Developing Soft and Parallel Programming Skills Using Project-Based Learning

Spring 2019, The Raspberry Five

Jacques Agbenu, Marcus Aqui, Bonnie Atelsek, Zach Benator, Jay Patel

Planning and Scheduling

We began communicating via Slack to allocate work among team members the day after Assignment 5 was released. It was established that Zach, as group coordinators had done in previous parts of the project, would write this report. He would also reformat the Pi after this project was complete. Jay volunteered to complete the parallel programming task. Marcus would shoot, edit, and upload the video. Bonnie would handle the GitHub and task assignment table, and Jaques and Bonnie agreed to work together on the parallel programming questions. The group immediately confirmed that we would meet on the Tuesday before the assignment was due (April 9) as usual.

However, Jay later informed us that he would unable to meet on April 9 due to a serious emergency at his job. Luckily this assignment was not due until Monday, April 15, so we were able to reschedule for Friday morning.

Jay brought the Pi on Friday morning having completed most of the parallel programming task. However, the issue of installing necessary libraries had still not been solved. Jay had already been in communication with members of several other groups who were also unable to solve the problem. Even as a team, we were unable to figure out what was wrong during our meeting. Jay was eventually able to solve the problem on his own after our meeting.

Assignee Name	Email	Task	Duration (hours)	Dependency	Due Date	Notes
Marcus Aqui	maqui1@student.gsu.edu	Edit and upload video	~1 hr	Project must be completed and video shot before editing can take place	4/15	
Jay Patel	jpatel118@student.gsu.edu	Parallel programming	~4 hr		4/13	
Bonnie Atelsek	batelsek1@student.gsu.edu	Update Github, maintain task table, answer parallel programming questions	~1.5 hr		4/13	
Jacqes Agbenu	jagbenu1@student.gsu.edu	Answer parallel programming questions	~1 hr		4/15	
Zach Benator **(Coordinator)**	zbenator1@student.gsu.edu	Write report, reformat Pi	~3 hr	Parallel programming and video must be complete before the report can be completed	4/15	

