

# **Developing Soft and Parallel Programming Skills Using Project-Based Learning**

**By Assembly Chefs  
Spring 2020**

**Team Coordinator:**

**Mezemir Gebre**

**Members:**

**Andy Lee**

**Bryanna Hardy**

**Mezemir Gebre**

**Nathan Heckman**

**Rahul Sunkara**

# Table of Contents

## Introduction:

<u>Project Cover Page</u>	pg. 1
<u>Table of Contents</u>	pg. 2
<u>Scheduling and Planning</u>	pg. 3

## Rahul Sunkara's Reports:

<u>Rahul's Title Page</u>	pg. 4
<u>Parallel programming foundations and</u>	pg. 5

## Andy Lee's Reports:

<u>Andy's Title Page</u>	pg. 9
<u>Parallel programming foundations</u>	pg. 10

## Bryanna Hardy's Reports:

<u>Bryanna's Title Page</u>	pg. 17
<u>Parallel programming foundations</u>	pg. 18

## Nathan Heckman's Reports:

<u>Nathan's Title Page</u>	pg. 23
<u>Parallel programming foundations</u>	pg. 24

## Mezemir Gebre's Reports:

<u>Mezemir's Title Page</u>	pg. 30
<u>Parallel programming foundations</u>	pg. 31

## Appendix:

<u>Links to Slack, GitHub, YouTube</u>	pg. 42
<u>Screenshots</u>	pg. 43

## Scheduling and Planning

Member's name	Email	Task	Due Date
Rahul Sunkara	rsunkara3@student.gsu.edu	Parallel programming basics individual report and parallel programming and map reduce question answers.	4/23/20
Andy Lee	Alee162@student.gsu.edu	Parallel programming basics individual report and parallel programming and map reduce question answers.	4/23/20
Bryanna Hardy	bhardy11@student.gsu.edu	Parallel programming basics individual report and parallel programming and map reduce question answers. Record and edit presentation video	4/23/20
Nathan Heckman	nheckman1@student.gsu.edu	Parallel programming basics individual report and parallel programming and map reduce questions. Creating folders for GitHub.	4/23/20
Mezemir Gebre	mgebrel@student.gsu.edu	Team coordinator. Parallel programming basics individual report and parallel programming and map reduce questions answer.	4/23/20

Member's name	Duration	Dependency	Note
Rahul Sunkara	Meeting: 1:30 hours Task: 2:00 hours	Microsoft Words, Raspberry Pi, GitHub	Done All Task 100% Grade
Andy Lee	Meeting: 1:30 hours Task: 1:30 hours	GitHub, Microsoft Word, Slack, Raspberry Pi	Done All Task 100% Grade
Bryanna Hardy	Meeting: 1:30 hours Task: 2:15 hours	Movie Editor, GitHub, Raspberry Pi, Microsoft Word	Done All Task 100% Grade
Nathan Heckman	Meeting: 1:30 hours Task: 2:30 hours	Raspberry Pi, Microsoft Word, GitHub	Done All Task 100% Grade
Mezemir Gebre	Meeting: 1:30 hours Task: 2:30 hours	GitHub, Microsoft Word, Raspberry Pi	Done All Task 100% Grade

### Summary:

The project assignment 5 is required everyone to communicate with each of the team members and collectively/ successfully achieve the final product. The crucial assignments for Project A5 were to: create a schedule table, completing the individual parallel programming report and answering the questions for the parallel programming and map reduce question, invite the TA into Slack and create required folders in GitHub, lastly, to record and upload a presentation video.

As mentioned in the schedule/ planner above, Mezemir Gebre, the team coordinator, had the responsibility of creating the planning table and making the final report. All the team members including me took up the responsibility to answer the parallel programming foundations questions and do their own individual parallel programming reports. Bryana Hardy had the responsibility of making a presentation video.

# **Rahul Sunkara Individual Questions and Reports**

# Parallel Programming Foundations

## Task 3

1. 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 are decomposition; assignment; orchestrating; and mapping. This drug design is a great example of parallel programming since the computer does decomposition through generating ligands, the assigning and orchestrating through the value of the ligand and lastly, mapping through finding the highest value ligand.

2. What is MapReduce?

MapReduce is a type of programming model, which has associated implementations for processing and generating big data sets. It does this through an algorithm that is parallel and distributed on a cluster.

3. What is a map and what is reduce?

Map is a function that takes in a key or a value pair, so that it can generate intermediate pairs. Reduce is a function that takes all the intermediate values that have the same immediate key and merge them.

4. Why MapReduce?

We choose MapReduce because of its ability to write the big data using the parallel programming techniques to process and to cluster the huge amounts of data, in a relatively small amount of time.

5. Show an example for MapReduce.

The Drug design program is once again a great example of MapReduce. Because for the computer to run it, it needs to generate protein ligands, which is a huge data and through parallel programming we can achieve it in less time.

6. Explain in your own words how MapReduce model is executed?

Every time MapReduce is used it breaks the tasks it has to do. During the executing the processor calls on the MapReduce library, which uses the parallel distribution to break the tasks and then they are assigned to clusters. After this the processor receives its task, then the master will assign the tasks to the worker which then either assigns it to map or to reduce. If the task is assigned to map, the worker program performs the task using the shared data set. After all the tasks have been completed, the worker program writes all the answers to the intermediate files, then the generated output is reduced by yet another worker program. After reducing the result is given to the master.

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

Word counter is a great example. For word finder we will be finding the unique words and the number of occurrences of those unique words. First, we divide the input into splits. This will distribute the work among all the map nodes.

Then, we tokenize the words in each of the mappers and give a hardcoded value to each of the words. Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. The mapping process remains the same on all the nodes. After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are

sent to the corresponding reducer. So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. Now, each Reducer counts the values which are present in that list of values. Finally, all the output key/value pairs are then collected and written in the output file.

8. When do we use OpenMP, MPI, and MapReduce (Hadoop), and why?

OpenMP is highly effective and used directive-based libraries.

MapReduce (Hadoop) is used when there is a large data, it breaks it up and does it task by task. Because of this breaking up process this is highly effective over large datas.

MPI is used as a distributed memory parallel model implementation it is mainly used for scientific applications.

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

DNA is the genetic code that contains the instructions for making proteins in all living things. The function of the protein is determined by its shape. To design a drug the programs needs to change the protein shape using the ligands it finds. In this program the computer is generating ligands to do the tasks of a particular protein. The programs computes a score/value for each ligand so that it can predict how well it will fit in the protein and change the shape accordingly. Then, the program identifies the highest value ligand and does the testing. In the drug design exemplar code, the line: `./drugdesign-static threads maxlen count,` shows the number of threads for the simultaneous and the maxlen is the maximum length of a ligand. Every ligand has a specific length and this is the score.

