

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

**Spring 2019**

**Group Name: ATLAS-SQUAD**

**Team Members: Zeak Sims(Team Coordinator)**

**Jason Poston**

**T'Avvion Jones**

**Sai Rampally**

**Shili Guan**

## Planning and Scheduling

Assignee Name	Email	Task	Duration (hrs)	Dependency	Due Date	Note
Zeak Sims (Coordinator)	zsims2@student.gsu.edu	<ul style="list-style-type: none"> <li>Task 1, 2, 4, 5, 6</li> <li>Return and format pi</li> <li>Turn in digital and physical report</li> <li>Parallel programming</li> </ul>	5 hrs	Parallel programming	04/15/19	.
T'Avvion Jones	Tjones172@student.gsu.edu	<ul style="list-style-type: none"> <li>Task 2, 3, 4, 5, 6</li> <li>Record presentation</li> <li>Parallel programming</li> <li>Task 3 part a 1 &amp; 2</li> </ul>	5 hrs	Parallel programming Record presentation	04/15/19	
Shili Guan	sguan2@student.gsu.edu	<ul style="list-style-type: none"> <li>Task 2, 3, 4, 5, 6</li> <li>Parallel programming</li> <li>Task</li> <li>Task 3 part a 3, 4 &amp; 5</li> </ul>	5 hrs	Parallel programming	04/15/19	
Sai Rampally	srampally1@student.gsu.edu	<ul style="list-style-type: none"> <li>Task 2, 3, 4, 5, 6</li> <li>Upload to GitHub</li> <li>Task 3 part a 6 &amp; 7</li> </ul>	5 hrs	Parallel programming GitHub	04/15/19	
Jason Poston	jposton1@student.gsu.edu	<ul style="list-style-type: none"> <li>Task 2, 3, 4, 5, 6</li> <li>Parallel programming</li> <li>Task 3 part a 8 &amp; 9</li> </ul>	5 hrs	Parallel programming	04/15/19	

## **Parallel Programming Skills**

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

1. Identifying the task that can be done concurrently/independently
2. Master/Worker implementation technique:
  - a. Master: Splits data up according to the number of workers and sends each worker a section, then, receives results from each worker.
  - b. Worker: Processes given data from Master then returns the results back to the master.

In example, below there is a parallel program.

```
Task = 15;           //number of task that need to be done
W = 5;               // number of workers
TpW = Task/W;        //number of task per worker
```

```
//each worker does the following:
//assign each student a number
For(int i = 0; i < tpW; i++){
    int student = 0;
    System.out.println("Student number:" + student);
}
```

The master then receives the workers assigned task and prints all of the student numbers.

### **What is MapReduce?**

MapReduce is a parallel programming technique for processing big data. It allows for simpler computations of a large amount of data on multiple processors.

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

Map takes as input a function and sequence of values, then it applies the function to each value in the sequence. A reduce uses a binary operation to combines all the elements of a sequence.

### **Why MapReduce?**

MapReduce as the name that shows it has both feature from Map and Reduce. Google developed it as a mechanism for large amount of raw data. MapReduce has some great feature that can help task complete more efficient. It allows Google engineers to accomplish simple computer arithmetic while hiding the details of parallelization, data distribution, load balancing. It also is fault tolerance. It is a great tool to use for large amount of data.

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

The reading has provided few great examples for using MapReduce. Here is one of them that I think it's very easy to get the idea for large data.

Example: Count if URL Access Frequency: The map function processes log 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.

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

The MapReduce model is executed when the Map invocations are distributed across multiple machines. After distributing the invocations, the input data will get partitioned and set into the M splits or shards. And then, the input shards will be processed in parallel and on multiple different machines. Now, partitioning the intermediate key space distributes the Reduce invocations into R pieces using a partitioning function (e.g.,  $\text{hash}(\text{key}) \bmod R$ ). The user then specifies number of partitions and the partitioning function. When all the map and reduce tasks have been completed, the master wakes up the user program. Then the MapReduce call in the user program will return to the user code.