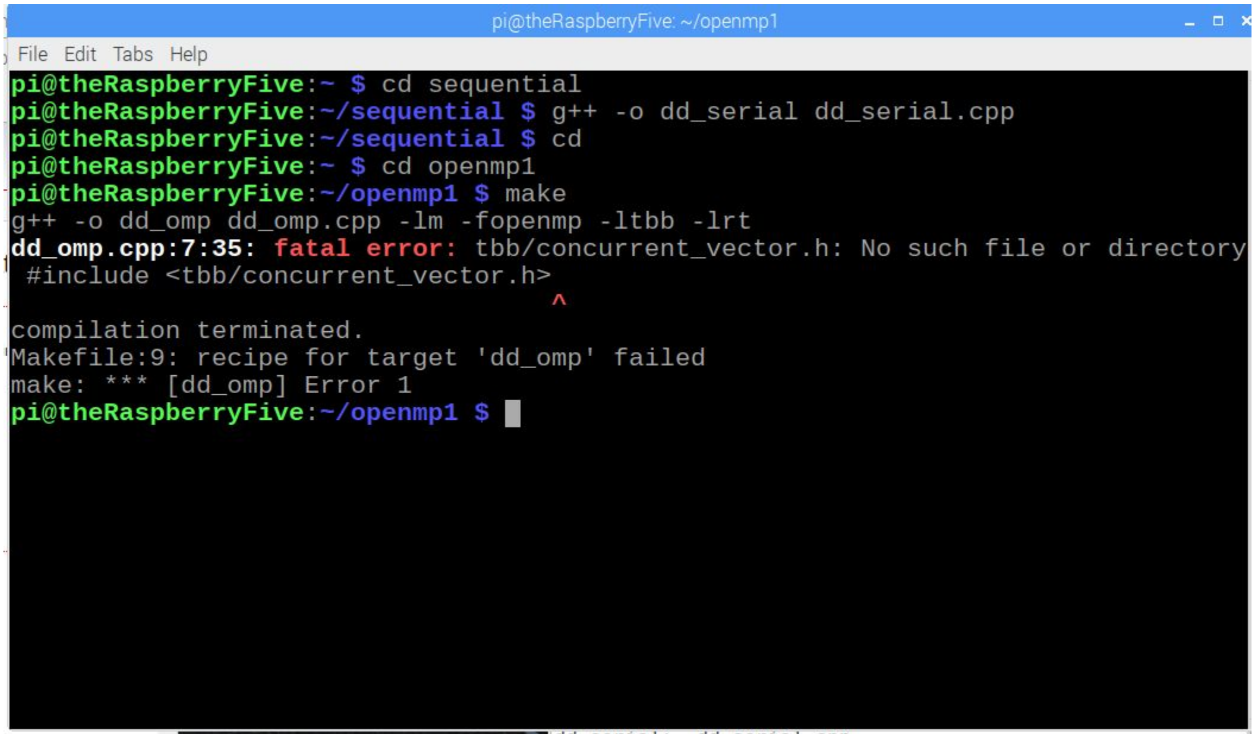
Parallel Programming Skills

a) Foundation

- What are the basic steps (show all steps) in building a parallel program? Show at least one example.
    - The first step in building a parallel program is to identify sets of tasks that can run concurrently and/or partitions of data that can be processed concurrently.
    - The user than sends the instructions to the master which then sends some of the tasks to the the workers to finish the tasks
    - When the workers are finished with the tasks they return it to the master to check their work
    - If the work is sufficient, then the master sends the finished task back to the user.
    - One example of this would be the use of a parallel program to approximate pi.
- What is MapReduce?
    - MapReduce is a structure and implementation for processing big data in parallel.
- What is map and what is reduce?
    - Both are functions specified by the user. Map is used to produce key/value pairs from other key/value pairs. Reduce is used to merge the created pairs later in the program based off of what key they correspond to.
- Why MapReduce?
    - MapReduce is useful for processing very large amounts of data and is scalable.
- Show an example for MapReduce.
    - MapReduce would be a very good way to crawl through documents or web request logs. Google uses it to manage its huge stores of data.
- Explain in your own words how MapReduce model is executed?
    - The user first prescribes the map and reduce functions to specify how they want the data handled. After that, the data is split up by the map function and processed in parallel. After the data has finished processing, it is sorted and relinked using the reduce function.
- List and describe three examples that are expressed as MapReduce computations.
    - An example of MapReduce would be the Distributed Grep. This is where the map function creates a line if it matches a given pattern. The reduce function becomes an identity function that copies the supplied intermediate data to the output.
    - Another example of MapReduce would be the Reverse-Web Link graph.The map function makes <target, source> pairs for each link in the target URl which is found in a page named "source". The reduce function adds the list of all source URLs associated with a given target URL and gives the pair : <target, list(source)>.
    - One final example of MapReduce would be a count of URL Access Frequency. The map function processes collections of web pages requests and gives the output <URL, 1>. The reduce function adds all the values for the same URL and outputs a <URL, total count> pair.
- When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?

    - OpenMp is useful for basic uses of parallelism in a programmer's code like loops because it is efficient, elegant, and relatively straightforward to implement. MPI is useful for scientific applications because they are tightly written and MPI is not good at handling outlier and non-fit cases. MapReduce is especially useful when handling large amounts of data, although it can also be used for scientific applications as opposed to MPI because it is better at handling faults, although it lacks somewhat in performance.
- In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.
    - Designing drugs is about making new ligands that will fit and change the already existing proteins of the body. The problem with that is that it takes a lot of work to be able to decipher which ligands go together to change a particular protein to get the needed result. There seems like there would be a lot of trial and error associated with finding the right ligands to change a certain protein. Keeping track of how well each ligand works with each protein would help the process go faster and might help with future testing.