Run-time for 1:

Implementation	Time 1 thread
dd_serial	0m0.01s
dd_omp	0m0.02s
dd_threads	0m0.02s

```

pi@raspberrypi:~/sequential $ time -p ./dd_serial 1
maximal score is 1, achieved by ligands
p c r c n p p h o c c n n w i r r c p i o t r c e t a w h r n i r r o w y r h y
y c w a c y e o p e
real 0.01
user 0.01
sys 0.00

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 e w n n c o p p r a c o p r p i c h c c e o c y r r w y r i y
o a c h e y p n w r
real 0.02
user 0.00
sys 0.02

pi@raspberrypi:~/openmpi $ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
c o y e i c n n n y c h w i c c e c y a o r c t w r r i y p e c a r o h n h r p
real 0.02
user 0.01
sys 0.02

```

Run-time for 2, 3, 4 on Omp and Threads

Implementation	Time 2 threads	Time 3 threads	Time 4 threads
dd_omp	0.02s	0.04s	0.34s
dd_thread	0.02s	0.05s	0.23s

```

pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
n o o r a p t n r r t a a c h h o p a r h c t o
real 0.02
user 0.02
sys 0.01

pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 3
max_ligand=3 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
w h t h o p c h w
real 0.05
user 0.11
sys 0.00

pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4
max_ligand=4 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
h k i c o r e l i h j o r d r y h c h c i o w h t o r d t
real 0.23
user 0.82
sys 0.00

```

```

pi@raspberrypi:~/openmpi $ time -p ./dd_omp 2
max_ligand=2 nligands=120 nthreads=4
OMP defined
maximal score is 2, achieved by ligands
o p n o h h t n a c t o o r h c t a a p r r a r
real 0.02
user 0.02
sys 0.02

pi@raspberrypi:~/openmpi $ time -p ./dd_omp 3
max_ligand=3 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
h o p c h w w h t
real 0.04
user 0.10
sys 0.03

pi@raspberrypi:~/openmpi $ time -p ./dd_omp 4
max_ligand=4 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
o r e l o r d t h c h h k i c i h j o c i o r d r y w h t
real 0.34
user 0.83
sys 0.01

```

1. Which approach was the fastest?

For 1 thread the serial approach was the fastest. For 2 threads the runtime for omp and thread was the same; for 3 threads the omp approach was faster. For 4 threads the threads approach was 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?

Serial has 171 lines; Omp has 194 lines; and Threads has 208 lines. C++11 libraries are used for general purpose and only give the user certain amount of control. OpenMp on the other hand is way more efficient than the C++ threads..

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

```
pi@raspberrypi:~/openmpi $ time -p ./dd_omp 5
max_ligand=5 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
hoach
real 0.95
user 2.60
sys 0.02
pi@raspberrypi:~/openmpi $ cd ~
```

```
pi@raspberrypi:~/openmpi $ time -p ./dd_omp 7
max_ligand=7 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 82.74
user 149.50
sys 0.02
```

```
pi@raspberrypi:~/sequential $ time -p ./dd_serial 7
maximal score is 5, achieved by ligands
acehch ieehkc
real 138.92
user 138.90
sys 0.01
pi@raspberrypi:~/sequential $ cd ~
```



# **Andy Lee Individual Questions and Reports**

# Parallel Programming Foundations

## Task 3

### **“Introduction to Parallel Programming and MapReduce”**

**(15p) What are the basic steps (show all steps) in building a parallel program? Show at least one example.**

There is 4 steps to do when building the parallel program. These are 1) Decomposition 2) Assignment 3) Orchestration and 4) Mapping. One example that can be used to explain the “building a parallel program” is a “Drug design exemplar code”. In “Drug design exemplar code”, the computer must perform a 3 steps. First, Generate the ligands to try for particular protein, Second Compute the score for each ligand, and third, identify the highest scoring ligands.

Comparing those 3 steps with the 4 steps to build the parallel program, we could identify that these steps are similar. For example, Generate ligands can be interprets as Decomposition, Compute a score for each ligand is same as assignment and Orchestration, last, identifying highest scoring ligands(find the pattern) is same as mapping.

To simplify the statement using the Drug design exemplar example,  
Decomposition == Generate ligands to try for particular protein  
Assignment and Orchestration == Compute a score for each ligand  
Mapping == Identify the highest scoring ligands.

Overall, this is the way to build the parallel program.,

### **(5p) What is MapReduce?**

MapReduce is a programming model that is used to perform a parallel programming when generating a big data set to the cluster. When user generate the specific map function, the processor will generate user specific key, and those key will generate the intermediate key pairs. This chain reaction will eventually reduce the function because those specific key will merge all the intermediate key values associated with the same intermediate key.

### **(10p) What is map and what is reduce?**

Map and reduce are a constructs that user could apply in your data set, when performing a parallel programming. This idea is typically used a HDFS system, meaning you would use map and reduce when you want to apply the operation to each individual element in data set. Map and reduce typical used in very large data set where you map the similar data into one group, then reduce the number of data so that user’s computer would perform the task faster (due to parallel tasking).

### **(5p) Why MapReduce?**

Since the MapReduce can write the scalable data using the parallel programming to process/cluster the large amount of data, we could generate the big data sets in small amount of time. These usage on large data set usually used in the scientific application, thus, this is the main reason why the scientist application used a MapReduce (like DNA example)

### **(5p) Show an example for MapReduce**

In example from the PDF, the drug Design and DNA is perfect example. In order to perform this, the computer must generate the ligands to define the particular protein. The number of generating ligands is huge and computing these data would takes forever if we are not using the parallel processing. When we compute the similarities of the DNA, processor will score each individual ligand, thus find the pattern/order of each element. Once processor has scored all of

the individual ligands, now the processor would sort the values, then identify the highest scoring ligand. These all will be performed using the MapReduce.

**(10p) Explain in your own words how MapReduce model is executed?**

Every MapReduce starts with splitting the task. The processor will perform this task by calling the MapReduce library. The MapReduce library would split the task equally using the parallel distributing algorithm and assigned to cluster. Once the processor received the task, the master will assign the task to the worker either assign map or assign reduce. Once worker assigned a map, the worker would perform the task using a dataset given by a shred, then retrieved the resulted value. Once all worker finished their task, the worker would now write all the answer to the intermediate files (aka local disk), so that the file is accessible to every worker. Now, the generated resulted output would go to the workers again to perform a reduce. Once the reduce is done, all the necessary result will be given to the master, then the master write the output to the designated file.

**(6p) List and describe three examples that are expressed as MapReduce computations.**

According to the towardsdatascience website, the one example I could find is “Term-Vector per Host”.

A “Term-Vector per Host” is a term vector that summarized the most important word that occurs in a certain document by using two parameters: word, frequency. Once the map function received the input of hostname, term vector, it will identify the most shown word/most significant word on the document. Using a MapReduce is a significant since computer would compute individual words, then define the result.

The second is “Inverted Index”. For each individual word, it will create the list of the document using two parameters: word, document ID. Once parameter is set, the reduce function accepts all pairs of word, then sort it out into certain pattern. Then, it will emit the word, document ID in to sorted pair. Overall, the set of all output will get inverted, then you finish the task.

Last but not least, Distributed Grep is example of Map function that emits the line if the given pattern is matched with a result. The reduce is used when identifying the function that are copied and supplied from intermediate data point.

**(6p) When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?**

The OpenMP is widely spread that is almost set to be a convention when programmer needs a efficient directive based library. The OpenMP is free to use which gave a huge advantage.

