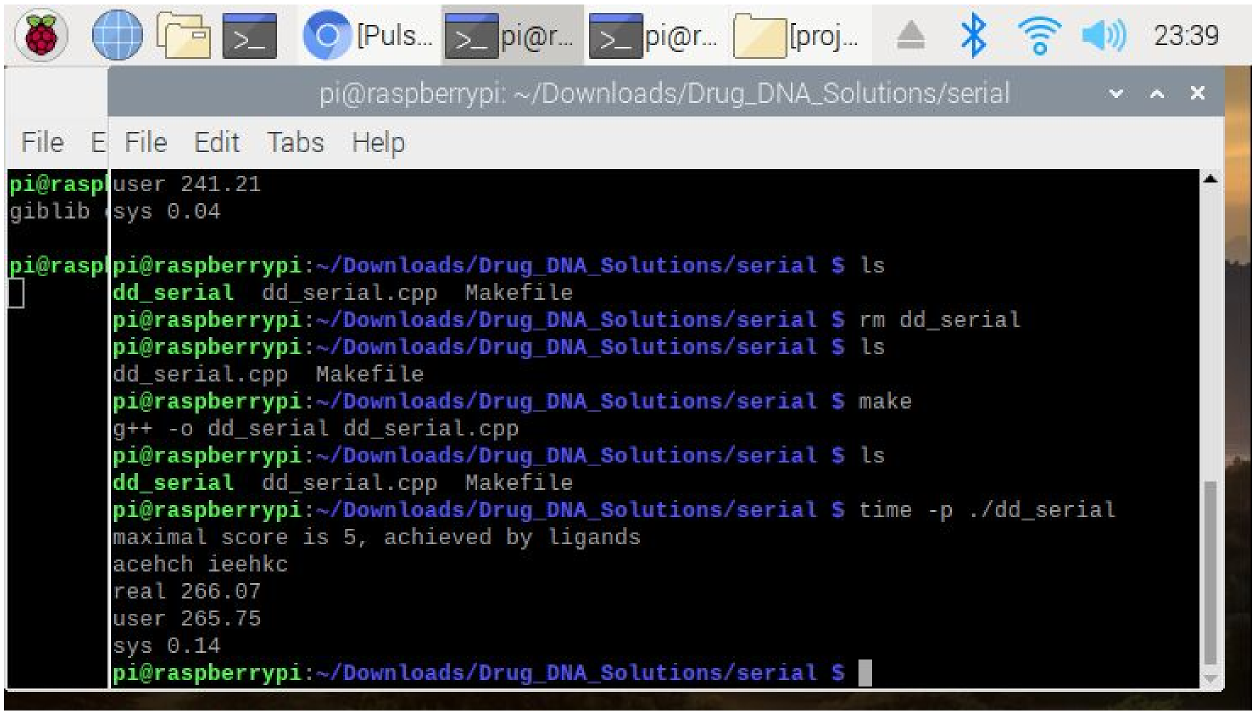
➔ **When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**
- OpenMP is used when there is shared memory in the program and so this library is able to handle the task of multithreading and apply OpenMP parallel features.
- MPI is used as a way of communicating with procedures in parallel programming and is used to allow a application to run in parallel across separate computers
- MapReduce is used when processing large amounts of data where the map function collects sets of data and the reduce function combines that data into a smaller set

➔ **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**
- Drug design is using computational capabilities to find the effectiveness of a drug by finding its ligands. Ligands are small molecules that can potentially be used as drugs, and finding these will allow medical professionals to design drugs so that they can fight against diseases and remain intact. The DNA problem is finding the

ligand for a drug associated with a disease by computational means and finding if the ligand binds to protein of the disease.

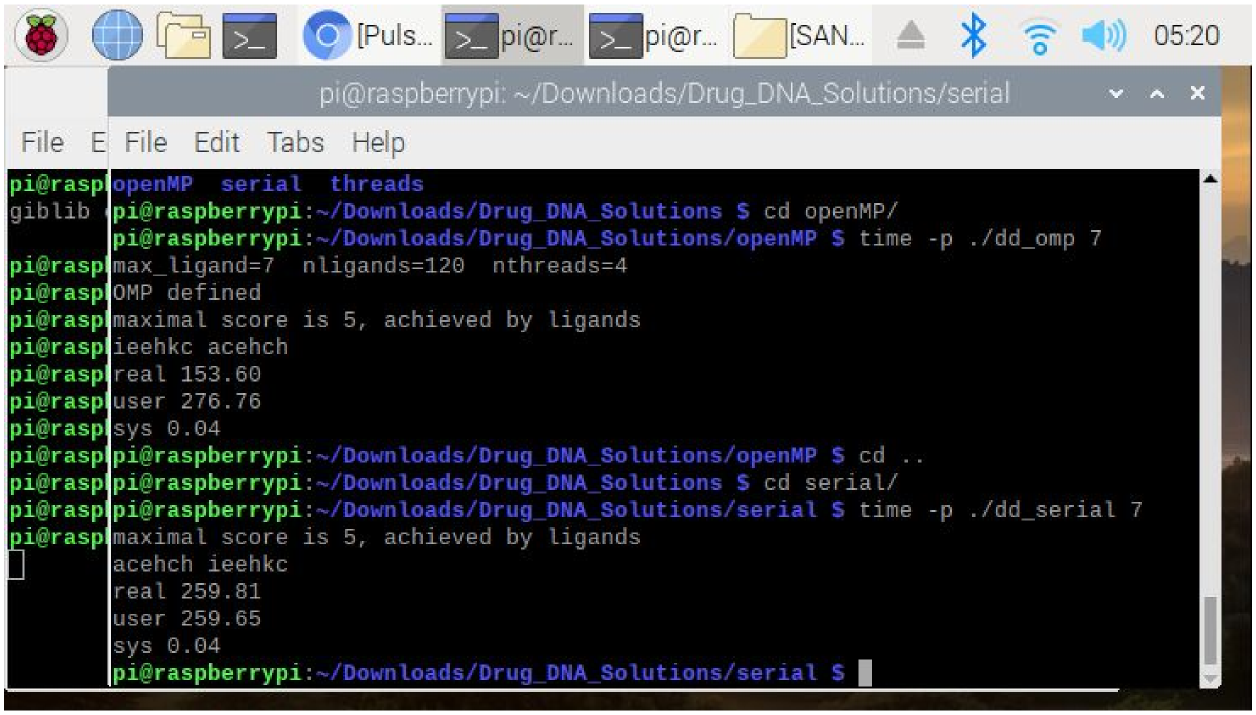
Parallel Programming Basics:

Part 1



```
user 241.21
sys 0.04

pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ ls
dd_serial  dd_serial.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ rm dd_serial
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ ls
dd_serial.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ make
g++ -o dd_serial dd_serial.cpp
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ ls
dd_serial  dd_serial.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 266.07
user 265.75
sys 0.14
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $
```

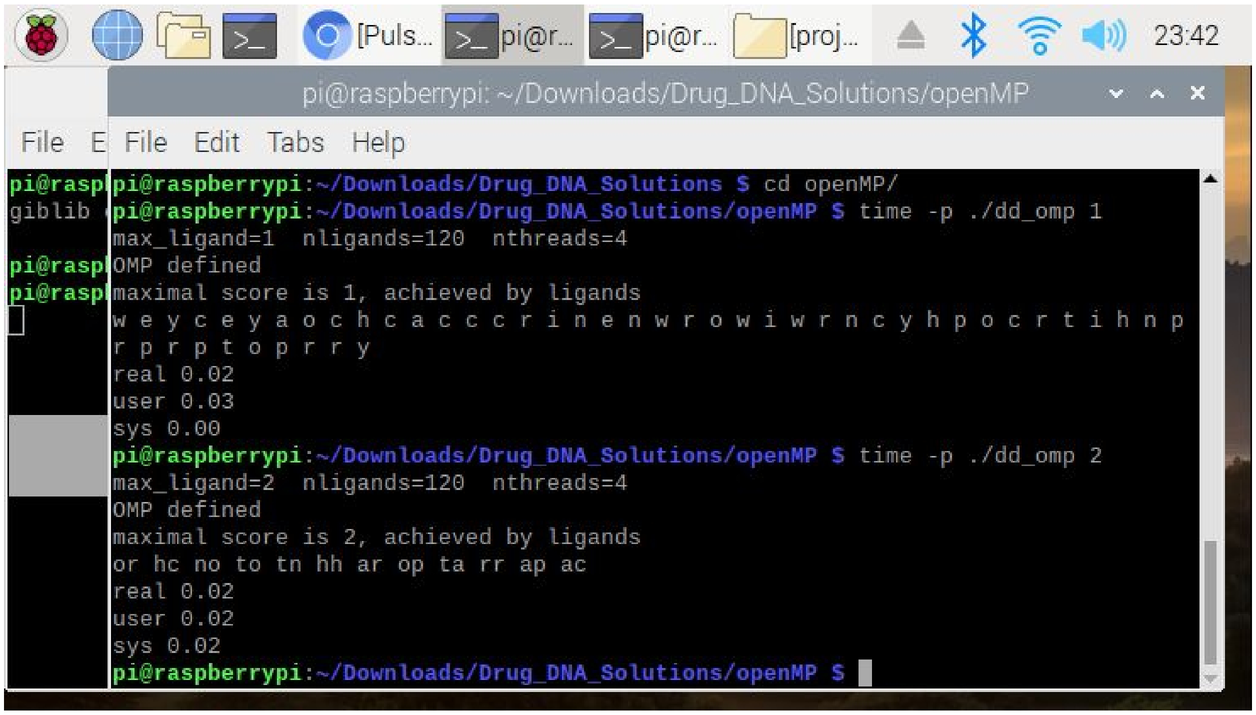
This is the output for the serial program. This was a surprising program as it took a long time, over 4 minutes, for it to display the output. This program shows how to design a drug by matching good candidates of ligands but this program executes sequentially.



```
openMP  serial  threads
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 153.60
user 276.76
sys 0.04
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ cd ..
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd serial/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 259.81
user 259.65
sys 0.04
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/serial $
```