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

- Distributed Grep
- Reverse Web-Link Graph
- Inverted Index

Distributed Grep computation occurs when the map function emits a line and if it matches a given pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.

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

Inverted Index computation occurs when the map function parses each document, and emits a sequence of <word, document ID> pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)>pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

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

Open MP is used for when you want to introduce shared memory parallelism to your code. Using Open MP allows the user to split (for example) loops into multiple threads so that each one can work simultaneously, while reducing the workload of the loop. MPI (Message Passing Interface) is typically used to develop tightly synchronous code and well load balanced. MPI can be used to develop parallel code that runs over MULTIPLE machines. Hadoop MapReduce can be used when you have huge amounts of data that you would like to extract, transform and load.

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

The simplest and most basic elementary way of explaining the Drug Design and DNA problem is, design a solution (ligands) to fit a protein (problem) and see how well it works. Designing multiple solutions to see which fits the problem the best, and works the most efficient. There is more than one way to reach the solution, however what we as designers (and what the Drug designers) want are the solutions (binding best with the protein) with the highest 'score' which is the most efficient. Once you identify the one with the best score.

# **Parallel Programming Basics: Drug Design and DNA in Parallel**

## **1) Drug Design**

In this project we learn the basic concept of designing a drug and further observe this first hand through molecular modeling computation. These programs take proteins and try to match ligands that match up best with these proteins to find possible drugs.

That strategy will use to approach the problem is map-reduce and has three stages. The first stage will include `Generate_task()` function that'll produce various ligand and protein pairs. The next stage of the process will utilize `map()` function in order to produce binding scores to ligands. The last stage will then use the `reduce()` function to find us the highest scoring ligand pair.

In this project we observed two main implementations of modeling computation. We observed one sequential process and two parallel processes that included multiple processes and threads.

## **2) Sequential Solution**

We first took a look at the sequential solution that really serves as our baseline that the codes we observed later built upon. To get started, we downloaded the DNA solutions onto the Pi, which contained all the codes that we needed. Once everything was ready, we created a new directory named sequential, so we could copy then implement the serial cpp solution and a Makefile to build the solution.

```
pi@raspberrypi: ~/sequential
File Edit Tabs Help
pi@raspberrypi:~$ cd sequential
pi@raspberrypi:~/sequential$ ls
2019-04-14-230321_1184x624_scrot.png dd_serial dd_serial.cpp Makefile zeak
pi@raspberrypi:~/sequential$ cat Makefile
CXX=g++
dd_serial: dd_serial.cpp
        $(CXX) -o dd_serial dd_serial.cpp
clean:
        rm dd_serial
pi@raspberrypi:~/sequential$ cat dd_serial.cpp
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <stdlib>

#define DEFAULT_max_ligand 7
#define DEFAULT_nligands 120
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {
    int key;
    string val;
    Pair(int k, const string &v) {key = k; val = v;}
};
```

```
pi@raspberrypi: ~/sequential
File Edit Tabs Help
int MR::Reduce(int key, const vector<Pair> &pairs, int index, string &values) {
    while (index < pairs.size() && pairs[index].key == key)
        values += pairs[index++].val + " ";
    return index;
}

/* class Help methods */

// returns arbitrary string of lower-case letters of length at most max_ligand
string Help::get_ligand(int max_ligand) {
    int len = 1 + rand() % max_ligand;
    string ret(len, '?');
    for (int i = 0; i < len; i++)
        ret[i] = 'a' + rand() % 26;
    return ret;
}

int Help::score(const char *str1, const char *str2) {
    if (*str1 == '\0' || *str2 == '\0')
        return 0;
    // both argument strings non-empty
    if (*str1 == *str2)
        return 1 + score(str1 + 1, str2 + 1);
    else // first characters do not match
        return max(score(str1, str2 + 1), score(str1 + 1, str2));
}

pi@raspberrypi:~/sequential$ scrot
pi@raspberrypi:~/sequential$ make
make: 'dd_serial' is up to date.
pi@raspberrypi:~/sequential$ scrot
```