b) Parallel Programming Basics: Drug Design and DNA in Parallel

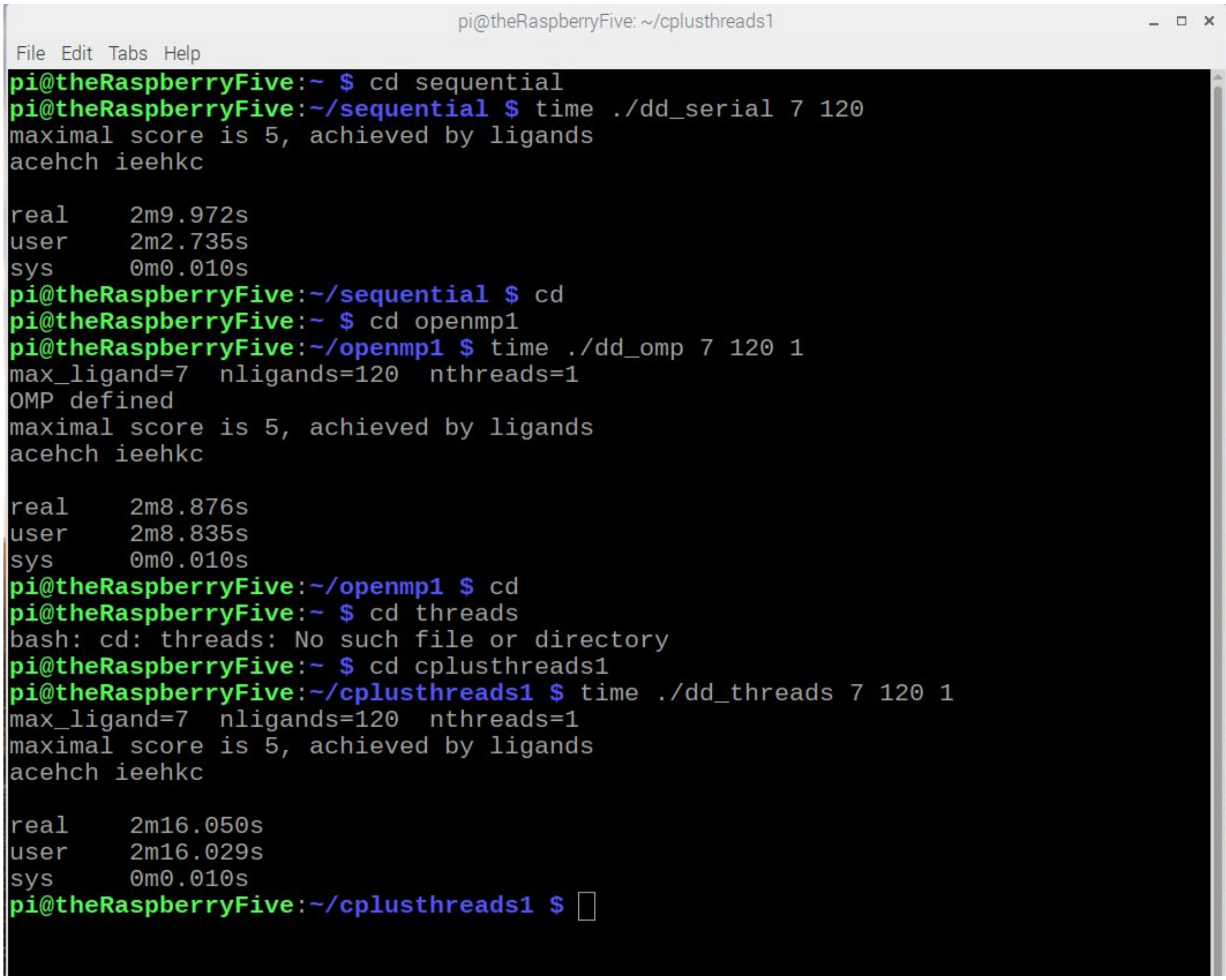


While following the directions provided in parallel programming tasks I ran into an error. While using the following command line at first “g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt” I got a “tbb” error which would not compile and was having trouble finding the tbb library. After some research I resolved this issue by manually installing tbb and setting flags and environment variables following this guide:

