

Developing Soft and Parallel Programming Skills Using Project-Based Learning

Team HUSTLE

Spring-2019

Smit Patel, Lucky Phan, Eseosasere (Eseosa) Omobude, Hwysun (Harry) Park, Tommy Lim

Task 1: Planning and Scheduling

Name	E-mail	Tasks	Duration	Note
Smit Patel (Coordinator)	spatel261@student.gsu.edu	Allocate tasks, summarize project into a report, ensure everyone stays on schedule, update GitHub	3 hours	Check on Slack, update GitHub, review report, Final Report due Mon, 04/15/19
Lucky Phan	lphan7@student.gsu.edu	Parallel Programming Basics	4 hours	Parallel Programming due on Mon, 04/15/19
Eseosasere Omobude	eomobude1@student.gsu.edu	Answer the questions for Parallel Programming Skills Foundation Part	5 hours	Parallel Programming Skills questions due Mon, 04/15/19
Hwysun Park	hpark44@student.gsu.edu	Parallel Programming Basics and Lab report	5 hours	Parallel Programming and Report due on Mon, 04/15/19
Tommy Lim	tlim3@student.gsu.edu	Record, edit, and upload group presentation onto YouTube	2.5 hours	Set recording date on Wed, 04/10/19. Video due Mon, 04/15/19

Task 2: Collaboration

Slack:

Team HuSTLE ✓ Smit Patel

🔍 All Threads

Channels

- # general
- # random

Direct Messages

- Slackbot
- Smit Patel (you)
- Eseosa Omobude
- Harry
- Kexin Ding
- Lucky
- Tommy

+ Invite people

Apps

#general ☆ | 👤 6 | ⭐ 1 | Company-wide announcements and work-based m: Yesterday

Still figuring out every time there comes a different error

Harry 5:55 PM
Sorry, I am on my way home after work. I will finish my part hopefully by tonight and try to figure out the openmp error after then

Lucky 6:02 PM
@Harry Ok keep us updated if any new error comes up, we will help google

Eseosa Omobude 6:54 PM
@Smit Patel let me know what you guys think and if I misunderstood some of the questions please!

Word Document

Task3A5.docx
99 kB Word Document

Task3:
(15p)
What are the basic steps (show all steps) in building a parallel program? Show at least one example.

- Designing and developing parallel programs has characteristically been a very manual process. The programmer is typically responsible for both identifying and implementing parallelism.

Understand the Problem and the Program
Before spending time in an attempt to develop a parallel solution for a problem, determine whether or not the problem is one that can actually be parallelized.

- Example of an easy to parallelize problem:

+ Message #general

Github:

team-hustle / team-hustle > Projects > CSC 3210 - Team HuSTLE

Filter cards

+ Add cards

Exit fullscreen

Menu

To Do 0

In Progress 0

Done 28

- A5: Task 6
Return the Raspberry Pi
Added by team-hustle
- A5: Task 5
Final Report
Added by team-hustle
- A5: Task4
Video Presentation
REcord, edit, and upload the group's presentation onto the team's YouTube
Added by team-hustle
- A5: Task 3B
Parallel Programming Basics
Added by team-hustle
- A5: Task 2
Collaboration
Added by team-hustle
- A5: Task3A
Parallel Programming Skills Foundation

Task 3: Parallel Programming Skills (Part-A Foundation)

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

- Designing and developing parallel programs has characteristically been a very manual process. The programmer is typically responsible for both identifying and implementing parallelism.

Understand the Problem and the Program

Before spending time in an attempt to develop a parallel solution for a problem, determine whether or not the problem is one that can actually be parallelized.

- Example of an easy to parallelize problem:

Calculate the potential energy for each of several thousand independent conformations of a molecule. When done, find the minimum energy conformation.

- Little to no parallelism

**Calculation of the Fibonacci series (0,1,1,2,3,5,8,13,21,...) by use of the formula:
 $F(n) = F(n-1) + F(n-2)$**

- The calculation of the $F(n)$ value uses those of both $F(n-1)$ and $F(n-2)$, which must be computed first.

Identify the programs hotspots

- Know where most of the real work is being done. The majority of scientific and technical programs usually accomplish most of their work in a few places.
- Focus on parallelizing the hotspots and ignore those sections of the program that account for little CPU usage

Identify bottlenecks in the program

- Are there areas that are disproportionately slow, or cause parallelizable work to halt or be deferred? For example, I/O is usually something that slows a program down.

Identify inhibitors to parallelism.

- One common class of inhibitor is *data dependence*, as demonstrated by the Fibonacci sequence above.

Investigate other algorithms if possible

Take advantage of optimized thirdparty parallel software and highly optimized math libraries available from leading vendors (IBM's ESSL, Intel's MKL, AMD's AMCL, etc.).