The MPI is used to develop the parallel scientific applications. It usually tightly synchronous code and well loaded balance. MPI pass the message and it distribute the memory when performing the parallel model implementation

Last but not Least, The Hadoop MapReduce, like other regular MapReduce, can be used in to identify the pattern/result from the large amount of data (large dataset). This used a HDF system where it distribute the file efficiently, which gave a huge advantage when doing a parallel program

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

Our DNA is structured with consist of two RNA, which have a important data to our body system. DNA can be considered as the book of recipes. Here, we have to understand the hundred and millions of data to store in a DNA and form the protein's shape. This requires a lot of data; which MapReduce is a perfect fits for performing a testing. There is a 3 steps to build the drug design software. First computer must generate the ligands to try for the particular protein. Each

ligands will be randomly generated with random character, and length. Some Ligands will carry a bunch of data and some are not, thus only some will fits. Now we must compute the score for each ligand. Computer will compare the maximum match with each other, then scoring solidly base on insertion and deletion. The highest score would represent how the DNA fits in a assigned protein shape. Once the scoring the ligands is done, now the computer would identify the highest scored ligands, and wrote it on the designated file

## Drug Design and DNA in Parallel

### Measure Run time

Implementation	time
dd_serial	137.71s
Dd_omp	0.02s
Dd_threads	0.02s

dd\_serial time :

```
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 137.71
user 137.59
sys 0.03
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ scrot
```

Dd\_omp time:

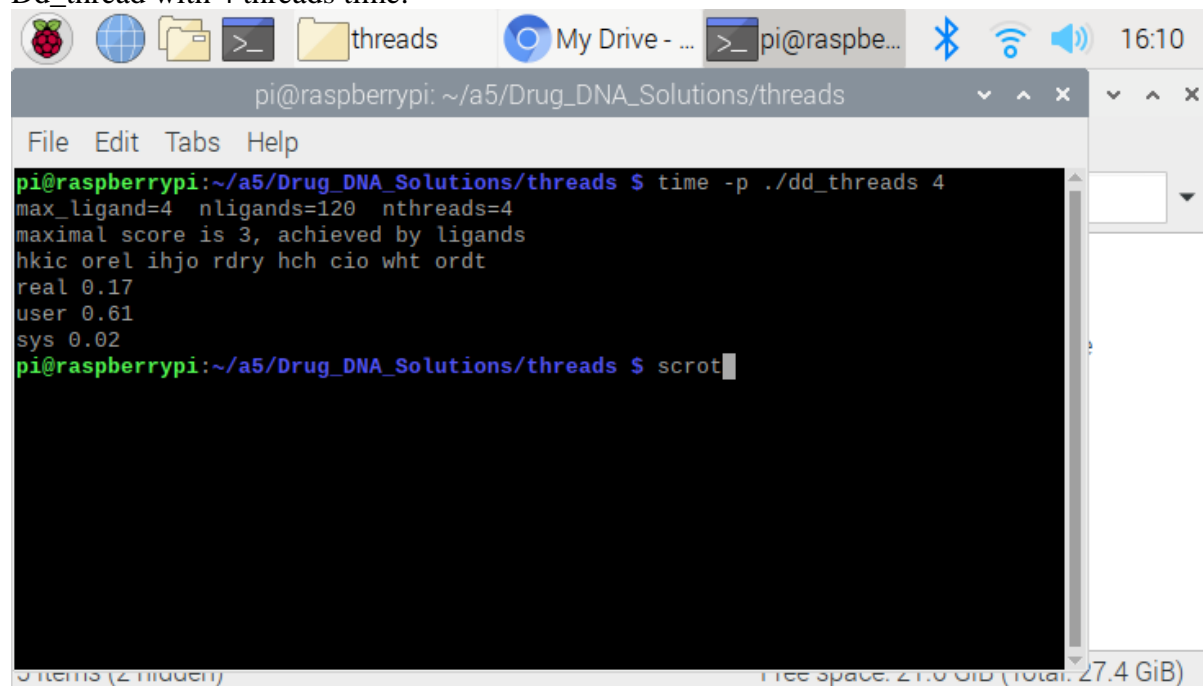
```
pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
r o r w w e c c r n n h n c c i o p y h c i a c y e w a t e c y c o o r r p p r
p p h i y r n w t r
real 0.02
user 0.00
sys 0.02
pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ scrot
```

Dd\_threads time:

```
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 1
max_ligand=1 nligands=120 nthreads=4
maximal score is 1, achieved by ligands
p n r c p p c o c h n c n w i r r c i p o t r t c e a w r h n h r r w c r o y y
i a c y w e y o e p
real 0.02
user 0.01
sys 0.01
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ scrot
```

Implementation	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads
Dd omp	0.02s	0.04s	0.21s
Dd threads	0.02s	0.04s	0.17s

Dd\_thread with 4 threads time:

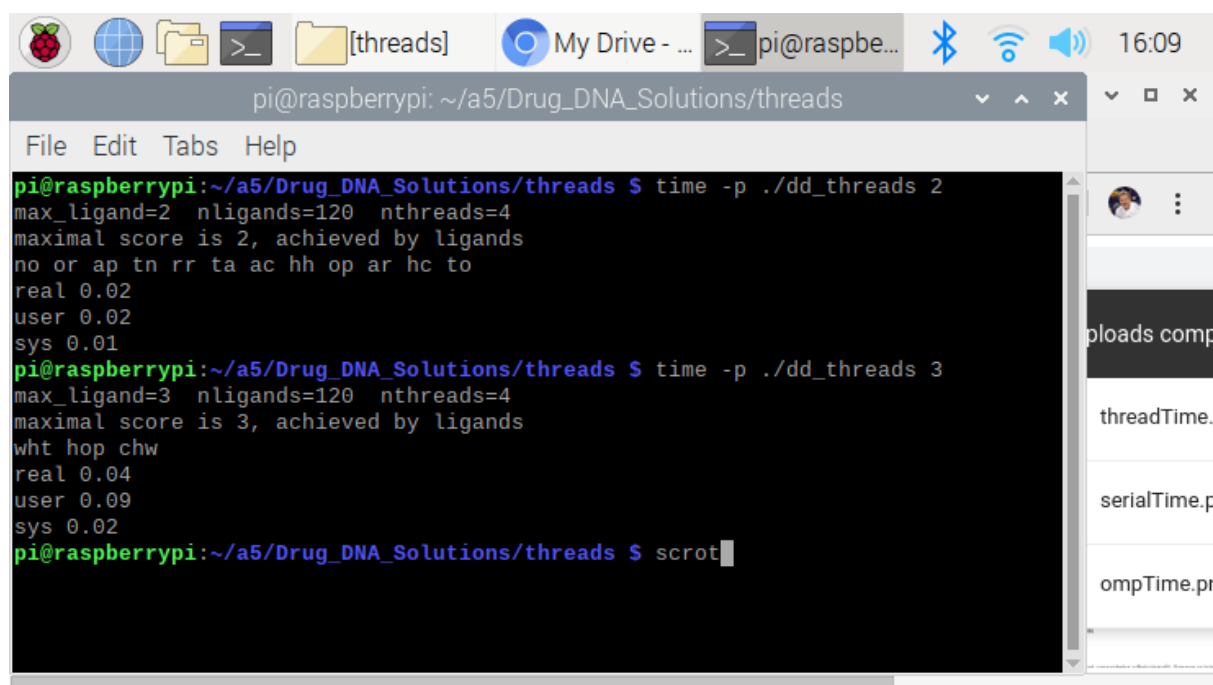


```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/threads
File Edit Tabs Help
pi@raspberrypi:~/a5/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 cio wht ordt
real 0.17
user 0.61
sys 0.02
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ scrot

```

Dd\_thread with thread 2 and 3 time:



```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/threads
File Edit Tabs Help
pi@raspberrypi:~/a5/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 ap tn rr ta ac hh op ar hc to
real 0.02
user 0.02
sys 0.01
pi@raspberrypi:~/a5/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 0.04
user 0.09
sys 0.02
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ scrot

```

Dd\_omp with 2 and 3 threads time:

```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/openMP
File Edit Tabs Help

pi@raspberrypi:~/a5/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
rr ar tn hc no to ac hh or op ta ap
real 0.02
user 0.03
sys 0.00

pi@raspberrypi:~/a5/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
wht chw hop
real 0.04
user 0.12
sys 0.00

pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ scrot

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/openMP
File Edit Tabs Help

pi@raspberrypi:~/a5/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
ordt hch orel hkic ihjo rdry cio wht
real 0.21
user 0.54
sys 0.04

pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ scrot

```

### Discussion Questions

1. Which approach is the fastest?

With only 1 threads, the dd\_omp and dd\_threads are tied each other. However, when it goes up to the 4 thread, dd\_threads method is clearly fastest.