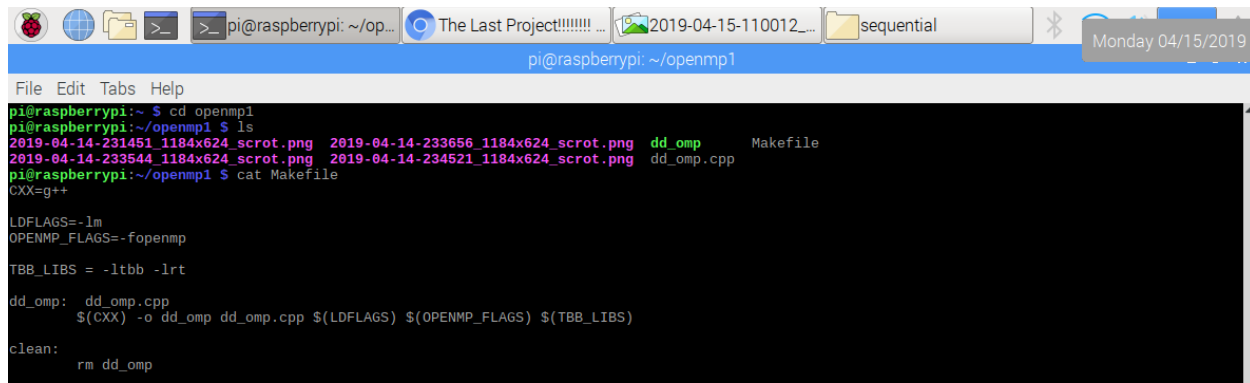
The first thing that we notice is how long it did take to process everything. There was a brief moment where we thought we might have done something wrong, but after the command line return we looked back into the directory and notice the new file. We did not understand much of it until we read on in parallel programming A5. We looked back at the code that we copied, and after walking through it with the notes on the code, we noticed all three stages being implemented. Having the code processing everything in a serial approach really helped us understand how drug processing software works. It completes one task to completion before moving on to the next one, which made it extremely easy to follow. It also explained why it took a slight delay for everything to compile. Analyzing the code, we first noticed the `Generate_task` functions that was producing the pairs. After that we saw how queue and vector were utilize to hold the pairs put together by the `Map()` and sorted by the `Reduce()` to find the pair with the highest score.

### 3) OpenMP Solution

The next implementation that we compiled and observed was one of the two parallel approaches. This code takes the `Map()` function of our previous code and parallelized it using OpenMP. We made the `Openmp 1` directory and copied the codes in there that we needed, but it did not compile at all. For this portion of the programming, we had to install the TBB library since it was not something that came along with the raspbian software we installed on the Pi. We read through the notes about the code first, but we simply just copied and pasted the codes and tried to compile them, but we were met with error. We first thought it was an error with implementing OpenMP, because we read how the OpenMP approach could result in errors due