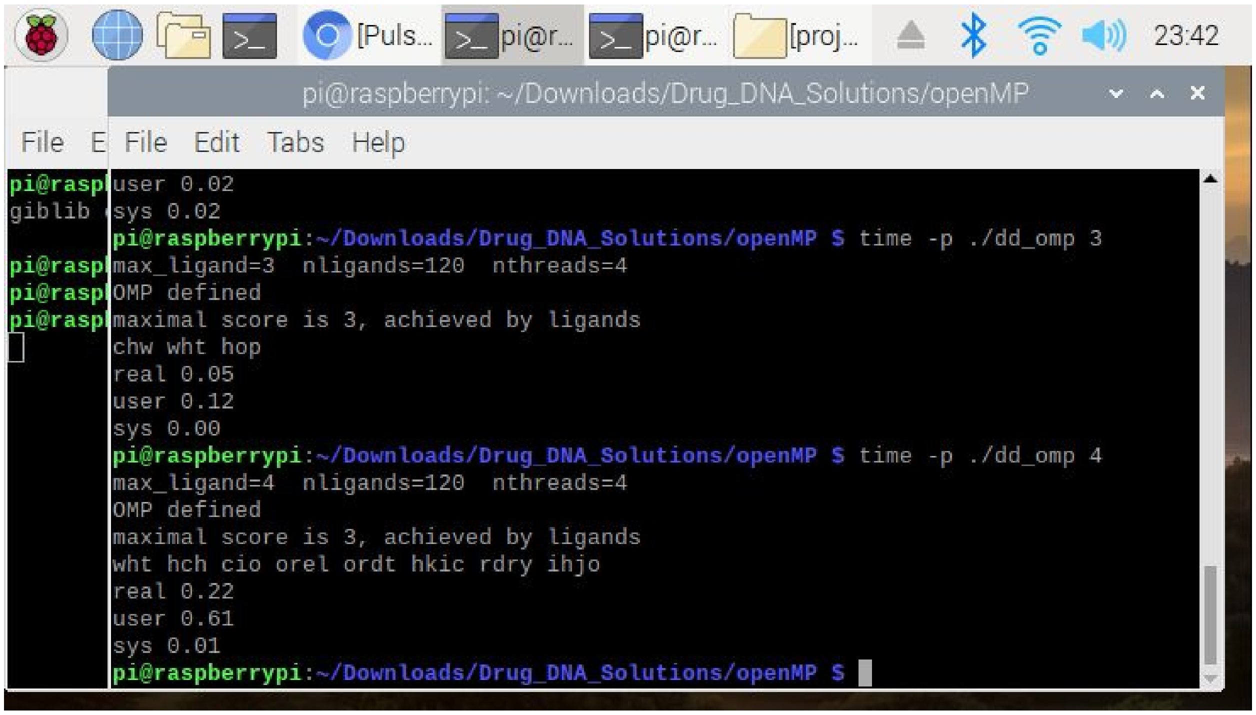
This program is the serial program but this one is told to use 7 threads. This also took a long time to load and shows how inefficient sequential programming is. This was the longest program to run when given 7 threads.

Part 2



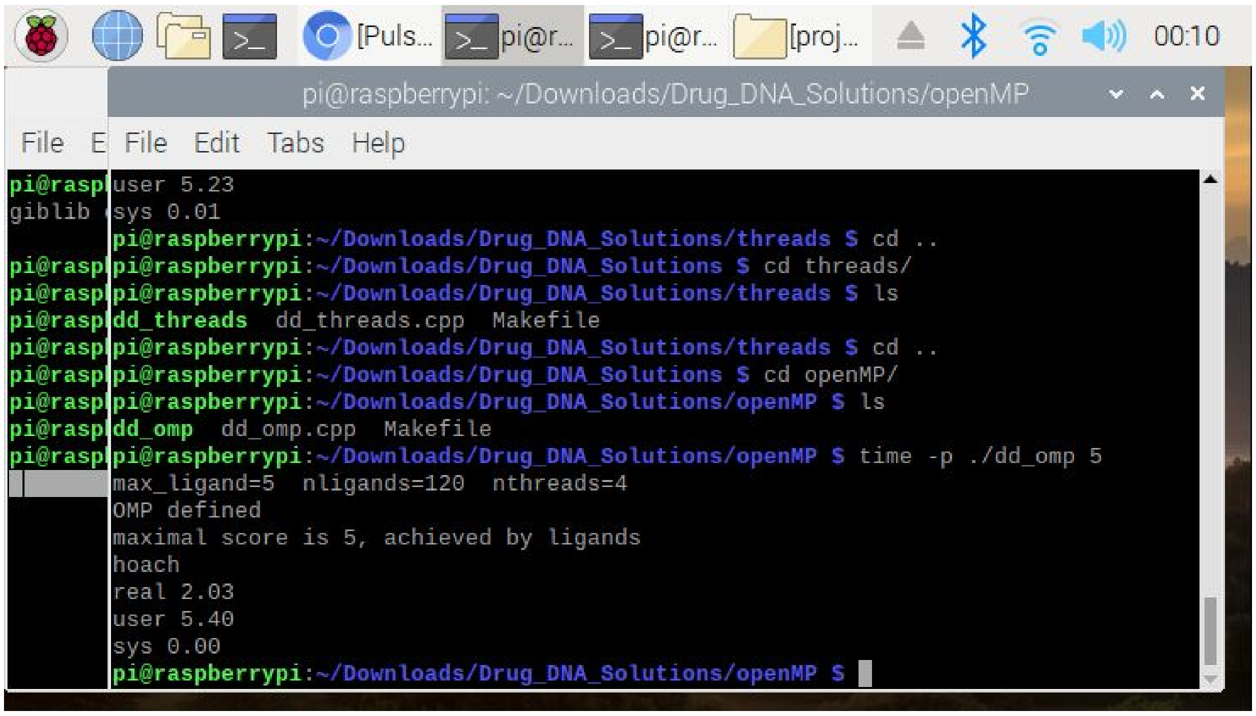
```
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 1
max_ligand=1  nligands=120  nthreads=4
OMP defined
maximal score is 1, achieved by ligands
w e y c e y a o c h c a c c c r i n e n w r o w i w r n c y h p o c r t i h n p
r p r p t o p r r y
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 2
max_ligand=2  nligands=120  nthreads=4
OMP defined
maximal score is 2, achieved by ligands
or hc no to tn hh ar op ta rr ap ac
real 0.02
user 0.02
sys 0.02
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $
```

This output shows the runtime of the OpenMP program where it was quick and efficient. The screenshot shows the number of threads the executed program had to use. To use this program, the tbb library needed to be installed into the raspberry pi. Unlike the serial program, this program requires the makefile to compile.



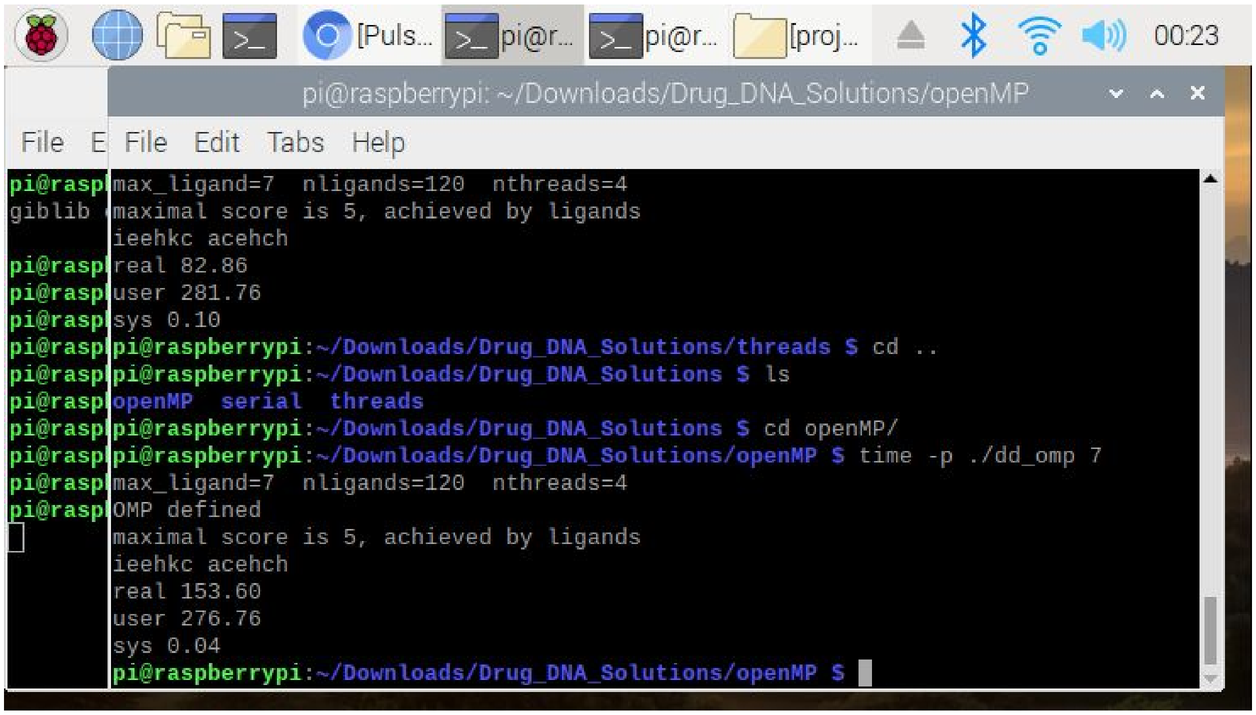
```
user 0.02
sys 0.02
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 3
max_ligand=3  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
chw wht hop
real 0.05
user 0.12
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 4
max_ligand=4  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
wht hch cio orel ordt hkic rdry ihjo
real 0.22
user 0.61
sys 0.01
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $
```

This screenshot is the runtime and execution of the OpenMP program where it has the program run with 3 (top) and 4 (bottom) threads. The more threads that were used the more time it took for the program to run.



```
user 5.23
sys 0.01
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ cd ..
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd threads/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ ls
dd_threads  dd_threads.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ cd ..
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ ls
dd_omp  dd_omp.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 5
max_ligand=5  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
hoach
real 2.03
user 5.40
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $
```

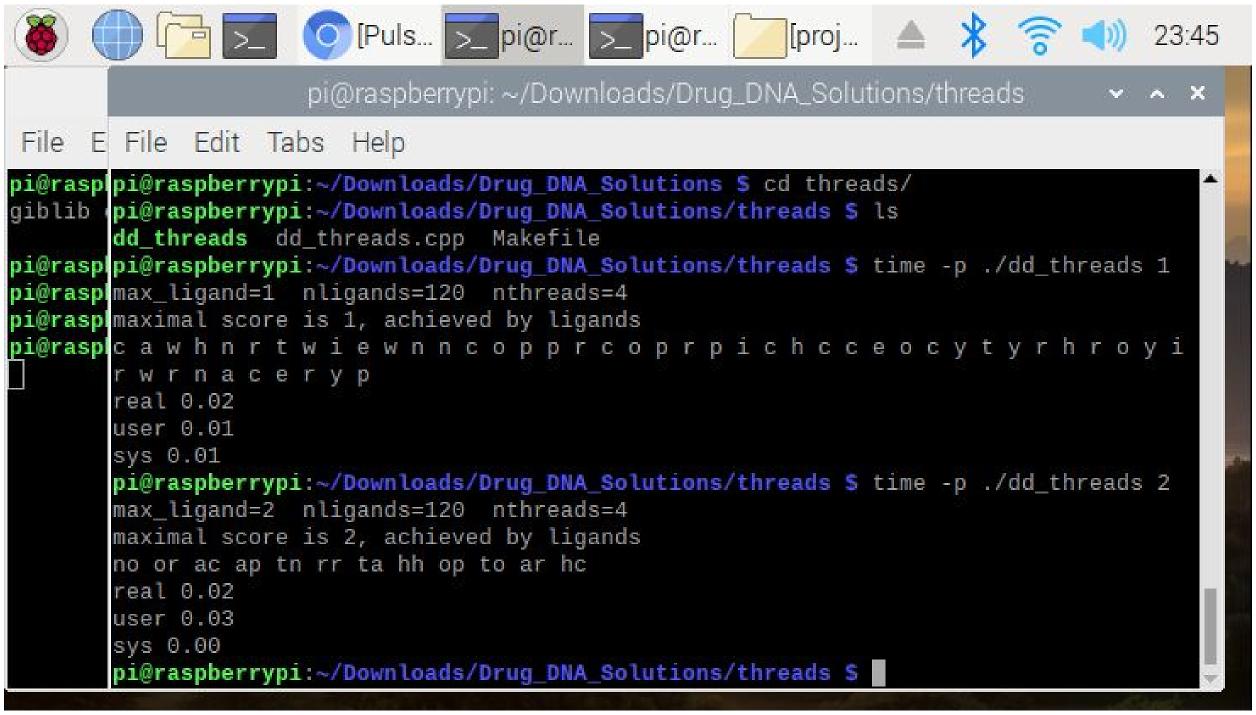
This screenshot is the runtime and execution of the OpenMP program at this point this program took a significantly longer time to execute.



```
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 82.86
user 281.76
sys 0.10
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ cd ..
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ ls
openMP  serial  threads
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 153.60
user 276.76
sys 0.04
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/openMP $
```