Regardless of number of threads, the serial method is definitely slowest method.

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

Dd\_omp's OpenMP has 152 words      Dd\_omp's C++ has 194 words  
 Dd\_serial's OpenMP has 63 words      Dd\_serial's C++ has 170 words  
 Dd\_threads's OpenMP has 137 words      Dd\_threads's C++ has 207 words

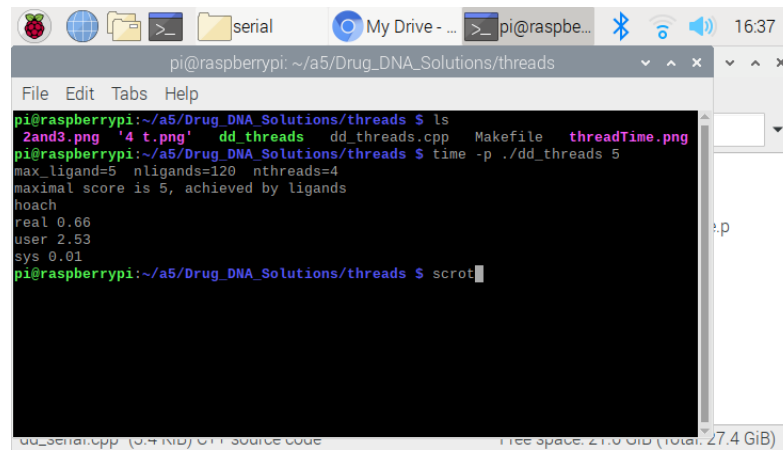
Looking these trends, the C++ requires the more lines of code compare to the OpenMP. This is because the C++ library is general purpose library, whereas the OpenMP is designed for the parallel program. Therefore, in parallel programming, the OpenMP is more efficient.

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

Dd\_threads runtime with 5: 0.66s

Dd\_omp runtime with 5: 1.08s

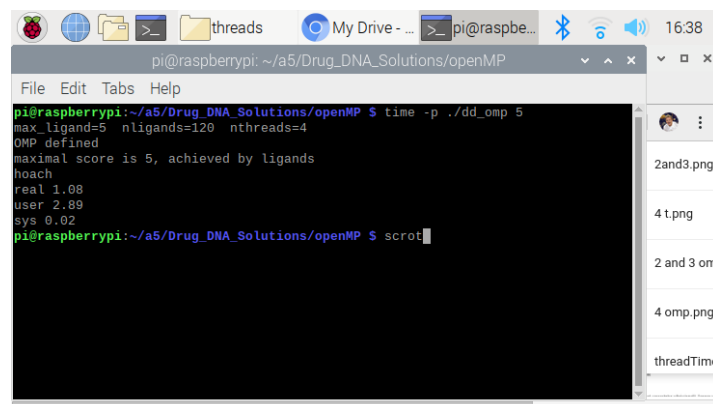
Dd\_serial runtime with 5: 2.76s



```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/threads
File Edit Tabs Help
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ ls
2and3.png '4 t.png' dd_threads dd_threads.cpp Makefile threadTime.png
pi@raspberrypi:~/a5/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 0.66
user 2.53
sys 0.01
pi@raspberrypi:~/a5/Drug_DNA_Solutions/threads $ scrot

```



```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/openMP
File Edit Tabs Help
pi@raspberrypi:~/a5/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 1.08
user 2.89
sys 0.02
pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ scrot

```

```

pi@raspberrypi: ~/a5/Drug_DNA_Solutions/serial
File Edit Tabs Help
pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ scrot
pi@raspberrypi:~/a5/Drug_DNA_Solutions/openMP $ cd ../
pi@raspberrypi:~/a5/Drug_DNA_Solutions $ cd ./serial/
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 5
maximal score is 5, achieved by ligands
hoach
real 2.59
user 2.58
sys 0.00
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ ^C
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 5
maximal score is 5, achieved by ligands
hoach
real 2.76
user 2.76
sys 0.00
pi@raspberrypi:~/a5/Drug_DNA_Solutions/serial $ scrot

```

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

The code is already set to be a maximum length of 7. Thus, the answer for each method is given above (question 1 and 2)



# **Bryanna Hardy Individual Questions and Reports**

# Parallel Programming Foundations

## Task 3

### Questions:

1. What are the basic steps (show all steps) in building a parallel program? Show at least one example.
  - a. The basic steps in building a parallel program are identifying sets of tasks that can run concurrently and partitions of data that can be processed concurrently. An example is taking the area of an inscribed circle. You would use the implementation technique called master/worker. Then master/worker method implements static loading balance which is used when all tasks are performing the same amount of work. In addition, it tries to spread tasks among the processors in a parallel system.
2. What is MapReduce?
  - a. A MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.
3. What is a map and what is reduce?
  - a. A **map** processes a key/value pair to generate a set of intermediate key/value pairs, and a **reduce** merges all intermediate values associated with the same intermediate key.
4. Why MapReduce?
  - a. Because it can be used to write tightly coupled scientific applications. Also, it processes and generates big data sets.
5. Show an example for MapReduce.
6. Explain in your own words how MapReduce model is executed?
  - a. The MapReduce library first shards the input files into M pieces. Then starts up many copies of the program on a cluster of machines. One of the copies of the program is special which is the master. The rest are workers that are assigned work by the master. The master picks workers and assigns each one with a task or a reduce task. The worker who is assigned to a map task reads the contents of the corresponding input shard. It pairs values out of the input data and passes each pair to the user-defined Map function. The buffered pairs are written to the local disk. The locations of these buffered pairs on the local disk are passed back to the master, whose task is to forward these locations to the reduced workers. When a reduce worker is notified by the master, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When it has read all intermediate data, it sorts it by the intermediate keys so all the same key is grouped together. If the data is too large to fit in memory, an external sort is used. The reduce worker iterates over the sorted intermediate data for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. When the map task and reduce tasks have been completed, the master

wakes up the user programs. The MapReduce call in the user program returns back to the user code.