multiple threads calling the Map() function leading to interference and overlap between the threads, but we realize that the TBB is there to be a thread-safe concurrent\_vector container for the pairs produce by the Map() function. Then we went on to realize that the error had to do with the TBB library, and the fact that we did not have the library installed in order to call it. But after installing the library, we were able to just compile the code just as we did for the serial solution.



```

pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~$ cd openmp1
pi@raspberrypi:~/openmp1$ ls
2019-04-14-231451_1184x624_scr0t.png  2019-04-14-233656_1184x624_scr0t.png  dd_omp      Makefile
2019-04-14-233544_1184x624_scr0t.png  2019-04-14-234521_1184x624_scr0t.png  dd_omp.cpp
pi@raspberrypi:~/openmp1$ cat Makefile
CXX=g++

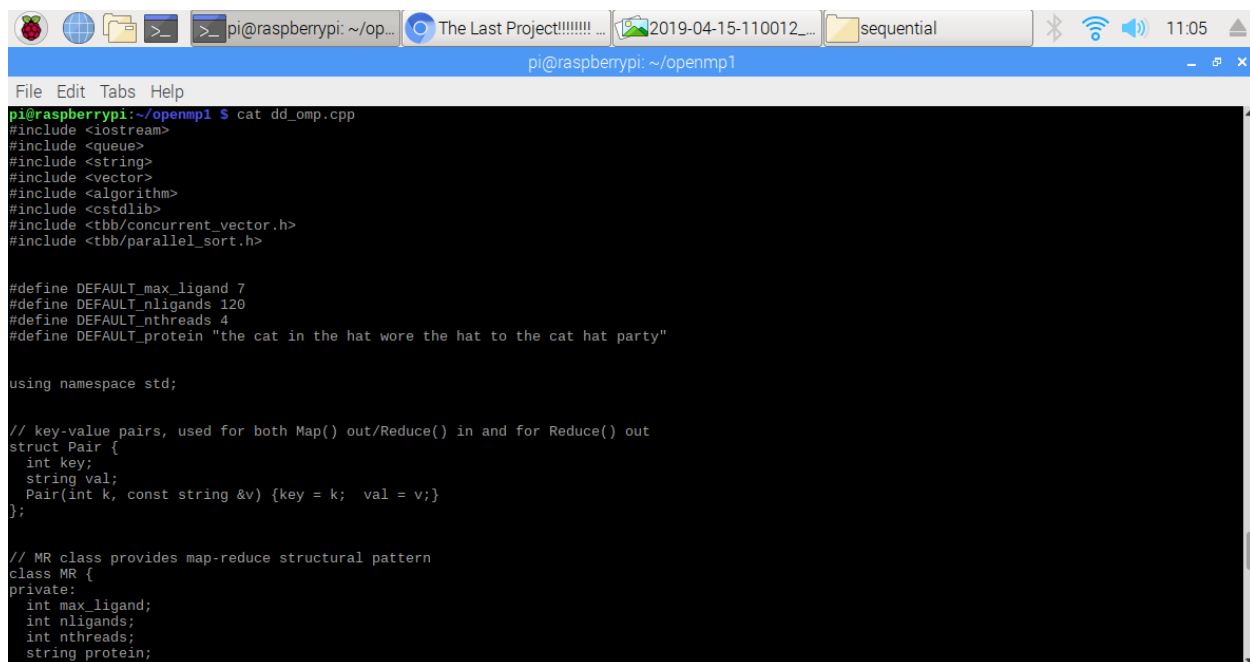
LD_FLAGS=-lm
OPENMP_FLAGS=-fopenmp

TBB_LIBS = -ltbb -lrt

dd_omp: dd_omp.cpp
$(CXX) -o dd_omp dd_omp.cpp $(LD_FLAGS) $(OPENMP_FLAGS) $(TBB_LIBS)

clean:
rm dd_omp