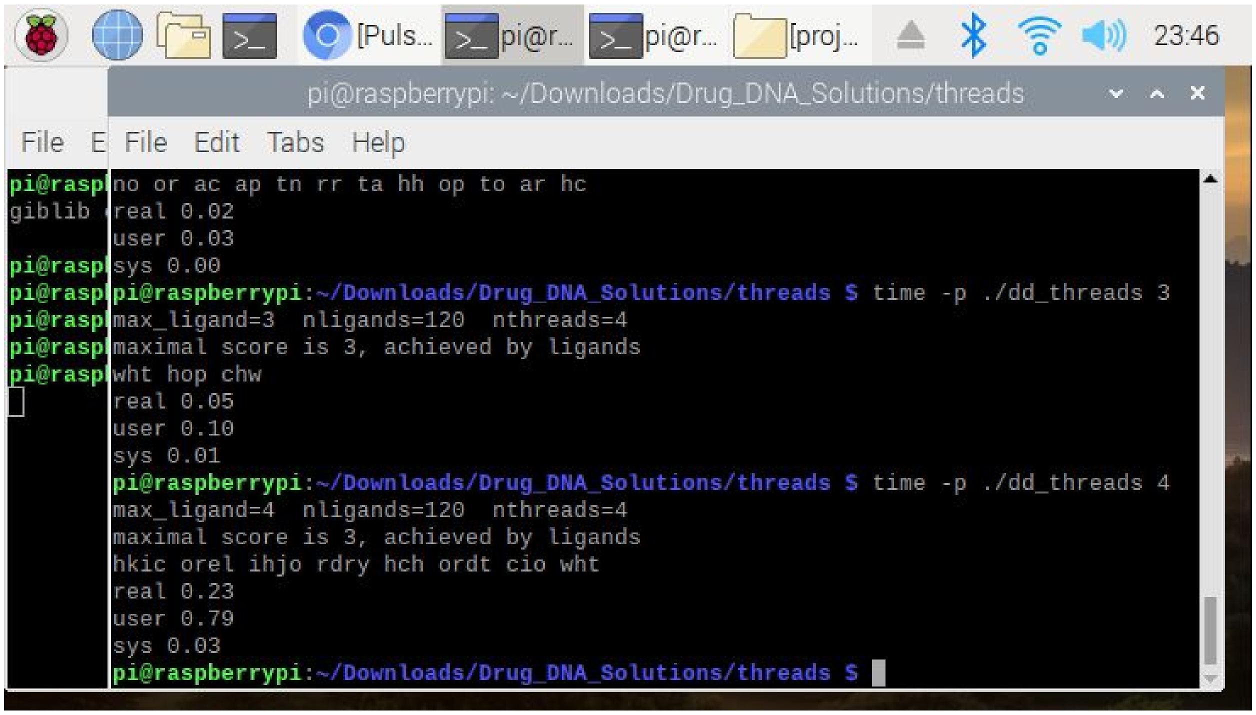
This screenshot is the runtime of the OpenMP where the program was told to use 7 threads and at this point for the OpenMP this was the longest time it took to run.

Part 3



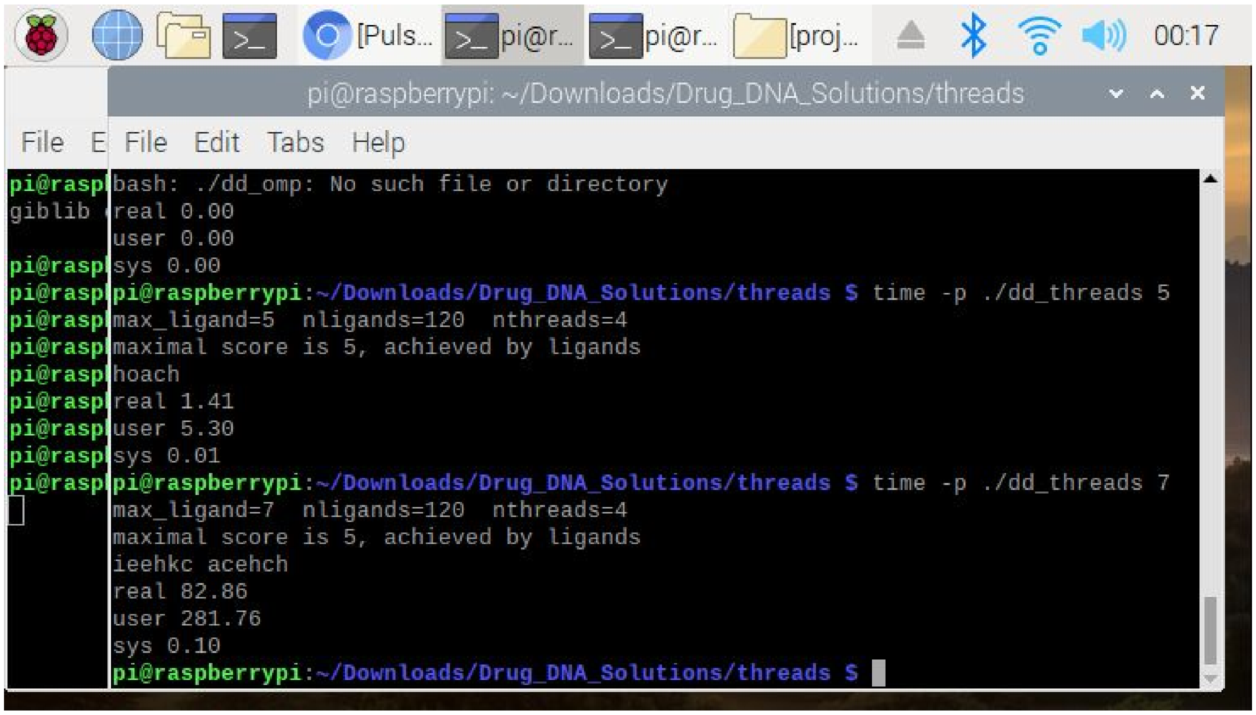
```
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions $ cd threads/
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ ls
dd_threads  dd_threads.cpp  Makefile
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 1
max_ligand=1  nligands=120  nthreads=4
maximal score is 1, achieved by ligands
c a w h n r t w i e w n n c o p p r c o p r p i c h c c e o c y t y r h r o y i
r w r n a c e r y p
real 0.02
user 0.01
sys 0.01
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 2
max_ligand=2  nligands=120  nthreads=4
maximal score is 2, achieved by ligands
no or ac ap tn rr ta hh op to ar hc
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $
```

This program is the execution and runtime of the threads program where it was told to use 1(top) and 2(bottom) threads. This program is comparable to the OpenMP program just that it has more code and was a more efficient program.



```
no or ac ap tn rr ta hh op to ar hc
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 3
max_ligand=3  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
wht hop chw
real 0.05
user 0.10
sys 0.01
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 4
max_ligand=4  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
hkic orel ihjo rdry hch ordt cio wht
real 0.23
user 0.79
sys 0.03
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $
```

This program is the execution and runtime of the threads program where it was told to use 3(top) and 4(bottom) threads. As shown the runtime is increasing as the threads increase.



```
bash: ./dd_omp: No such file or directory
real 0.00
user 0.00
sys 0.00
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 5
max_ligand=5  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
hoach
real 1.41
user 5.30
sys 0.01
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 82.86
user 281.76
sys 0.10
pi@raspberrypi:~/Downloads/Drug_DNA_Solutions/threads $
```

This program is the execution and runtime of the threads program where it was told to use 7 threads (bottom) and 5 threads(top). Threads running with 7 threads were by far the fastest to run and can be seen as significantly lower than the OpenMP Program.

Discussion Questions

Implementation	Time (s)
dd_serial	266.07
dd_omp	0.02
dd_threads	0.02

Implementation	Time (s) 2 Threads	Time (s) 3 Threads	Time (s) 4 Threads	Time (s) 5 Threads
dd_omp	0.02	0.05	0.22	2.03
dd_threads	0.02	0.05	0.23	1.37

1. **Which approach is the fastest?**

   a. dd_threads is the fastest approach.
2. **Determine the number of lines in each file (use wc -l).How does the C++11 implementation compare to the OpenMP implementations?**
   a. Serial has 170 lines, OpenMP has 193 lines, and Threads has 204 numbers of lines.
   b. The C++11 implementations are less efficient than the OpenMP implementations as OpenMP thread-pools for its Pragmas where it processes more of the tasks and is actually doing the work by avoiding shared-memory shuttling.
3. **Increase the number of threads to 5 threads. What is the run time for each?**

Implementation	Time (s) 5 Threads
dd_omp	2.03
dd_threads	1.37

4. **Increase the maximum ligand length to 7, and rerun each program.What is the run time for each?**

Implementation	Time (s) 7 Threads
dd_serial	259.81
dd_omp	153.60
dd_threads	82.86

<u>Parallel Programming Skills Foundation: Andre Nguyenphuc</u>

➔ **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**
   ◆ Identifying sets of tasks that can be processed concurrently. Then you have to decompose the data into equal-sized tasks. Those tasks are then assigned to processes where the load is balanced. Communications of data is established so that the task can be performed. Finally the task is mapped to the processor.
   ◆ A program using the master/worker technique to approximate pi
➔ **What is MapReduce?**
   ◆ A programming model which processes and generates big data sets with a parallel, distributed algorithm on a cluster.
➔ **What is map and what is reduce?**

- ◆ Map function takes data and then generates a set of intermediate key/value pairs, the reduce function then combines all the values that correlate with the same intermediate key.