7. List and describe three examples that are expressed as MapReduce computations.
  - a. Distributed Grep: the map function emits a line if it matches a given pattern. The reduce function is an identity function that copies the supplied intermediate data to the output.
  - b. Reverse Web-Link Graph: output <target, source> pairs for each link to a target URL found in a page named “source”. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>.
  - c. Count of URL Access Frequency: map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.
8. When do we use OpenMP, MPI, and MapReduce (Hadoop), and why?
  - a. OpenMP: It is an efficient directive-based library that you could use.
  - b. Hadoop MapReduce: Can use when applying over large data; it distributes. It is a distributed file system designed to run on commodity hardware.
  - c. MPI: Message Passing Interface is a distributed memory parallel model implementation, mostly used to develop parallel scientific applications.
9. In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.
  - a. DNA is illustrated like a book of recipes. DNA contains the instructions for making proteins in our bodies. The shape of a protein determines the function it performs in an individual's body. For you to design a drug, you need to find ligands, new pieces, to change a protein's shape. A way to design a drug design software, you first need to generate ligands to try for a particular protein. There are some ligands that will fit and some will not. The second thing is that you need to compute a score for each ligand that simulates how well it will fit in the protein and produces the desired shape change. Thirdly, identify the highest scoring ligands for actual synthesis (production) and testing. In the drug design exemplar code, the command-line arguments: “./drugdesign-static *threads maxlen count*. The threads are the number of simultaneous threads, the maxlen is the maximum length of a ligand. Each ligand has a random length up to this max. Then, count the number of ligands to score.

**MEASURE RUN-TIME:**

IMPLEMENTATION	TIME(S)
dd_serial	176.63
dd_omp	0.02
dd_threads	0.03

```
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 176.63
user 176.60
sys 0.00
```

```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
c o y i i c t n n e y c h w r w c c a e r w r o h i y n c y p w r r c r o n h p
c p p r t r o p a e
real 0.02
user 0.01
sys 0.01
```

```
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 p i c h c c e o c y i h e r r c o
y y n w a p w r r y
real 0.03
user 0.01
sys 0.01
```

IMPLEMENTATION	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads
dd_omp	0.02	0.05	0.26
dd_threads	0.02	0.05	0.17

```
max_ligand=2 nligands=120 nthreads=4
OMP defined
maximal score is 2, achieved by ligands
rr hh ta ac ar to hc tn no or op ap
real 0.02
user 0.02
sys 0.02
```

```
pi@raspberrypi:~/openmp1 $ 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 0.01
```

```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=4 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
ihjo hch rdry hkic wht cio ordt orel
real 0.26
user 0.61
sys 0.00
```

```
pi@raspberrypi:~/openmp1 $
```

```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
no or ap tn rr ta ac hh op ar hc to
real 0.02
user 0.00
sys 0.03
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 3
max_ligand=3 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
hop chw wht
real 0.05
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 0.17
user 0.55
sys 0.01
```

Which approach is the fastest?

By comparing the results, in the images, you can tell that cplusthreads runtime is faster than openmp. So, cplusthreads1 approach is the fastest.

Determine the number of lines in each file. How does the C++11 implementation compare to OpenMp implementations?

Serial had 171 lines in its file, 194 lines in omp file, and 208 lines in threads file. C++ multithreading libraries are general purpose and gives less control over. OpenMp is much efficient than C++ threads. OpenMp is designed to not impact the code and designed for multiple set of problems.

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

```
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1 120 5
max_ligand=1 nligands=120 nthreads=5
maximal score is 1, achieved by ligands
c w h n r t r t i e w n n c o p p r a c o p r i p c h c c e o c y r o a y i c r
h r e y y r w n p w
real 0.02
user 0.01
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 64.99
user 222.41
sys 0.02
```

```
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1 120 5
max_ligand=1 nligands=120 nthreads=5
OMP defined
maximal score is 1, achieved by ligands
r h c t w r i i a e c n a p w y p r n r w n r o t o p p y c r c n p y e y h c h
i r r e c w o c o c
real 0.02
user 0.01
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 91.73
user 201.46
sys 0.02
```

As you can see, increasing the number of the threads, you can see the difference in the real time when you compare it with the other times in the images above. OpenMp and Threads works at the

same pace when you increase the number of threads by 5. On the other hand, when you increase the maximum number of ligands to 7, you can tell that OpenMp has a longer runtime than Threads.

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

When I increased the maximum length to 7, the runtime is illustrated in the images above.

# **Nathan Heckman Individual Questions and Reports**

# Parallel Programming Foundations

## Task 3

Nathan Heckman

A5 Parallel Programming Foundations

### **What are the basic steps in building a parallel program?**

1. Identify tasks that have the ability to run concurrently
2. Determine if the task has no dependencies and if the same processing is required for each computational element
3. Implement the program using the master/worker technique and static load balancing if applicable

The example given revolved around approximating the value for pi. Inscribe a circle within a square and then have each worker generate two random numbers that lie within the square. The workers then see if the coordinate lies within the circle, and adds one to countCircle if it does. The master then computes pi from the given countCircle value.

### **What is MapReduce?**

MapReduce is a programming model for generating big data sets with a parallel algorithm on a cluster. It was developed by Google as a way to process large amounts of raw data by distributing it across thousands of machines in order to complete the process in a reasonable amount of time.

### **What is map and what is reduce?**

In Lisp, map takes a function and sequence of values as input and then applies the function to all the values. In MapReduce, map takes input and produces intermediate key pairs and then passes all of them to the reduce function. Reduce combines the intermediate keys given by map and a set of values to produce a smaller set of values that is easier to understand.

### **Why MapReduce?**

MapReduce allows engineers to perform simple computations quickly while hiding the underlying details of the program.

### **Show an example for MapReduce.**

A good example of MapReduce is the example given in the MapReduce article about counting occurrences of words in a large document collection. The map function emits each word plus an associated count of said word and the reduce function adds all the counts for a particular word.



### **How is the MapReduce model executed?**

Map segments are automatically distributed across multiple machines. Data is split into sets of shards and these are executed in parallel on different machines. Reduce segments are distributed by breaking up the intermediate key into R pieces using a partitioning function which is user-specified.

### **List and describe three examples that are expressed as MapReduce computations:**

Distributed Grep – map function emits a line if it matches the given pattern. Reduce function copies the supplied intermediate data to the output.

Count of URL Access Frequency – map function processes web page request logs and outputs. Reduce function adds together all values for the logs and emits a <URL, total count> pair.

Reverse Web-Link Graph – map function outputs target-source pairs to a target URL. Reduce function concatenates all source URLs that have something to do with a target URL and emits a pairing

### **When do we use OpenMP, MPI and MapReduce (Hadoop)? Why?**

OpenMP – OpenMP is a great directive-based library to use shared memory parallelism in code. Using #pragma parallel we can split the loop into threads and each thread then handles a chunk of the loop iterations.