```



```

pi@raspberrypi:~/openmp1$ cat dd_omp.cpp
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <tbb/concurrent_vector.h>
#include <tbb/parallel_sort.h>

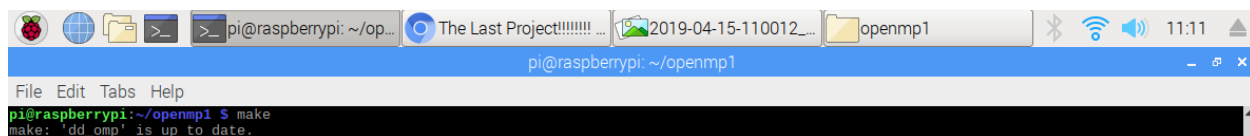
#define DEFAULT_max_ligand 7
#define DEFAULT_nligands 120
#define DEFAULT_nthreads 4
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {
    int key;
    string val;
    Pair(int k, const string &v) {key = k; val = v;}
};

// MR class provides map-reduce structural pattern
class MR {
private:
    int max_ligand;
    int nligands;
    int nthreads;
    string protein;

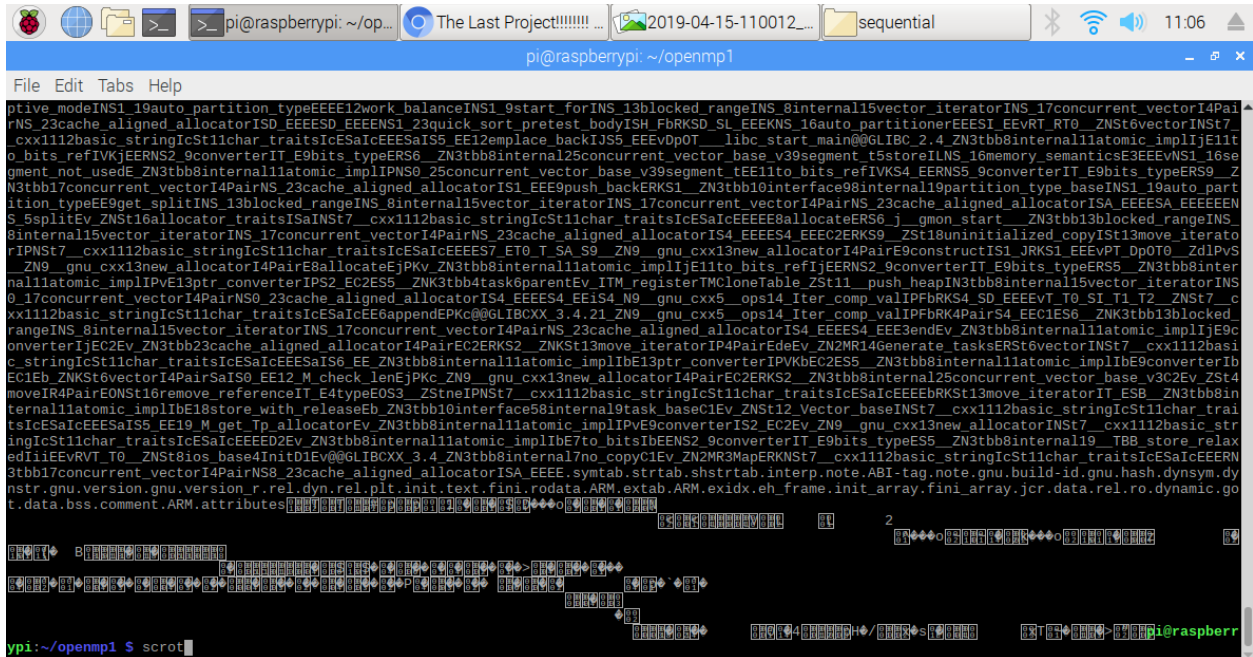
```



```

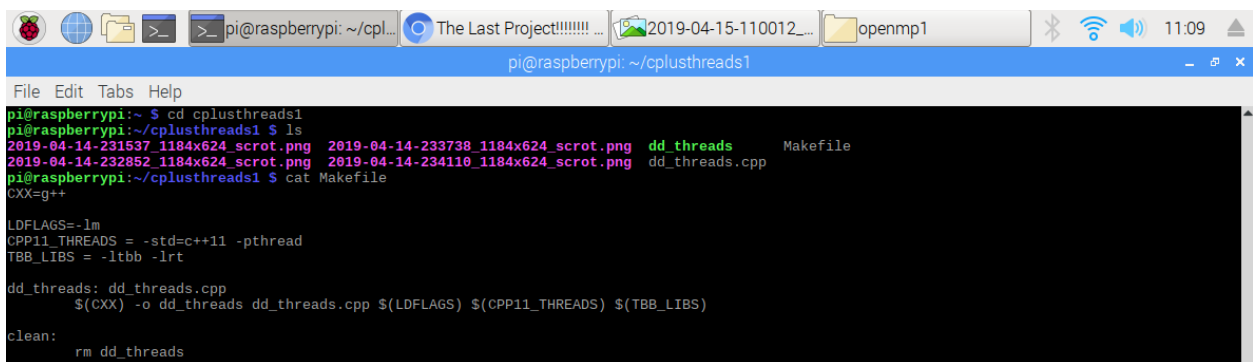
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1$ make
make: 'dd_omp' is up to date.