➔ **Why MapReduce?**

- ◆ It is able to process large amounts of data in a quick amount of time.

➔ **Show an example for MapReduce.**

- ◆ Hadoop MapReduce is an example because it uses map and reduce through the Hadoop Distributed File System.

➔ **Explain in your own words how MapReduce model is executed?**

- ◆ The MapReduce model uses mapping and reducing to create different pairs from the given data and combine those pairs with similar pairs to produce a result that shows all the pairs within the given data.

➔ **List and describe three examples that are expressed as MapReduce computations.**

- ◆ Inverted Index which parses each document and shows a sequence of <word,document ID> pairs
- ◆ Distributed Grep which has the map function emits a line when it matches a given pattern and the reduce function copies the given intermediate data to the output.
- ◆ Count of URL Access Frequency this has the map function process logs of web page requests and produces <URL,1> and the reduce function combines all the values for the same URL and produces a <URL,total count> pair.

➔ **When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**

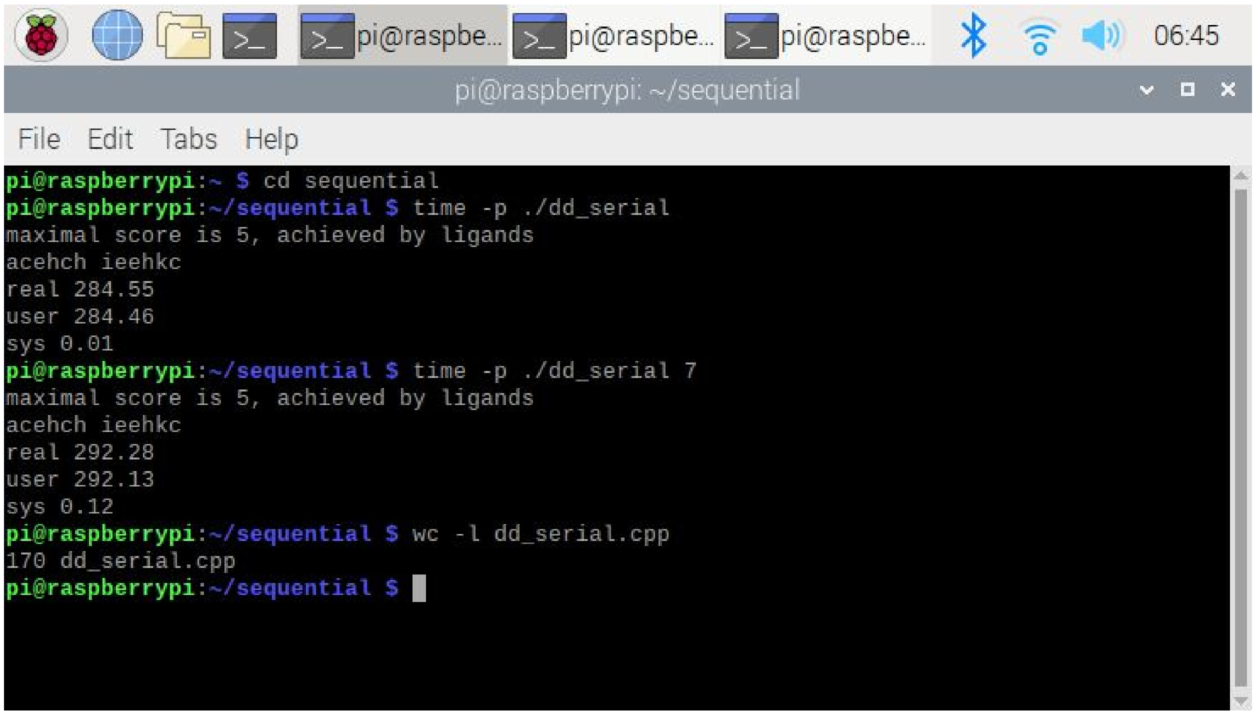
- ◆ OpenMP is used when you want shared memory parallelism in your code because it can split loops into multiple threads which allow each of them to handle a chunk of the loop's iterations.
- ◆ MPI is used when you want to develop parallel scientific applications. This is because it has tight synchronous code and well load balance.
- ◆ Hadoop MapReduce is used when you want to extract, transform, and load large amounts of data because it is able to chain maps and reduce to design complex problems.

➔ **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**

- ◆ Pharmaceutical companies design the medicine we use by finding ligands to change a protein's shape. This is because the shape of a protein is critical to the function it performs in someone's body. There are three steps to drug design software which is first generating different ligands to see if they fit the certain protein they want a fix to. Second they keep a score for each ligand to see how well it fits the protein and if it can generate the shape the company wants. The last step is to get the best scoring ligand to produce for medicine.

<u>Parallel Programming Basics: Andre Nguyenphuc</u>

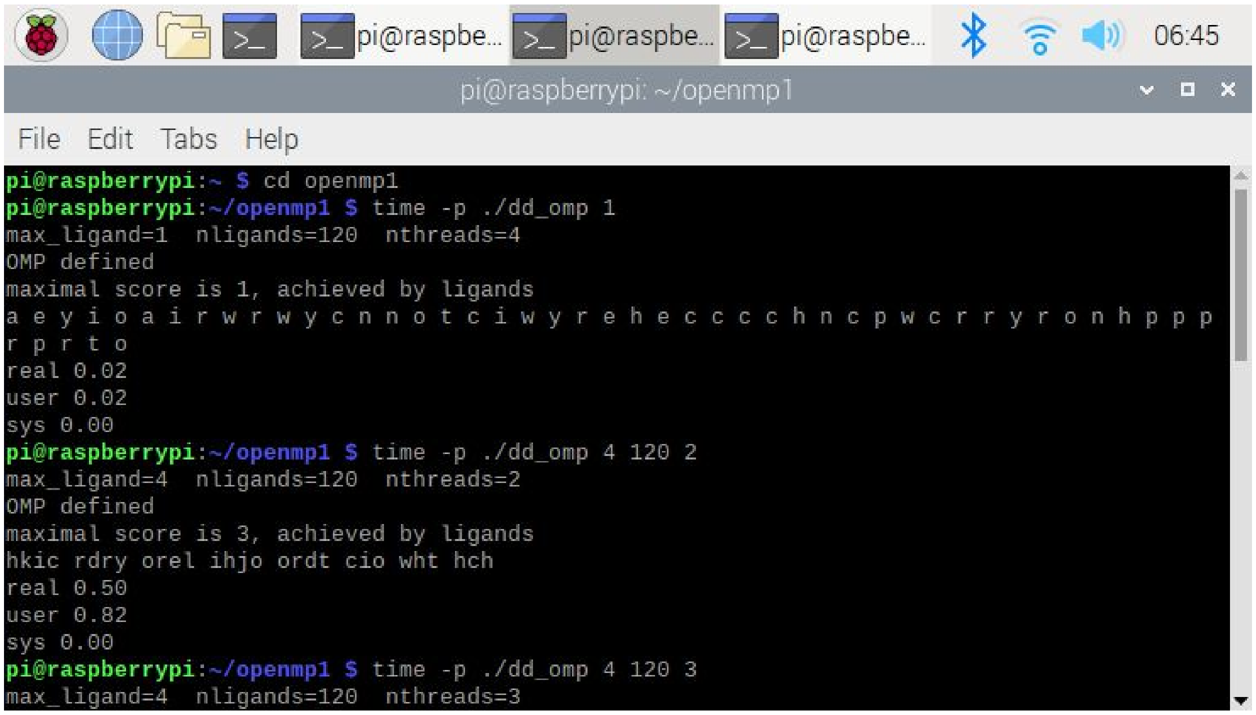
Part 1



```
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 284.55
user 284.46
sys 0.01
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 292.28
user 292.13
sys 0.12
pi@raspberrypi:~/sequential $ wc -l dd_serial.cpp
170 dd_serial.cpp
pi@raspberrypi:~/sequential $
```

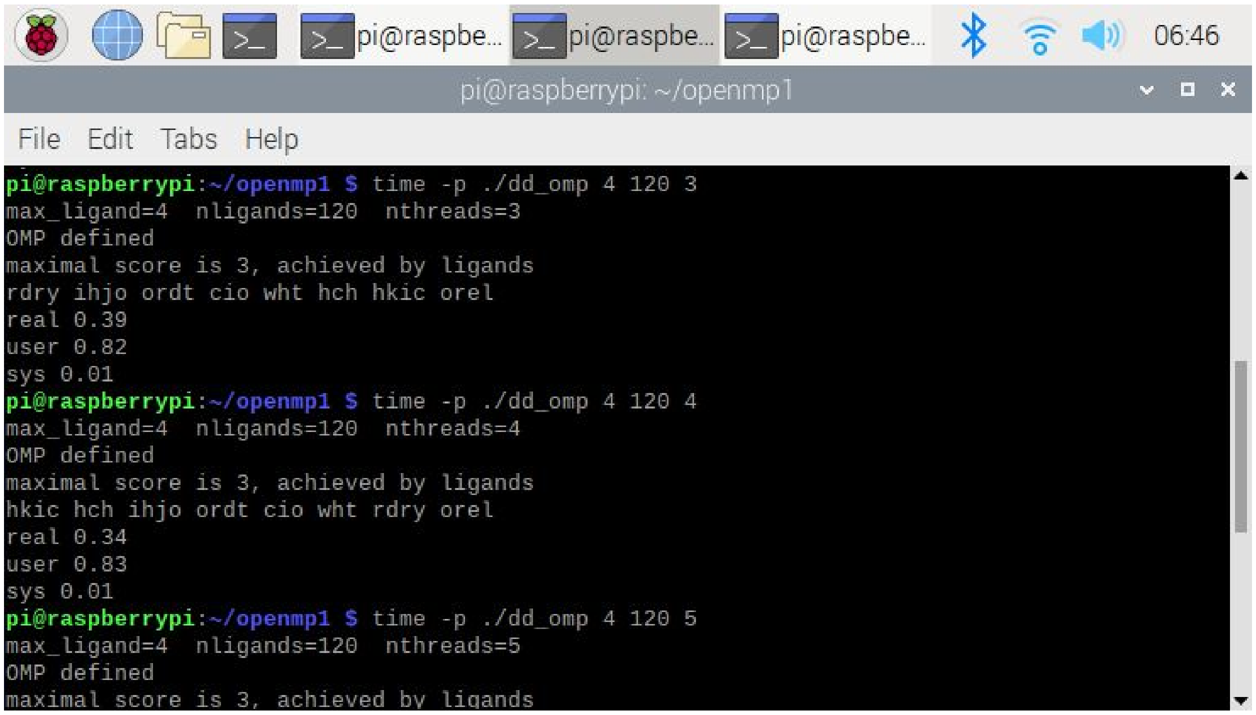
Created the directory and named it sequential and then I copied the files into it and compiled. I then ran the given example which took 284.55 seconds and 292.28 seconds. Sequential does not have nligands and nthreads like omp and threads. The last command is the line count. “acehch” and “ieehkc” are the best drug candidates.