https://www.theimpossiblecode.com/blog/intel-tbb-on-raspberry-pi/

After installing the tbb libraries, I ran into an error again about “arm architecture version armv7-a”. which was solved by installing opencv and pip, using the following guide:

https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/



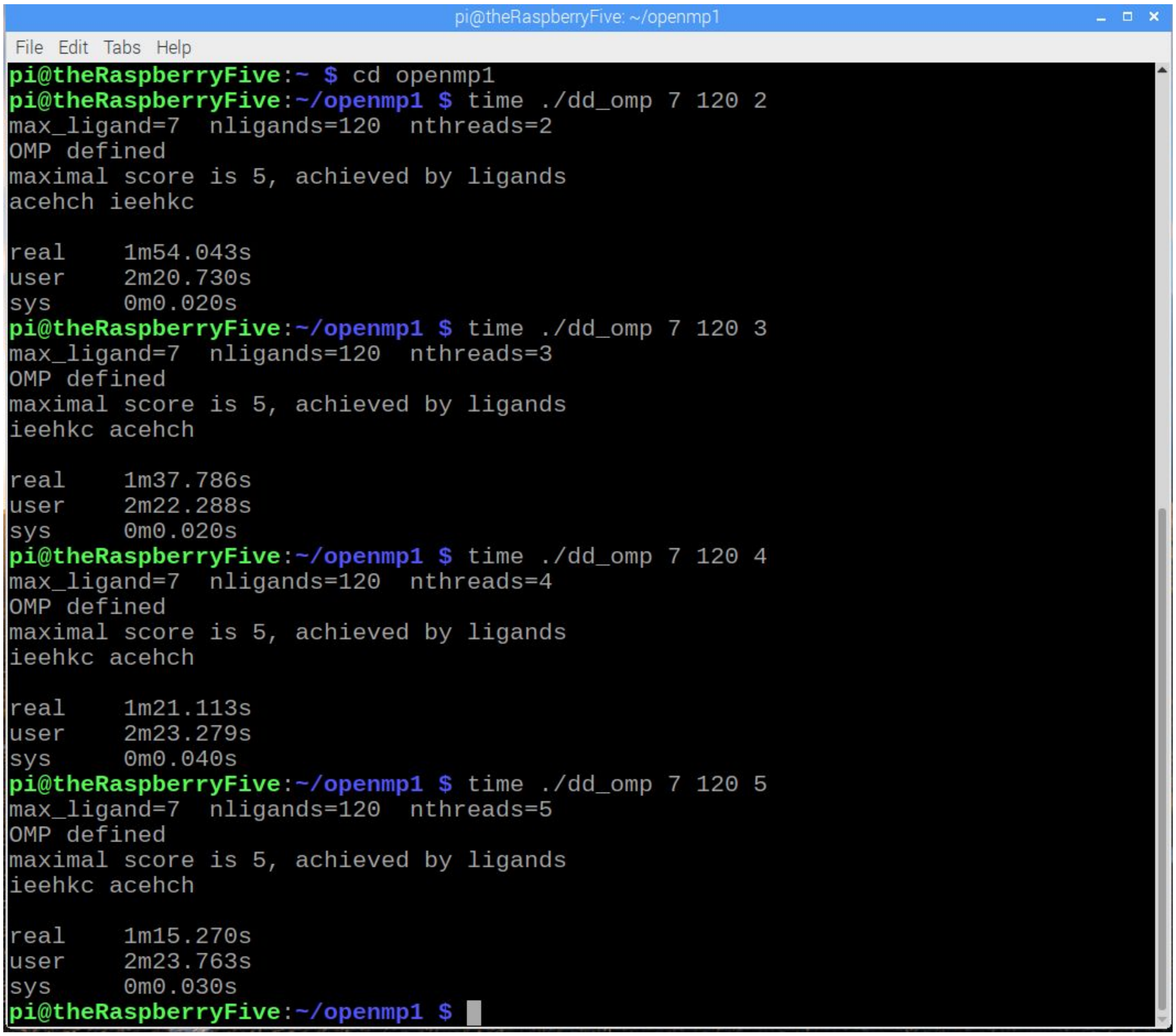
Running the program shown in the example (./dd_method 1) sets max ligands to 1 with default value 120 to nLigands and nThreads