```



#### 4) C++11 Threads Solution

For the second parallel solution, compilation went way easier because the TBB library was already installed, and we really got to just focus on what makes the C++11 implementation unique to the OpenMP approach and analyze which parallel solution could be more efficient. This was the easiest code to compile and went smoothly. We simply just made the cplusthreads directory, copied and pasted the Makefile and cpp code, built it, and finally analyzed the output. The difference here compared to the OpenMP, is now we need a master worker to create and manage our threads. The difference in time it takes to build each parallel code is unnoticeable to the human eye, so out right, we could not really tell which was the most efficient of the two codes.



```
pi@raspberrypi: ~/cpl... The Last Project!!!!!! ... 2019-04-15-110012... openmp1
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/cplusthreads1 $
pi@raspberrypi:~/cplusthreads1 $
pi@raspberrypi:~/cplusthreads1 $ cat dd_threads.cpp
/** drug design example with C++11 threads and tbb containers */
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <tbb/concurrent_vector.h>
#include <tbb/concurrent_queue.h>
#include <tbb/parallel_sort.h>
#include <thread>

#define DEFAULT_max_ligand 7
#define DEFAULT_nligands 120
#define DEFAULT_nthreads 4
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {
    int key;
    string val;
    Pair(int k, const string &v) {key = k; val = v;}
};

// MR class provides map-reduce structural pattern
class MR {
```

```
pi@raspberrypi: ~/cpl... The Last Project!!!!!! ... 2019-04-15-110012... openmp1
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
    string &values) {
    while (index < pairs.size() && pairs[index].key == key)
        values += pairs[index++].val + " ";
    return index;
}

/* class Help methods */

// returns arbitrary string of lower-case letters of length at most max_ligand
string Help::get_ligand(int max_ligand) {
    int len = 1 + rand()%max_ligand;
    string ret(len, '?');
    for (int i = 0; i < len; i++)
        ret[i] = 'a' + rand() % 26;
    return ret;
}

int Help::score(const char *str1, const char *str2) {
    if (*str1 == '\0' || *str2 == '\0')
        return 0;
    // both argument strings non-empty
    if (*str1 == *str2)
        return 1 + score(str1 + 1, str2 + 1);
    else // first characters do not match
        return max(score(str1, str2 + 1), score(str1 + 1, str2));
}

pi@raspberrypi:~/cplusthreads1 $ scrot
pi@raspberrypi:~/cplusthreads1 $ make
make: 'dd_threads' is up to date.
pi@raspberrypi:~/cplusthreads1 $ scrot
```

## 5) Measure Run-Time

Implementation	Time(s)
dd_serial	130.18
dd_omp	.03
dd_threads	.02

```

pi@raspberrypi: ~/op... sequential pi@raspberrypi: ~
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/sequential $ cd ..
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
c y i o i w t c e c n h y n r p c c e w a w r r o i h y c n n r c p r c r p y p t h o r p e a w r o
real 0.03
user 0.00
sys 0.02

```

```

pi@raspberrypi: ~/cpl... sequential pi@raspberrypi: ~
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ cd ..
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
c w h n r t r t i a w o n n c p p r e c o p r p i c h c c e o c y o r w n y r y a i c r h e p r y w
real 0.02
user 0.01
sys 0.01

```

Implementation	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads
dd_omp	.02	.04	.20
dd_threads	.02	.04	.14

```

pi@raspberrypi: ~/op... openmp1 [2019-04-14-231451_... pi@raspberrypi: ~
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
o r h c n o t n a c a r o p t o t a h h r r a p
real 0.02
user 0.00
sys 0.03
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 3
max_ligand=3 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
w h t c h w h o p
real 0.04
user 0.12
sys 0.00
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=4 nligands=120 nthreads=4
OMP defined
maximal score is 3, achieved by ligands
h k i c o r d t h c h o r e l c i o i h j o r d r y w h t
real 0.20
user 0.56
sys 0.00

```

```

pi@raspberrypi: ~/cpl... cplusthreads1
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
to tn rr op no ac hh ap or ta hc ar
real 0.02
user 0.00
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ 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 0.11
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
hkic ore1 ihjo rdry hch cio wht ordt
real 0.14
user 0.45
sys 0.02

```

## Discussion Questions

### 1. Which approach is the fastest?

The C++ threads solution turned out to be fastest when it go to 4 threads. Before getting to 4 threads, the execution time were the same.