Part 2

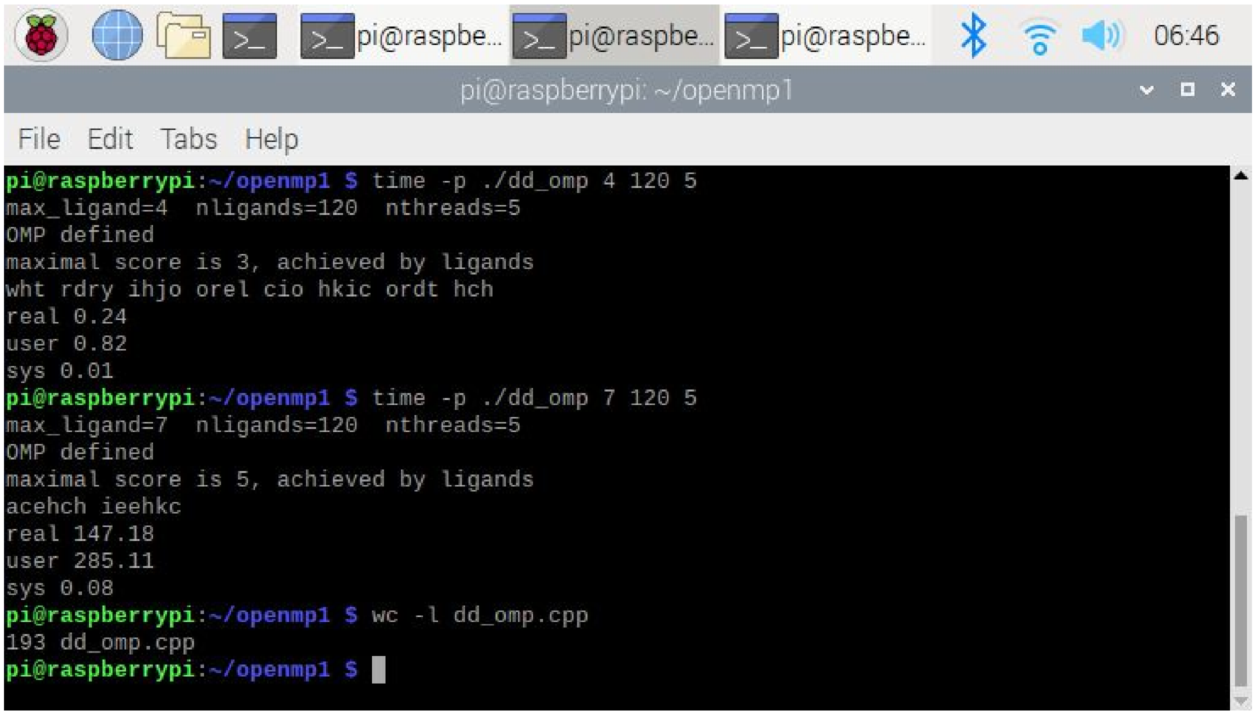


```
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1  nligands=120  nthreads=4
OMP defined
maximal score is 1, achieved by ligands
a e y i o a i r w r w y c n n o t c i w y r e h e c c c c h n c p w c r r y r o n h p p p
r p r t o
real 0.02
user 0.02
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 2
max_ligand=4  nligands=120  nthreads=2
OMP defined
maximal score is 3, achieved by ligands
hkic rdry orel ihjo ordt cio wht hch
real 0.50
user 0.82
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 3
max_ligand=4  nligands=120  nthreads=3
```

Created the directory and named it openmp1 and then I copied the files into it and compiled. I then ran the first given example which took 0.02 which is a significant decrease from sequential. I needed the tbb library to get this program to run.



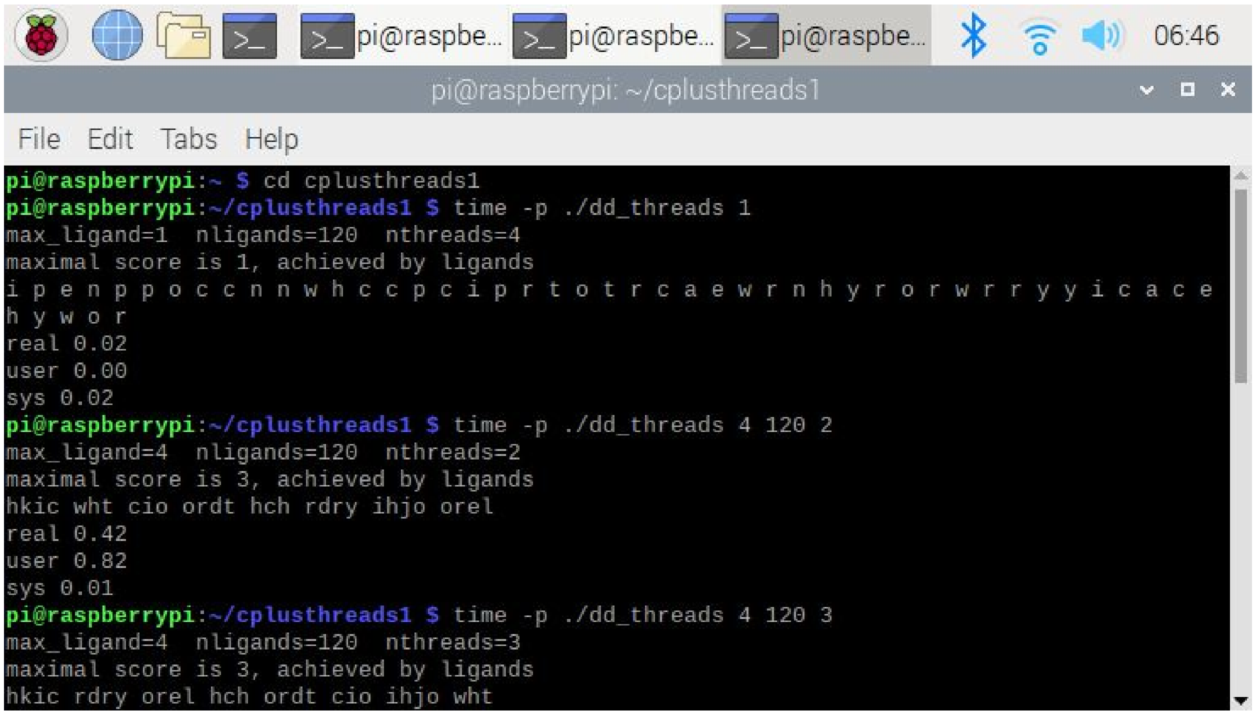
```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 3
max_ligand=4  nligands=120  nthreads=3
OMP defined
maximal score is 3, achieved by ligands
rdry ihjo ordt cio wht hch hkic orel
real 0.39
user 0.82
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 120 4
max_ligand=4  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
hkic hch ihjo ordt cio wht rdry orel
real 0.34
user 0.83
sys 0.01
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
```



```
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 orel cio hkic ordt hch
real 0.24
user 0.82
sys 0.01
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 147.18
user 285.11
sys 0.08
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
193 dd_omp.cpp
pi@raspberrypi:~/openmp1 $ 
```

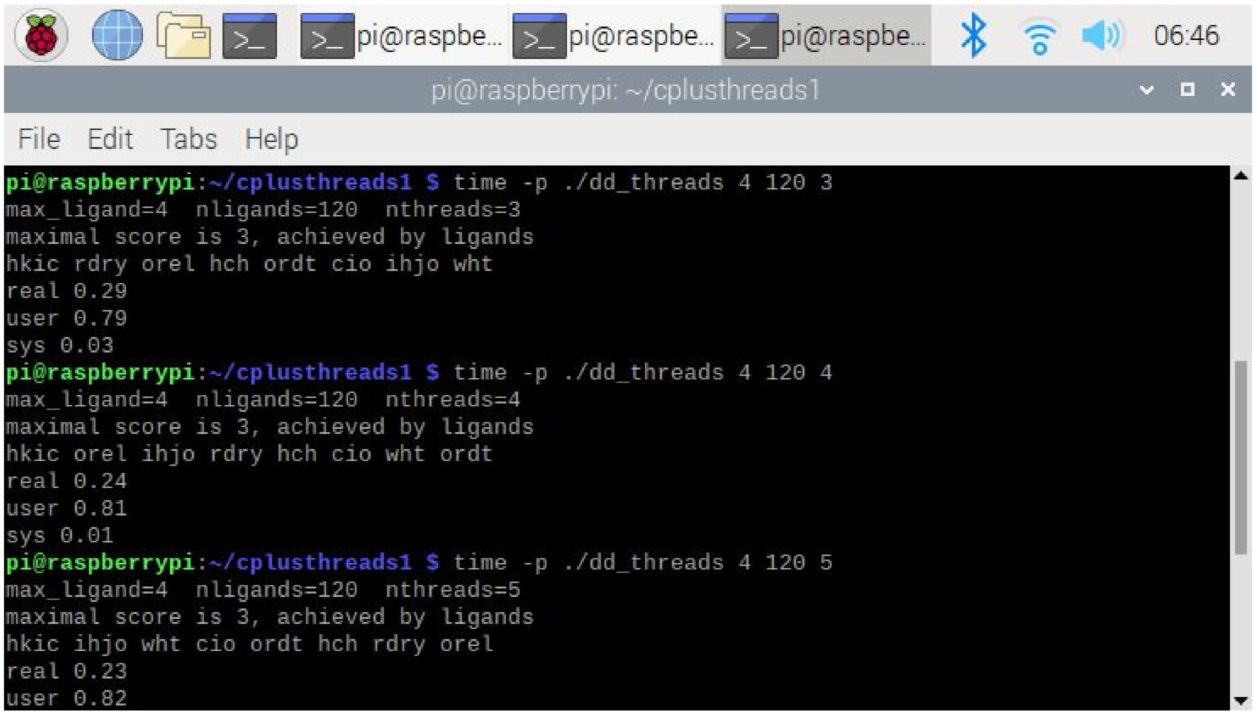
The two pictures above are the examples given and their runtime. I noticed that when typing the command the first number is max_lingard, the second number is nlingards, and the third number is nthreads. So for discussion questions I changed the numbers to the ones they asked so if it wanted to change the number of threads I would change the third number. Last command is the line count.