The Results:

Implementation	Time (s)
dd_serial	129.972
dd_openmp	128.876
dd_thread	136.050

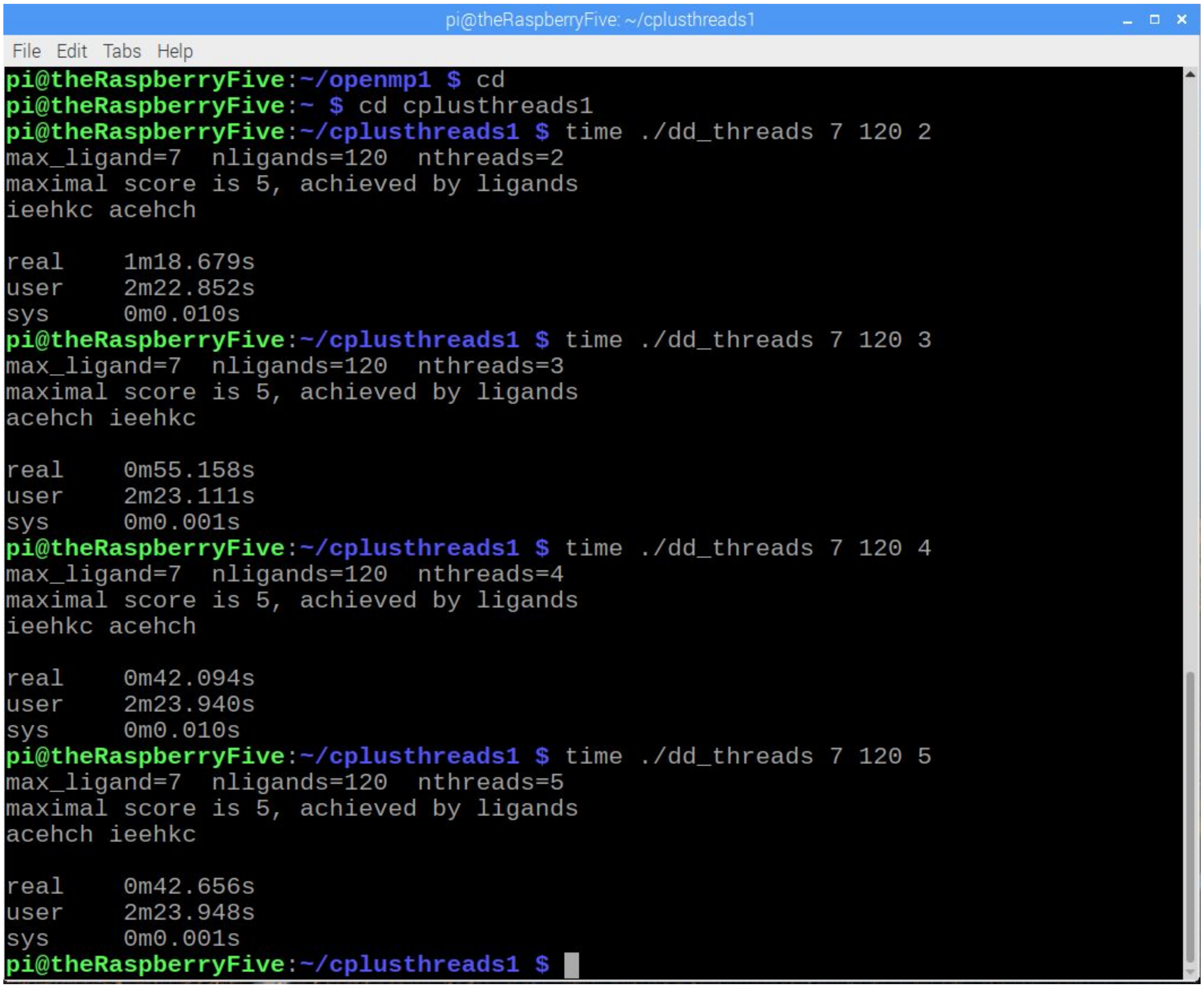
The run times look pretty close to dd_serial. While using one thread the run time is fast. Including libraries and more complicated program structure is most likely the reason behind these differences in times.