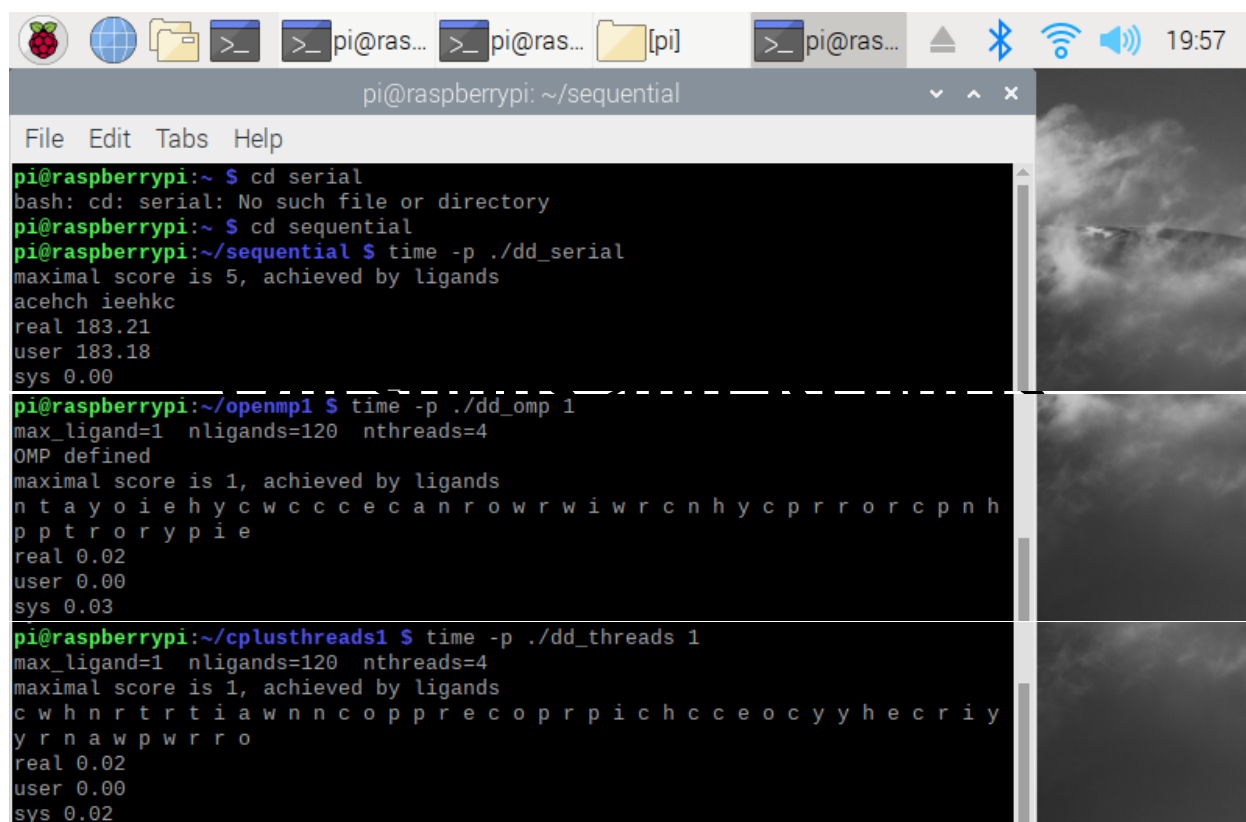
MPI – Message Passing Interface is used when developing parallel scientific applications. The code is tightly synchronous and well load balanced. Can be used to develop most parallel code that runs over multiple machines and gives access to hybrid programming.

MapReduce (Hadoop) – Hadoop gives two constructs that can be applied over large sets of data (map and reduce). Makes it possible to chain maps and reductions to design complex problems.

### **Explain what a Drug Design and DNA problem is:**

The strategy for drug design software involves generate ligands, computing a fitness score for each ligand, and identifying the highest ligand score and then testing it. This helps scientists develop medicine to help repair proteins and transform them into a desirable shape. DNA contains the instructions for producing proteins and their different shapes and drug design software can help us learn how to alter the protein structure in a body to produce a certain shape.

A5 Parallel Programming Task



```
pi@rasberrypi: ~/sequential
File Edit Tabs Help

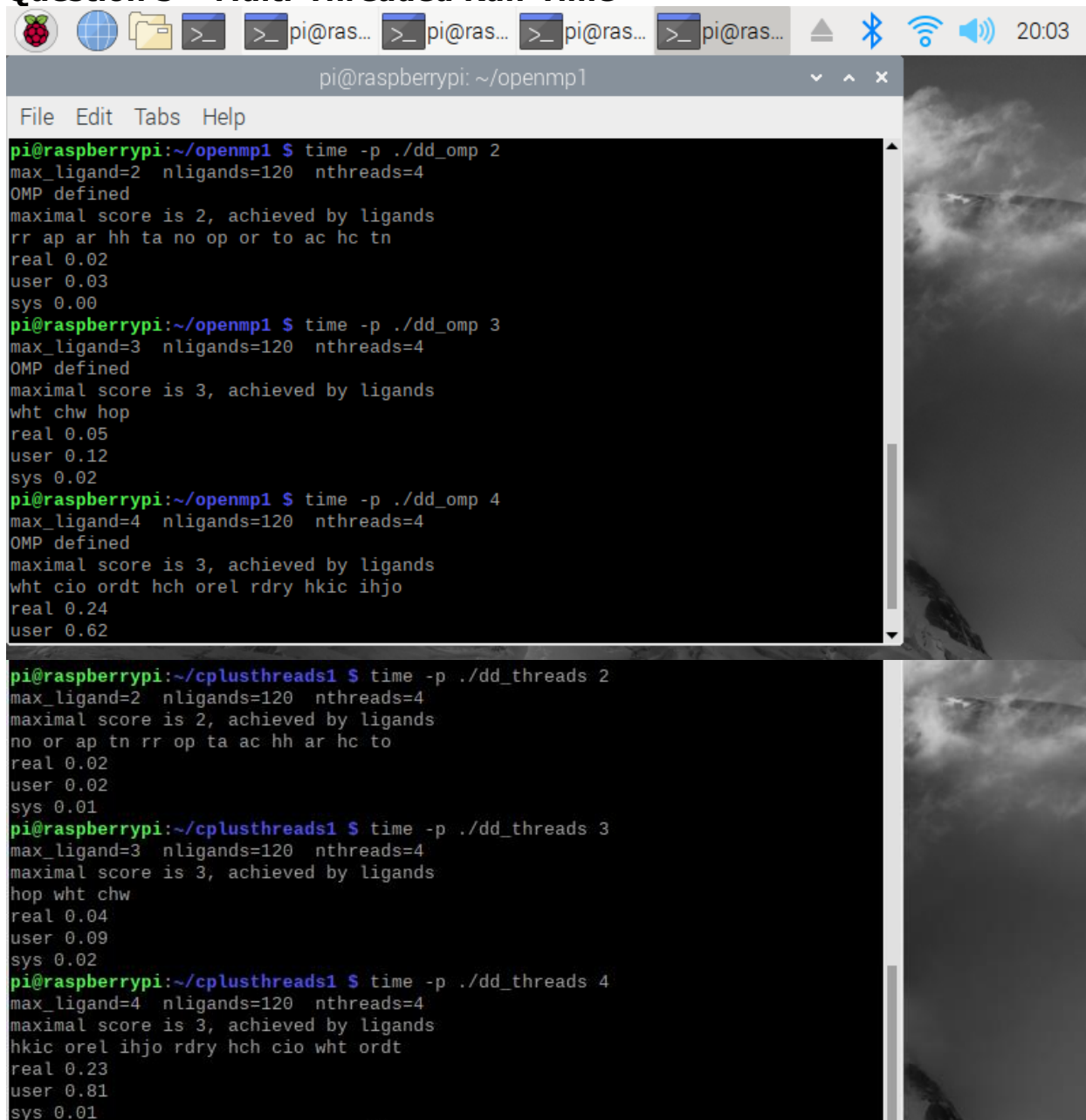
pi@rasberrypi:~ $ cd serial
bash: cd: serial: No such file or directory
pi@rasberrypi:~ $ cd sequential
pi@rasberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 183.21
user 183.18
sys 0.00

pi@rasberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
n t a y o i e h y c w c c c e c a n r o w r w i w r c n h y c p r r o r c p n h
p p t r o r y p i e
real 0.02
user 0.00
sys 0.03

pi@rasberrypi:~/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 p i c h c c e o c y y h e c r i y
y r n a w p w r r o
real 0.02
user 0.00
sys 0.02
```