Part 3

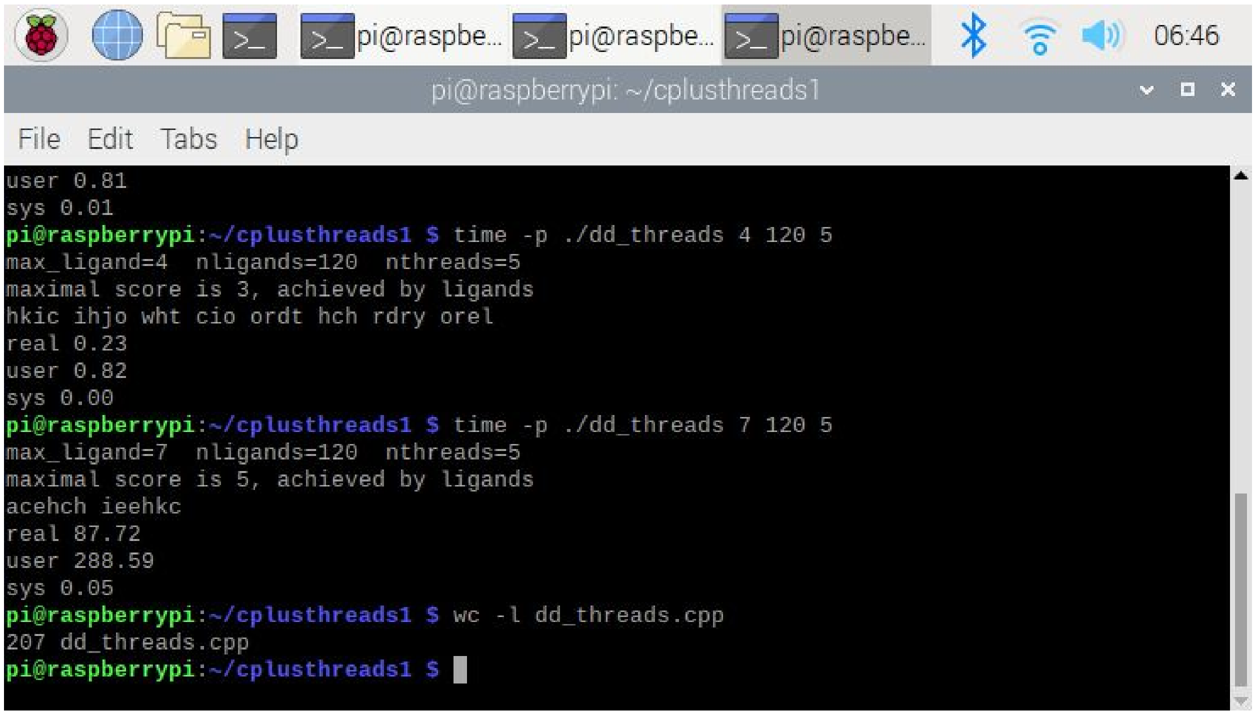


```
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1
max_ligand=1  nligands=120  nthreads=4
maximal score is 1, achieved by ligands
i p e n p p o c c n n w h c c p c i p r t o t r c a e w r n h y r o r w r r y y i c a c e
h y w o r
real 0.02
user 0.00
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 2
max_ligand=4  nligands=120  nthreads=2
maximal score is 3, achieved by ligands
hkic wht cio ordt hch rdry ihjo orel
real 0.42
user 0.82
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 3
max_ligand=4  nligands=120  nthreads=3
maximal score is 3, achieved by ligands
hkic rdry orel hch ordt cio ihjo wht
```

Created the directory and named it cplusthreads1 and then I copied the files into it and compiled.



```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 3
max_ligand=4  nligands=120  nthreads=3
maximal score is 3, achieved by ligands
hkic rdry orel hch ordt cio ihjo wht
real 0.29
user 0.79
sys 0.03
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 4
max_ligand=4  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
hkic orel ihjo rdry hch cio wht ordt
real 0.24
user 0.81
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 5
max_ligand=4  nligands=120  nthreads=5
maximal score is 3, achieved by ligands
hkic ihjo wht cio ordt hch rdry orel
real 0.23
user 0.82
```



```
user 0.81
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 120 5
max_ligand=4  nligands=120  nthreads=5
maximal score is 3, achieved by ligands
hkic ihjo wht cio ordt hch rdry orel
real 0.23
user 0.82
sys 0.00
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7 120 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 87.72
user 288.59
sys 0.05
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
207 dd_threads.cpp
pi@raspberrypi:~/cplusthreads1 $
```

The two pictures above are the examples given and their runtime. I noticed that threads have a lower runtime than omp. I also noticed as you increase the nthreads the runtime gets lower. However, when you increase max_lingand the runtime gets significantly longer. Last command is the line count.

Discussion Questions: Andre Nguyenphuc

Measure Runtime

Implementation	**Time**
dd_serial	284.55
dd_omp	0.02
dd_threads	0.02

On the document it says to do time -p ./dd_omp 2 and so on, but that changes max_lingard yet the chart asks for threads so I set max_lingard to 4 and did threads according to chart

Implementation (max_lingand = 4)	**Time(s) 2 Threads**	**Time(s) 3 Threads**	**Time(s) 4 Threads**
dd_omp	0.50	0.39	0.34
dd_threads	0.42	0.29	0.24

1. **Which approach is the fastest?**

   a. C++11 Threads(dd_threads) is the fastest

2. **Determine the number of lines in each file (use wc -l).How does the C++11 implementation compare to the OpenMP implementations?**
   a. Serial has 170 lines, omp has 193 lines, threads has 207 lines. C++11 (threads) has 14 more lines compared to OpenMP
3. **Increase the number of threads to 5 threads.What is the run time for each?**
   a.

Implementation (max_lingand = 4)	**Time(s) 5 Threads**
dd_omp	0.24
dd_threads	0.23

4. **Increase the maximum ligand length to 7, and rerun each program.What is the run time for each?**
   a.

Implementation (max_lingand = 7)	**Time(s)**
dd_serial	292.28
dd_omp	147.18 (5 Threads)
dd_threads	87.72 (5 Threads)

Parallel Programming Skills Foundation Miguel Romo:

➔ **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**
   ◆ Decomposition, Assignment, Orchestration, Mapping.
   ◆ An example could be making one large array into sub-arrays. By implementing a technique called master/worker, the master can initialize the array and split it up according to the number of available workers. The worker receives the subarray from the master, performs processing the subarray and returns results to master.

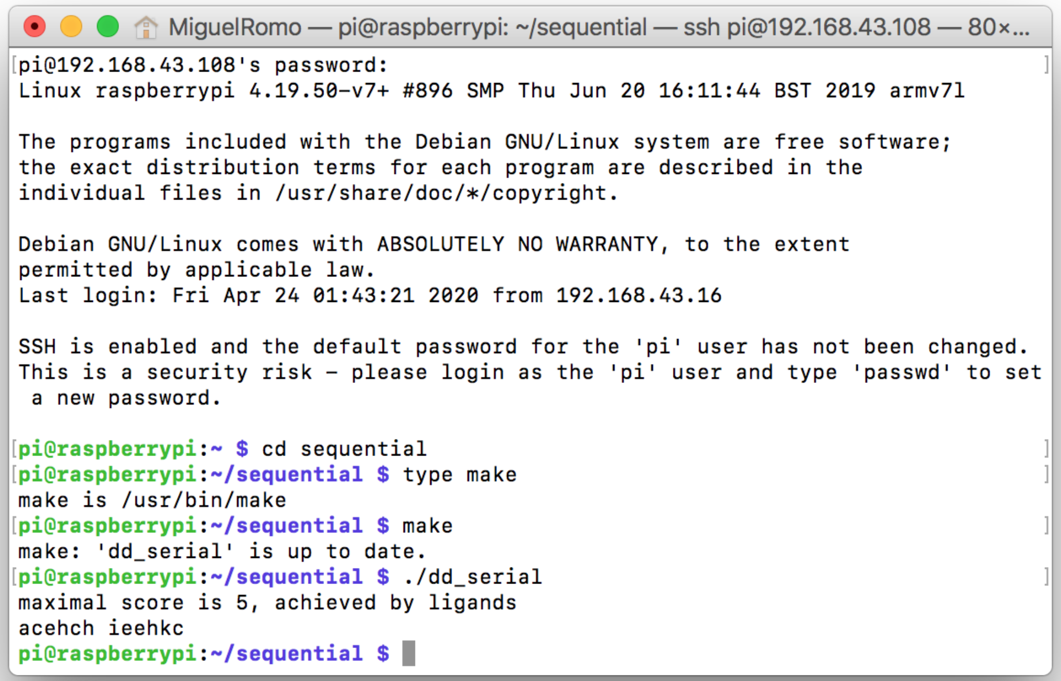
➔ **What is MapReduce?**
   ◆ MapReduce does what it says. Map takes inputs of a function and sequences of values and reduces combines all elements of sequence using binary operation.

➔ **What is map and what is reduce?**
   ◆ Map - takes an input pair and produces a set of intermediate key/value pairs.
   ◆ Reduce - accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values.

- **Why MapReduce?**
  - to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.
- **Show an example for MapReduce.**
  - Assume you have 7 files, and each file contains 3 columns (a key and a value in Hadoop terms) that represent a counties and the corresponding temperature recorded in that counties for the various measurement days. The counties are the key, and the temperature is the value. For example: (Fulton, 20). Out of all the data we have collected, you want to find the maximum temperature for each county across the data files. You can break this down into five map tasks, where each mapper works on one of the five files.
- **Explain in your own words how MapReduce model is executed?**
  - MapReduce program executes in three stages, map stage, shuffle stage, and reduce stage.
- **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 . The reduce function adds together all values for the same URL and emits a pair.
  - Reverse Web-Link Graph: The map function outputs 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:
- **When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**
  - OpenMp: If you want to introduce shared memory parallelism to your code, then OpenMP is an efficient directive based library that you could use.
  - MPI: Typically used to develop parallel scientific applications.
  - MapReduce(Hadoop): Two constructs that you can apply over your large data; map and reduce
- **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**
  - In the drug design and DNA problem The goal was to find a small molecule that was a good candidate for a drug. We do this by applying a map reduce strategy and implementing a Master-work program. We break down this problem into three steps. The first step is to generate the amount of molecules that need to be tested. The second step we apply a map () function that will compute a score with the highest successful pair. The final step is to use the reduce () to see which pairs score the highest. We do all this by simplifying the problem and using a simple

string base comparison. We use a sequential solution, followed by an openmp solution, and finally C++11thread solution. Our tests conclude that threads' solution run time is the quickest but also has the most lines of code, which is what the goal was. To prove how using Hadoop MapReduce is more efficient when running a program with big data.