Increasing number of threads in openmp1 (Threads from 2 to 5):



```
pi@theRaspberryFive: ~/openmp1
File Edit Tabs Help
pi@theRaspberryFive:~ $ cd openmp1
pi@theRaspberryFive:~/openmp1 $ time ./dd_omp 7 120 2
max_ligand=7  nligands=120  nthreads=2
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc

real    1m54.043s
user    2m20.730s
sys     0m0.020s
pi@theRaspberryFive:~/openmp1 $ time ./dd_omp 7 120 3
max_ligand=7  nligands=120  nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch

real    1m37.786s
user    2m22.288s
sys     0m0.020s
pi@theRaspberryFive:~/openmp1 $ time ./dd_omp 7 120 4
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch

real    1m21.113s
user    2m23.279s
sys     0m0.040s
pi@theRaspberryFive:~/openmp1 $ time ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch

real    1m15.270s
user    2m23.763s
sys     0m0.030s
pi@theRaspberryFive:~/openmp1 $
```

Increasing number of threads in cplusthreads (Threads from 2 to 5):



```
pi@theRaspberryFive: ~/cplusthreads1
File Edit Tabs Help
pi@theRaspberryFive:~/openmp1 $ cd
pi@theRaspberryFive:~ $ cd cplusthreads1
pi@theRaspberryFive:~/cplusthreads1 $ time ./dd_threads 7 120 2
max_ligand=7  nligands=120  nthreads=2
maximal score is 5, achieved by ligands
ieehkc acehch

real	1m18.679s
user	2m22.852s
sys	0m0.010s
pi@theRaspberryFive:~/cplusthreads1 $ time ./dd_threads 7 120 3
max_ligand=7  nligands=120  nthreads=3
maximal score is 5, achieved by ligands
acehch ieehkc

real	0m55.158s
user	2m23.111s
sys	0m0.001s
pi@theRaspberryFive:~/cplusthreads1 $ time ./dd_threads 7 120 4
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch

real	0m42.094s
user	2m23.940s
sys	0m0.010s
pi@theRaspberryFive:~/cplusthreads1 $ time ./dd_threads 7 120 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc

real	0m42.656s
user	2m23.948s
sys	0m0.001s
pi@theRaspberryFive:~/cplusthreads1 $
```

Results:

Implementation	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads
dd_omp	114.043	97.786	81.113
dd_thread	78.679	55.158	42.094

The times start to drop with the addition of more threads, especially for the dd_threads implementation. The time dd_thread takes with 2 threads is faster than dd_opm can do with 4. However, we start to see diminishing returns with the increase of threads in dd_threads.

Questions:

1) Which approach is fastest?
   - Depends on the number of threads you plan on using. With one thread you are better off using dd_serial as it will save just a little bit of time since it doesn't contain as much code and libraries. However, if you want the flexibility of using more threads dd_threads using tbb is the fastest and is superior to dd_omp.

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

```
pi@theRaspberryFive: ~/openmp1
File Edit Tabs Help

pi@theRaspberryFive:~/cplusthreads1 $ wc -l dd_threads.cpp
208 dd_threads.cpp
pi@theRaspberryFive:~/cplusthreads1 $ cd
pi@theRaspberryFive:~ $ cd openmp
bash: cd: openmp: No such file or directory
pi@theRaspberryFive:~ $ cd openmp1
pi@theRaspberryFive:~/openmp1 $ wc -l dd_omp.cpp
194 dd_omp.cpp
pi@theRaspberryFive:~/openmp1 $
```

   - As shown in this image, the word count of dd_omp.cpp is 194 and dd_threads.cpp is 208. There is 13 word difference between the two, which is pretty good how much quicker dd_threads is compared to dd_omp.