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

The C++ implementation had fewer lines of code than the OpenMP.

```

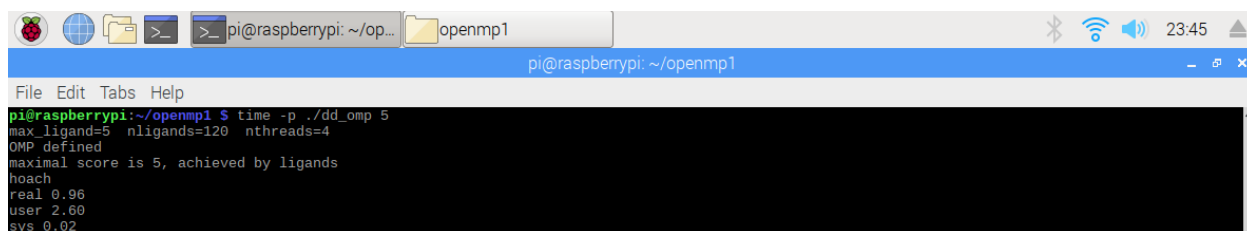
pi@raspberrypi: ~/op... openmp1
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ wc -l dd_omp
179 dd_omp

pi@raspberrypi: ~/cpl... openmp1
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads
157 dd_threads

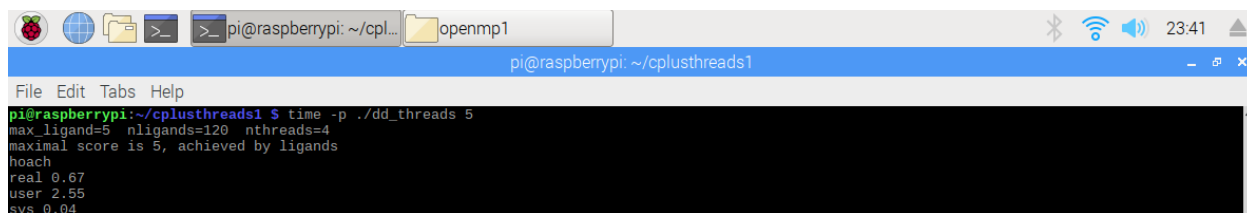
```

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

Implementation	Time(s) 5 Threads
dd_omp	.96
dd_threads	.67



```
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5
max_ligand=5 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
hoach
real 0.96
user 2.60
sys 0.02
```



```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5
max_ligand=5 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
hoach
real 0.67
user 2.55
sys 0.04
```

**4. Increase the maximum ligand length to 11, and rerun each program. What is the run time for each?**

As you can see in the screenshots below, this is the data I collected, but I ran execution time again for dd\_omp and dd\_threads a was presented with different results in another set of charts and screen shots.

Implementation	Time(s) 1 Threads ligand length 11
dd_serial	130.18
dd_omp	.01
dd_threads	.02

The image consists of three vertically stacked screenshots of a Raspberry Pi terminal window. The window title is 'pi@raspberrypi: ~/sequential'. The terminal shows the following commands and output:

```
pi@raspberrypi:~/sequential $ nano dd_serial.cpp
pi@raspberrypi:~/sequential $ cat dd_serial.cpp
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>

#define DEFAULT_max_ligand 11
#define DEFAULT_nligands 120
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {
    int key;
    string val;
    Pair(int k, const string &v) {key = k; val = v;}
};

// MR class provides map-reduce structural pattern
class MR {
private:
    int max_ligand;
    int nligands;
    string protein;
```

The second screenshot shows the compilation command:

```
pi@raspberrypi:~/sequential $ make
g++ -o dd_serial dd_serial.cpp
```