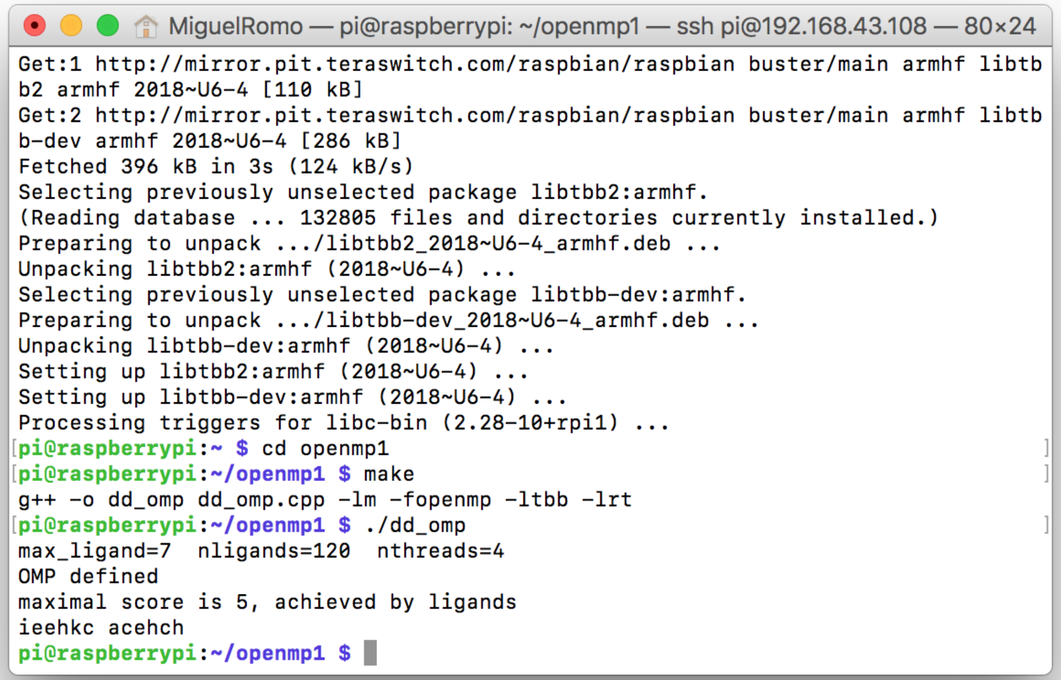
<u>Parallel Programming Basics Miguel Romo:</u>
<u>Part 1</u>


MiguelRomo — pi@raspberrypi: ~/sequential — ssh pi@192.168.43.108 — 80×...

```
pi@192.168.43.108's password:
Linux raspberrypi 4.19.50-v7+ #896 SMP Thu Jun 20 16:11:44 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Apr 24 01:43:21 2020 from 192.168.43.16

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ type make
make is /usr/bin/make
pi@raspberrypi:~/sequential $ make
make: 'dd_serial' is up to date.
pi@raspberrypi:~/sequential $ ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
pi@raspberrypi:~/sequential $
```

This is the screenshot of running the ./dd_serial. The results are, max score is 5.

Part 2

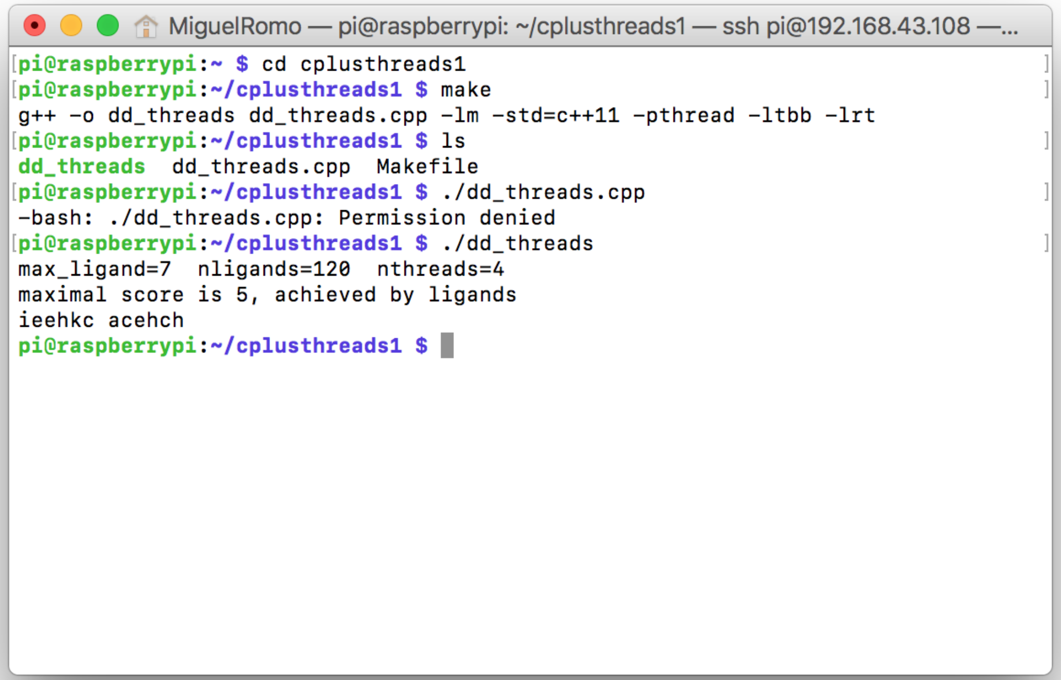


MiguelRomo — pi@raspberrypi: ~/openmp1 — ssh pi@192.168.43.108 — 80×24

```
Get:1 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtb
b2 armhf 2018~U6-4 [110 kB]
Get:2 http://mirror.pit.teraswitch.com/raspbian/raspbian buster/main armhf libtb
b-dev armhf 2018~U6-4 [286 kB]
Fetched 396 kB in 3s (124 kB/s)
Selecting previously unselected package libtbb2:armhf.
(Reading database ... 132805 files and directories currently installed.)
Preparing to unpack .../libtbb2_2018~U6-4_armhf.deb ...
Unpacking libtbb2:armhf (2018~U6-4) ...
Selecting previously unselected package libtbb-dev:armhf.
Preparing to unpack .../libtbb-dev_2018~U6-4_armhf.deb ...
Unpacking libtbb-dev:armhf (2018~U6-4) ...
Setting up libtbb2:armhf (2018~U6-4) ...
Setting up libtbb-dev:armhf (2018~U6-4) ...
Processing triggers for libc-bin (2.28-10+rpi1) ...
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ make
g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/openmp1 $ ./dd_omp
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
pi@raspberrypi:~/openmp1 $
```

This is a screenshot of running ./DD_OMP and it's results. Initially when running a program we received an error with the TBB Library. This error was resolved by installing the correct files so the ttb could be read on the pi. The results max_ligand=7, nligands=130, nthreads=4 and a max score is 5.

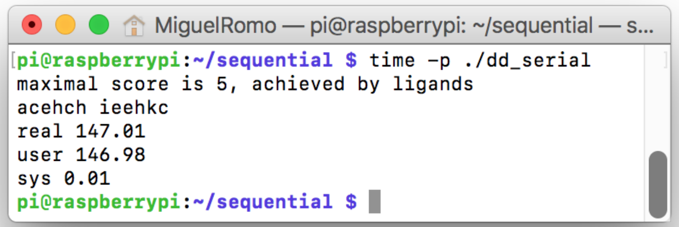
Part 3


MiguelRomo — pi@raspberrypi: ~/cplusthreads1 — ssh pi@192.168.43.108 —...

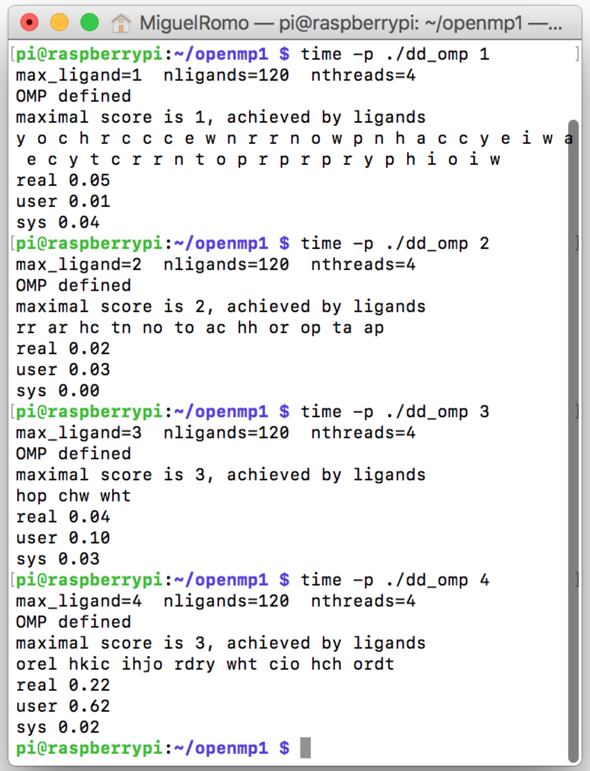
```
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ make
g++ -o dd_threads dd_threads.cpp -lm -std=c++11 -pthread -ltbb -lrt
pi@raspberrypi:~/cplusthreads1 $ ls
dd_threads  dd_threads.cpp  Makefile
pi@raspberrypi:~/cplusthreads1 $ ./dd_threads.cpp
-bash: ./dd_threads.cpp: Permission denied
pi@raspberrypi:~/cplusthreads1 $ ./dd_threads
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
pi@raspberrypi:~/cplusthreads1 $
```

Here is a screenshot after running ./DD_threads. The results max_ligand=7, nligands=120, nthreads=4 and a max score is 5.

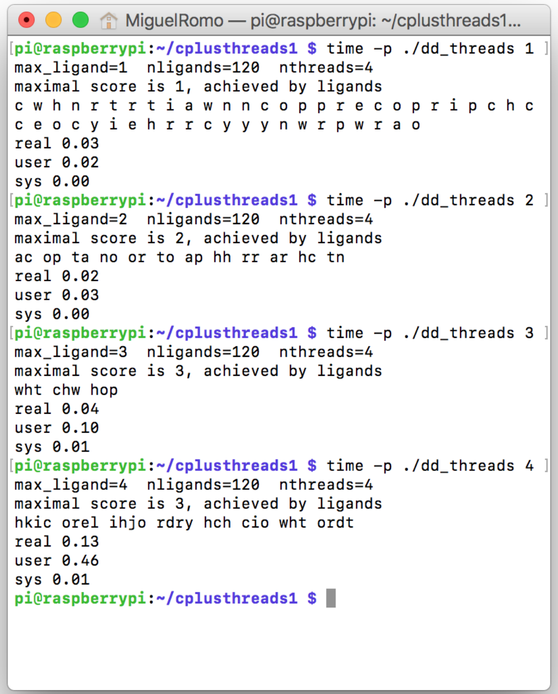
Measure Run-Time



```
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 147.01
user 146.98
sys 0.01
pi@raspberrypi:~/sequential $
```



```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1  nligands=120  nthreads=4
OMP defined
maximal score is 1, achieved by ligands
y o c h r c c c e w n r r n o w p n h a c c y e i w a
 e c y t c r r n t o p r p r p r y p h i o i w
real 0.05
user 0.01
sys 0.04
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 2
max_ligand=2  nligands=120  nthreads=4
OMP defined
maximal score is 2, achieved by ligands
rr ar hc tn no to ac hh or op ta ap
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 3
max_ligand=3  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
hop chw wht
real 0.04
user 0.10
sys 0.03
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=4  nligands=120  nthreads=4
OMP defined
maximal score is 3, achieved by ligands
orel hkic ihjo rdry wht cio hch ordt
real 0.22
user 0.62
sys 0.02
pi@raspberrypi:~/openmp1 $
```



```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1
max_ligand=1  nligands=120  nthreads=4
maximal score is 1, achieved by ligands
c w h n r t r t i a w n n c o p p r e c o p r i p c h c
c e o c y i e h r r c y y y n w r p w r a o
real 0.03
user 0.02
sys 0.00
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2  nligands=120  nthreads=4
maximal score is 2, achieved by ligands
ac op ta no or to ap hh rr ar hc tn
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 3
max_ligand=3  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
wht chw hop
real 0.04
user 0.10
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4
max_ligand=4  nligands=120  nthreads=4
maximal score is 3, achieved by ligands
hkic orel ihjo rdry hch cio wht ordt
real 0.13
user 0.46
sys 0.01
pi@raspberrypi:~/cplusthreads1 $
```