3) Increase the number of threads to 5 threads. What is the runtime for each?

```
pi@theRaspberryFive:~/openmp1 $ time ./dd_omp 7 120 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch

real    1m15.270s
user    2m23.763s
sys     0m0.030s
```

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

real    0m42.656s
user    2m23.948s
sys     0m0.001s
```

Implementation	Time(s) 5 Threads
dd_omp	75.270
dd_threads	42.656

- It seems almost no change in time while running threads 5. This is because the raspberry pi is quad core, meaning it can handle 1 thread per core, which makes it run faster.

4) Increase the maximum ligand length to 7, and rerun each program. What is the run time for each?
   - The DEFAULT_max_ligand was already set at 7 and is what I used for my test to find the run time. Compared to times for max_ligand < 7 the runtime is much greater.

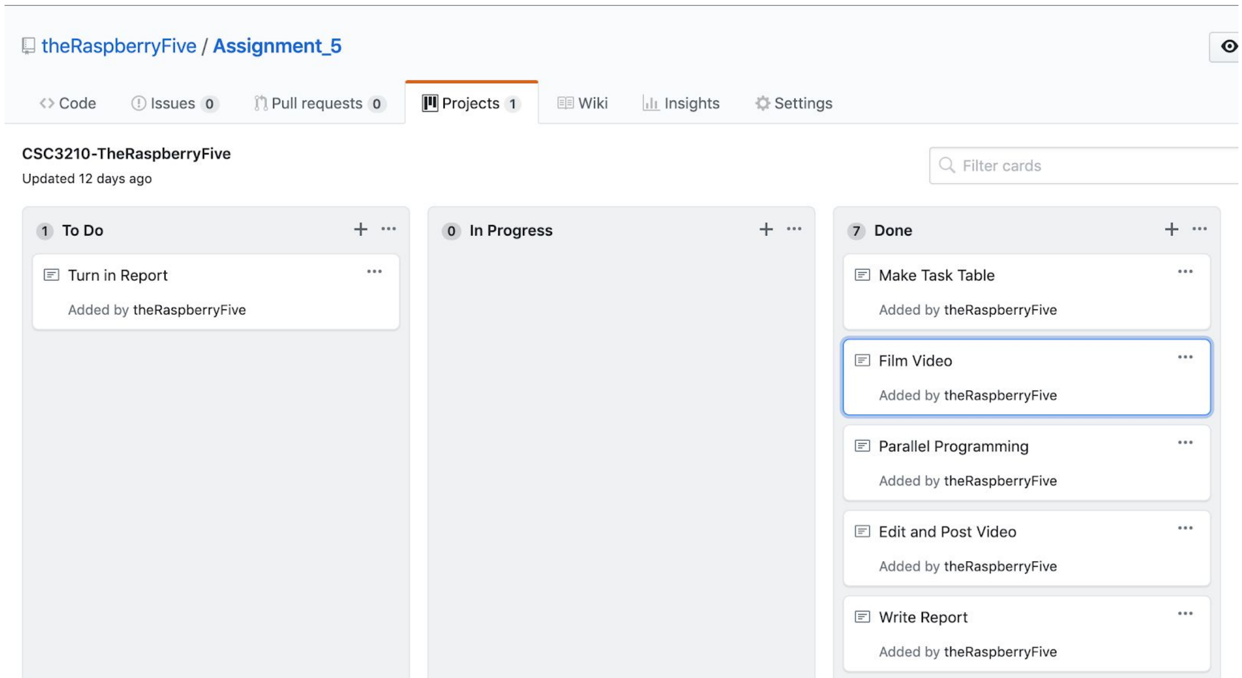
Runtime for max_ligand 7

Implementation	Time(s)
dd_serial	129.972
dd_openmp	128.876
dd_thread	136.050

Appendix

Slack Link: https://raspberryfive.slack.com/messages/CFQUGDXMX/details/

GitHub Link: https://github.com/theRaspberryFive



Youtube Video Link: https://youtu.be/r-IDnbSK88Y