Implementation	Time (s)
dd_serial	183.21
dd_omp	0.02
dd_threads	0.02

### Question 5 – Multi-Threaded Run-Time



```

pi@raspberrypi: ~/openmp1
File Edit Tabs Help

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 ap ar hh ta no op or to ac hc tn
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
wht chw hop
real 0.05
user 0.12
sys 0.02
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=4 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
wht cio ordt hch orel rdry hkic ihjo
real 0.24
user 0.62
sys 0.01

pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
no or ap tn rr op ta ac hh ar hc to
real 0.02
user 0.02
sys 0.01
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 3
max_ligand=3 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
hop wht chw
real 0.04
user 0.09
sys 0.02
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 0.23
user 0.81
sys 0.01

```

Implementation		Time (s) 3 Threads	Time (s) 4 Threads
dd_omp	0.02	0.05	0.24
dd_threads	0.02	0.04	0.23

I realized after finishing this that following the assignment instructions for this part only change the max\_ligand value and not the thread count, hence why the times seem off. I followed exactly what the instructions gave so I don't think we should lose points on this question.

## 2.3 Discussion Questions

### Which approach is fastest?

The dd\_threads approach is fastest across the board, including the example below that uses 5 threads and a max\_ligand size of 7.

### Determine the number of lines in each file.

OpenMP – 193

Serial – 170

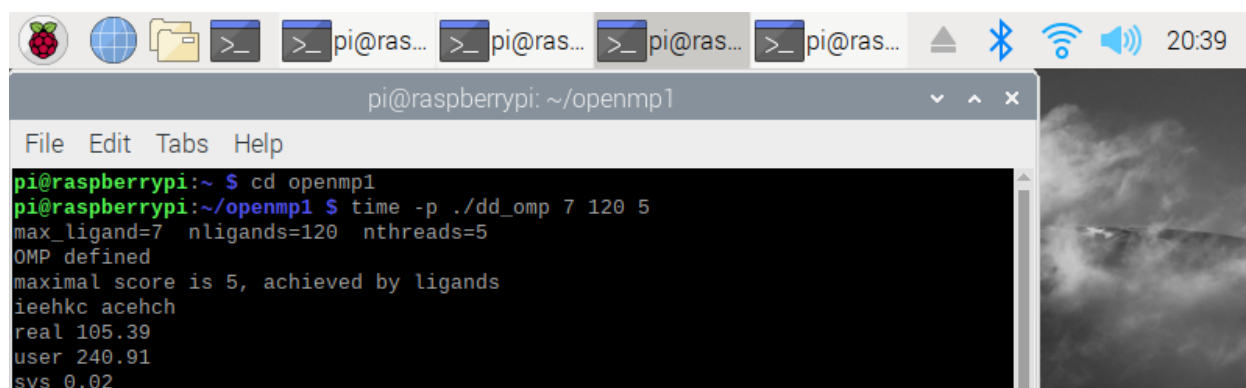
Threads – 207

The threads approach has the most lines but is also the most efficient.

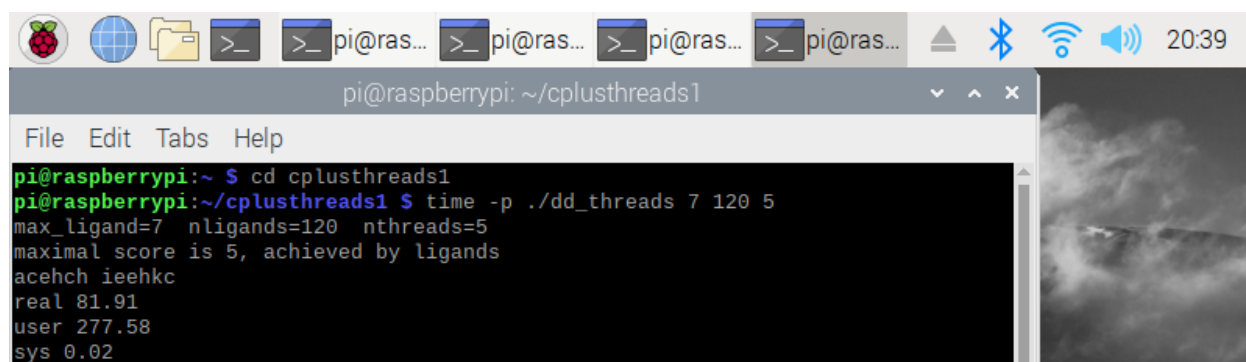
### Increase the number of threads to 5. What is the runtime for each?

OpenMP – 105.39s

Threads – 81.91s



```
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~ $ cd openmp1
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 105.39
user 240.91
sys 0.02
```



```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~ $ cd cplusthreads1
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 81.91
user 277.58
sys 0.02
```

**Increase the maximum ligand length to 7, and rerun. What is runtime for each?**

See above images. They incorporate both the max\_ligand size of 7 and 5 threads.

# **Mezemir Gebre Individual Questions and Reports**

# Parallel Programming Foundations

## Task 3

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

**The first step is identifying the tasks that could potential run simultaneously then the next step is about checking whether the tasks are independent or not and determine if the process required for each of them. The final step is about implementing the program using the master/worker system and static load balancing if it is possible. Inscribing a square in a circle could be an example.**

- What is Map Reduce?

**Map Reduce is a parallel computation mechanism for processing and distributing large amount of raw data across multiple machines.**

- What is map and what is reduce?

**These are two different functions in the MapReduce library developed by users of this library. Map takes pair as an input and gives a set of key and value pairs as an output. Reduce accepts a key and value from the output of the map function and generate smaller set of values by merging the values together for that key.**

- Why MapReduce?

**MapReduce significantly reduces our program's run time by allowing parallel computation as a result we will have a faster run time than sequential executing programs**

- Show an example for MapReduce.

**For example, let's say we have a program that counts the number of occurrences of each word in a document. In this program we can use the map function to map each word with its number of occurrence and then we can use the reduce function to ignore the key/ value pairs that have already been counted.**

- Explain in your own words how MapReduce model is executed?

**First the map reduce library distribute multiple copies of the tasks with the input files in a cluster of a machine. From those copies, there is master one that assigns map or reduce tasks to the workers, which are the rest of the copies of the program. Based on their assigned task these workers read the contents of their input shard then they parse the key/value pairs and send it to the map function which is stored in the local disk. Then these pairs from the local disk will be passed back to the master then the master notifies these locations to the reduce workers. Then the reduce workers read the buffered data from the local disks to the map workers and it sort it by the intermediate keys. The reduce workers iterate over the sorted data and pass these data to the reduce function, which adds the result to the final output file. Finally, the master runs the user programs and the MapReduce call will end and go back to the user code.**

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



**The distributed Grep:** produces a line if there is a match between the document and the given pattern. The reduce function copies the given intermediate data to the output.

**Term-vector per Host:** organize the most important word in a document with its frequency of occurrence as a pair.

**Inverted index:** The map function pairs the word with its id. The reduce function accept those pairs and sort them and produce pairs of word and list then they form an inverted index.