Implementation	**Time**
dd_serial	147.01
dd_omp	0.05
dd_threads	0.03

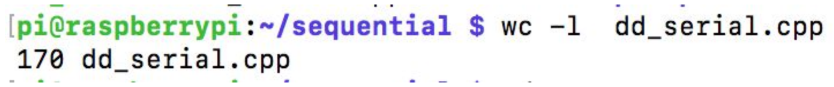
Implementation	**Time(s) 2 Threads**	**Time(s) 3 Threads**	**Time(s) 4 Threads**
dd_omp	0.02	0.04	0.24
dd_threads	0.02	0.04	0.13

Discussion Questions

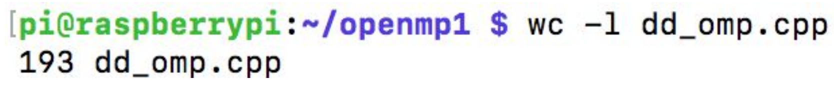
1. **Which approach is the fastest?**

   dd_threads is the fastest approach.

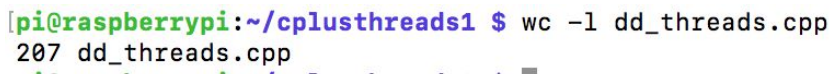
2. **Determine the number of lines in each file (use wc -l).How does the C++11 implementation compare to the OpenMP implementations?**



```
pi@raspberrypi:~/sequential $ wc -l  dd_serial.cpp
170 dd_serial.cpp
```



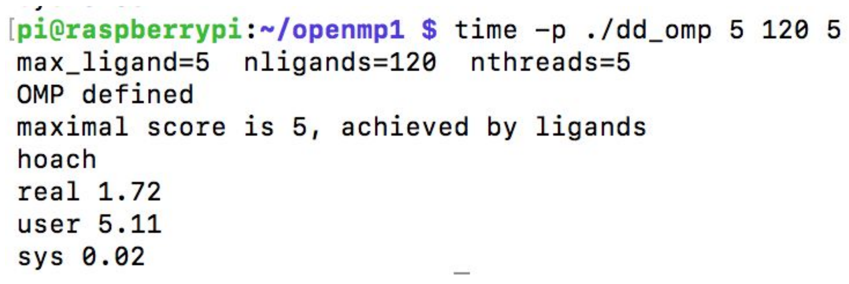
```
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
193 dd_omp.cpp
```



```
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
207 dd_threads.cpp
```

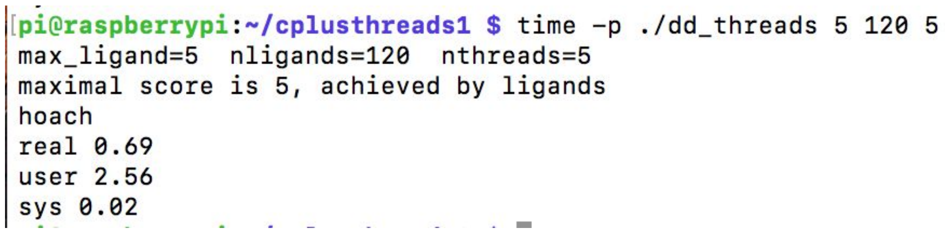
C++11(Thread) has 14 more lines than OpenMP(omp)

3. **Increase the number of threads to 5 threads. What is the run time for each?**



```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5 120 5
max_ligand=5  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
hoach
real 1.72
user 5.11
sys 0.02
```

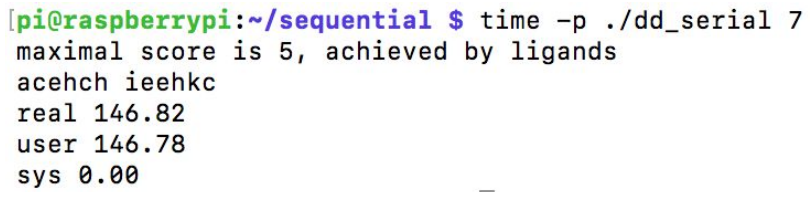
Runtime = 1.72



```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5 120 5
max_ligand=5  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
hoach
real 0.69
user 2.56
sys 0.02
```

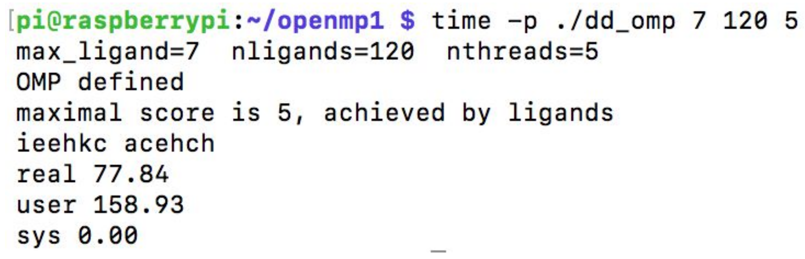
Runtime = 0.69

4. **Increase the maximum ligand length to 7, and rerun each program.What is the run time for each?**



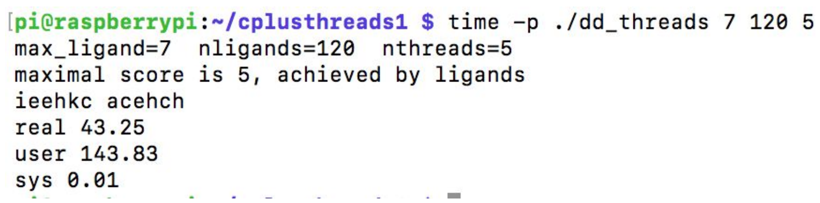
```
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 146.82
user 146.78
sys 0.00
```

Runtime = 146.82



```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 77.84
user 158.93
sys 0.00
```

Runtime = 77.84



```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7 120 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
ieehkc acehch
real 43.25
user 143.83
sys 0.01
```

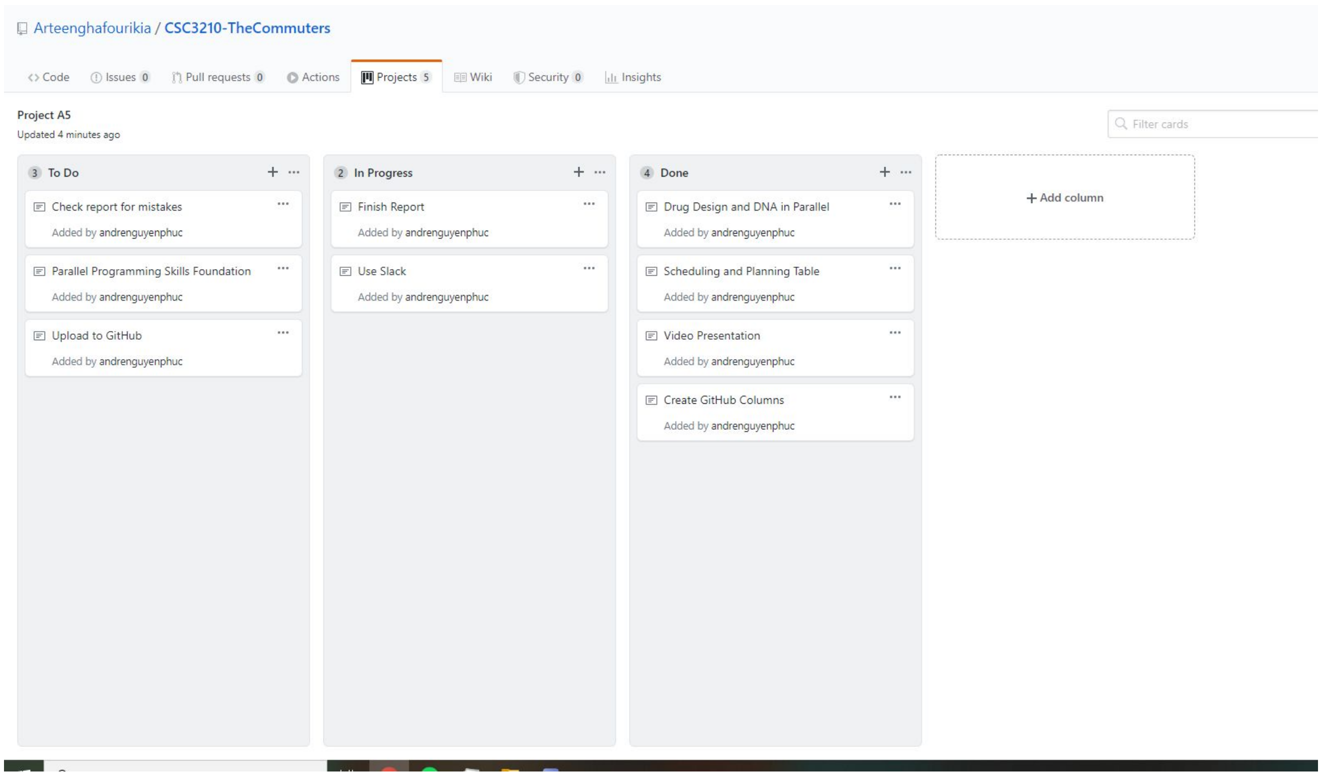
Runtime = 43.25

Appendix:

Slack:https://app.slack.com/client/TSWLWS9LK/CT8581ZU0

Youtube: https://www.youtube.com/watch?v=T9QMbkOWJDQ

GitHub: https://github.com/Arteenghafourikia/CSC3210-TheCommuters



Project A5
Updated 4 minutes ago

3 To Do
- Check report for mistakes — Added by andrenguyenphuc
- Parallel Programming Skills Foundation — Added by andrenguyenphuc
- Upload to GitHub — Added by andrenguyenphuc

2 In Progress
- Finish Report — Added by andrenguyenphuc
- Use Slack — Added by andrenguyenphuc

4 Done
- Drug Design and DNA in Parallel — Added by andrenguyenphuc
- Scheduling and Planning Table — Added by andrenguyenphuc
- Video Presentation — Added by andrenguyenphuc
- Create GitHub Columns — Added by andrenguyenphuc