The third screenshot shows the execution command and its output:

```
pi@raspberrypi:~/sequential $ time -p ./dd_serial

^C
real 1546.42
user 1546.29
sys 0.01
```



```
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ cat dd_omp.cpp
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <tbb/concurrent_vector.h>
#include <tbb/parallel_sort.h>

#define DEFAULT_max_ligand 11
#define DEFAULT_nligands 120
#define DEFAULT_nthreads 4
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {
    int key;
    string val;
    Pair(int k, const string &v) {key = k; val = v;}
};

// MR class provides map-reduce structural pattern
class MR {
private:
    int max_ligand;
    int nligands;
    int nthreads;
    string protein;
```

```
pi@raspberrypi:~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ cd openmp1
pi@raspberrypi:~/openmp1 $ ls
2019-04-14-231451_1184x624_scr0t.png 2019-04-14-234521_1184x624_scr0t.png 2019-04-15-110626_1184x624_scr0t.png dd_omp.cpp
2019-04-14-233544_1184x624_scr0t.png 2019-04-15-110537_1184x624_scr0t.png 2019-04-15-111115_1184x624_scr0t.png Makefile
2019-04-14-233656_1184x624_scr0t.png 2019-04-15-110600_1184x624_scr0t.png dd_omp
pi@raspberrypi:~/openmp1 $ nano dd_omp.cpp
pi@raspberrypi:~/openmp1 $ scrot
pi@raspberrypi:~/openmp1 $ make
g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/openmp1 $ scrot
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=1 nligands=120 nthreads=4
OMP defined
maximal score is 1, achieved by ligands
r i t p o h y i w c w c h p r i y o c c e e w a c y r n c p o r y n p n r w h p c e r a r n o r c t
real 0.01
user 0.00
sys 0.02
```

```
pi@raspberrypi: ~/cpl... The Last Project!!!!!!! 2019-04-15-110012... cplusthreads1
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
arithmetic3.c.save.1 Desktop masterWorker project 5 screen shot Templates
pi@raspberrypi:~$ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1$ clear

pi@raspberrypi:~/cplusthreads1$ ls
2019-04-14-231537_1184x624_scrot.png 2019-04-14-234110_1184x624_scrot.png 2019-04-15-111010_1184x624_scrot.png dd_threads.cpp
2019-04-14-232852_1184x624_scrot.png 2019-04-15-110916_1184x624_scrot.png 2019-04-15-111034_1184x624_scrot.png Makefile
2019-04-14-233738_1184x624_scrot.png 2019-04-15-110955_1184x624_scrot.png dd_threads

pi@raspberrypi:~/cplusthreads1$ nano dd_threads.cpp
pi@raspberrypi:~/cplusthreads1$ cat dd_threads.cpp
/** drug design example with C++11 threads and tbb containers */
#include <iostream>
#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <tbb/concurrent_vector.h>
#include <tbb/concurrent_queue.h>
#include <tbb/parallel_sort.h>
#include <thread>

#define DEFAULT_max_ligand 11
#define DEFAULT_nligands 120
#define DEFAULT_nthreads 4
#define DEFAULT_protein "the cat in the hat wore the hat to the cat hat party"

using namespace std;

// key-value pairs, used for both Map() out/Reduce() in and for Reduce() out
struct Pair {

pi@raspberrypi:~/cplusthreads1$ make
g++ -o dd_threads dd_threads.cpp -lm -std=c++11 -pthread -ltbb -lrt
pi@raspberrypi:~/cplusthreads1$ scrot
pi@raspberrypi:~/cplusthreads1$ time -p ./dd_threads 1
max_ligand=1 nligands=120 nthreads=4
maximal score is 1, achieved by ligands
r r n c p p o c h c n n w c i c r p p i t o r t c e a w r h n p w h r y o i c y r w r y a e c y o e
real 0.02
user 0.01
sys 0.01
```

After executing it another two times, this data table contains the definitive times of the execution

Implementation	Time(s) 1 Threads ligand length 11
dd_serial	130.18
dd_omp	.02
dd_threads	.01



# APPENDIX

**Slack Account:** <https://atlas-squad.slack.com/messages/CFR6D9LHJ/team/>

**GitHub** <https://github.com/ATLAS-SQUAD/CSc-3210>

**Youtube Channel:** <https://www.youtube.com/watch?v=fnobwjwvPqk&feature=youtu.be>