- When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?  
**OpenMP** is a free efficient directive-based library so many developers use it widely to support their programs.

**MPI** used for parallel model implementation interface by passing a message and distribute that to the memory.

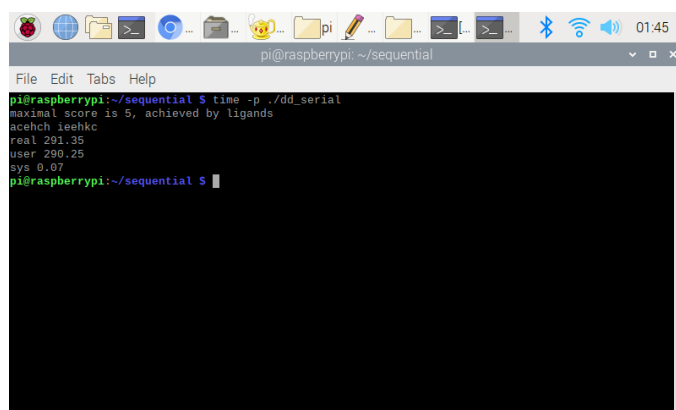
**Hadoop:** is a programming model that can be used for processing big data in parallel by decomposing the work into independent set of tasks.

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

**The goal of the Drug design and DNA problem is finding drugs or ligands that are good nominees based on previous identification of a protein that are related with the disease through an experiment or molecular modeling computation. In the experiment or in the computation, these selected ligands will be tested against the identified protein of the disease to know if they bind with the protein in meaningful way or not. A score will be set based on the binding properties and the best scores will be marked and become a good nominee to make the drug for that disease.**

#### **5) Measure Run-Time**

Implementation	Time 1 thread
dd_serial	291.35
dd_omp	0.02
dd_threads	0.02



```

pi@raspberrypi: ~/sequential
File Edit Tabs Help
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 291.35
user 290.25
sys 0.07
pi@raspberrypi:~/sequential $

```

```

pi@raspberrypi:~/openmp$ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
p w o w r e r c c r n h c c i o y w c y t c e a e y p c a i c h y n i n h p p r t r p p o r r r e
w
real 0.02
user 0.03
sys 0.00
pi@raspberrypi:~/openmp$

pi@raspberrypi:~/cplusthreads1$ make
make: *** No targets specified and no makefile found. Stop.
pi@raspberrypi:~/cplusthreads1$ make -f Makefile.txt
g++ -o dd_threads dd_threads.cpp -lm -std=c++11 -pthread -ltbb -lrt
pi@raspberrypi:~/cplusthreads1$ time -p ./dd_threads 1
max_ligand=1 nligands=120 nthreads=4
maximal score is 1, achieved by ligands
i e r n c p p c c n n w h c p c o i r p t o t r c e a w r n h i p o r w r r y y c a c h w e y o
r
real 0.02
user 0.02
sys 0.00
pi@raspberrypi:~/cplusthreads1$

```

### Run time of dd\_omp and dd\_threads with 2,3 and 4 threads.

Implementation	Time(s) 2 Threads	Time (s) 3 Threads	Time(s) 4 threads
dd_omp	0.02	0.04	0.34
dd_threads	0.02	0.04	0.23

```

pi@raspberrypi:~/openmp$ time -p ./dd_omp 2
max_ligand=2 nligands=120 nthreads=4
OMP defined
maximal score is 2, achieved by ligands
o p a c n o t n h h o r t a h c a r a p r r t o
real 0.02
user 0.03
sys 0.01
pi@raspberrypi:~/openmp$ cd ~
pi@raspberrypi:~$ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1$ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
n o o r a p t n r r t a a c h h o p a r h c t o
real 0.02
user 0.00
sys 0.03
pi@raspberrypi:~/cplusthreads1$

pi@raspberrypi:~/cplusthreads1$ time -p ./dd_threads 3
max_ligand=3 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
w h t h o p c h w
real 0.04
user 0.11
sys 0.00
pi@raspberrypi:~/cplusthreads1$ cd ~
pi@raspberrypi:~$ cd openmp1
pi@raspberrypi:~/openmp1$ time -p ./dd_omp 3
max_ligand=3 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
h o p c h w w h t
real 0.04
user 0.12
sys 0.01
pi@raspberrypi:~/openmp1$

```

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=4 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
ordt hkic rdry ihjo wht cio orei hch
real 0.34
user 0.82
sys 0.02
pi@raspberrypi:~/openmp1 $ cd ~
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4
max_ligand=4 nligands=120 nthreads=4
maximal score is 3, achieved by ligands
hkic orei ihjo rdry hch ordt cio wht
real 0.23
user 0.80
sys 0.02
pi@raspberrypi:~/cplusthreads1 $

```

## 2.3 Discussion Questions

1. Which approach is the fastest?

**The dd\_threads approach is the fastest. As you can see from the table and the screenshots, the omp and threads solution have the same run time. However, with 4 threads the omp took more time than the threads approach.**

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

The threads file has **208** lines. The omp file has **194** lines. The sequential file has **171** lines. The threads program has more lines than the omp program.

```

pi@raspberrypi: ~/sequential
File Edit Tabs Help
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads.cpp
208 dd_threads.cpp
pi@raspberrypi:~/cplusthreads1 $ cd ~
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ wc -l dd_omp.cpp
194 dd_omp.cpp
pi@raspberrypi:~/openmp1 $ cd ~
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ wc -l dd_sequential.cpp
wc: dd_sequential.cpp: No such file or directory
pi@raspberrypi:~/sequential $ wc -l dd_serial.cpp
171 dd_serial.cpp
pi@raspberrypi:~/sequential $

```

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

Serial = 5.58  
 Omp = 2.07  
 Threads = 1.43

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
maximal score is 5, achieved by ligands
hoach
real 2.07
user 5.58
sys 0.01
pi@raspberrypi:~/openmp1 $ cd ~
pi@raspberrypi:~ $ cd sequential
pi@raspberrypi:~/sequential $ time -p ./dd_serial 5
maximal score is 5, achieved by ligands
hoach
real 5.58
user 5.58
sys 0.00
pi@raspberrypi:~/sequential $ cd ~
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5
max_ligand=5 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
hoach
real 1.43
user 5.58
sys 0.00
pi@raspberrypi:~/cplusthreads1 $

```

Increase the maximum ligand length to 7 and rerun each program. What is the run time for each? By changing the first input, which is the maximum ligand length of the programs to 7 as you can see in the bottom screenshots, I got those runtimes for omp and the threads program.

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/sequential $ cd ~
pi@raspberrypi:~ $ cd openmp1
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 7
max_ligand=7 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 161.31
user 287.39
sys 0.02
pi@raspberrypi:~/openmp1 $ cd ~
pi@raspberrypi:~ $ cd cplusthreads 1
bash: cd: too many arguments
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 7
max_ligand=7 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 84.15
user 288.15
sys 0.05
pi@raspberrypi:~/cplusthreads1 $

```

## Links to Slack, GitHub, YouTube

Slack Channel: georgia-state-cs → proj-assemblychefs  
<https://app.slack.com/client/TNCC77K3K/GTDJAQL05>

YouTube – [https://youtu.be/\\_fS6DxK\\_UQU](https://youtu.be/_fS6DxK_UQU)

GitHub - <https://github.com/AssemblyChefs/CSC3210-AssemblyChefs/tree/master/Project%20A2%20Files>

# Screenshots