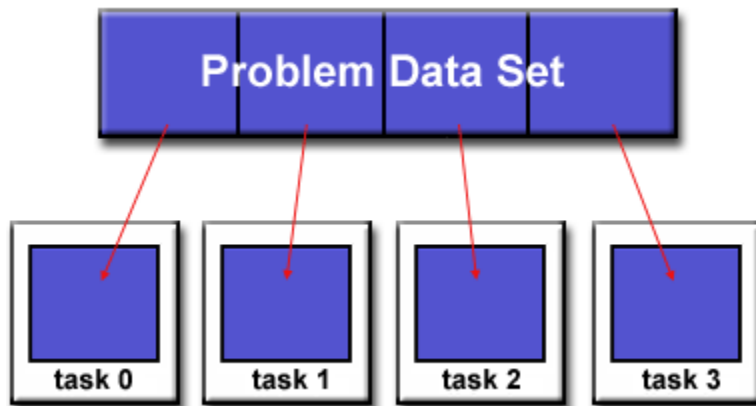
Designing Parallel Programs

Partitioning

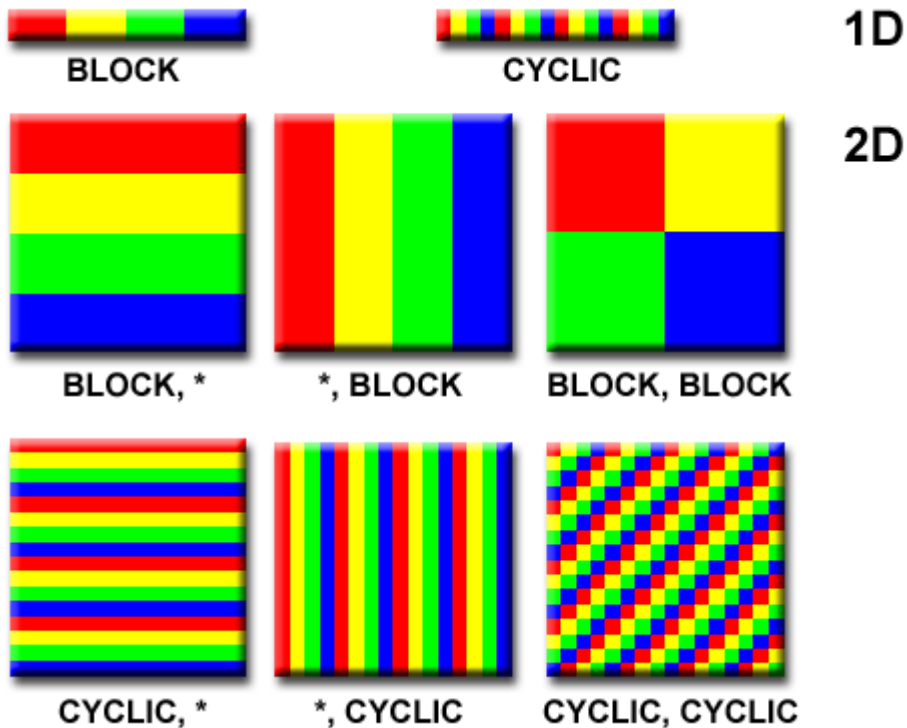
- One of the first steps in designing a parallel program is to break the problem into discrete "chunks" of work that can be distributed to multiple tasks. This is known as decomposition or partitioning.
- There are two basic ways to partition computational work among parallel tasks: **domain decomposition** and **functional decomposition**.

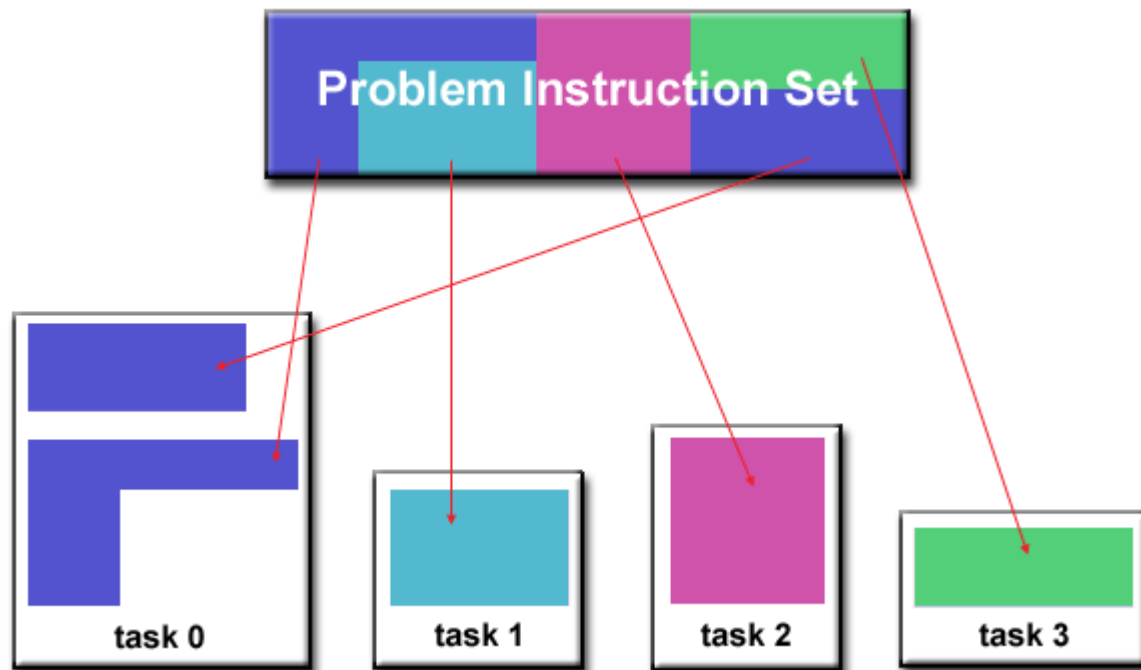
► Domain Decomposition:

- The data associated with a problem is decomposed. Each parallel task then works on a portion of the data.



- There are different ways to partition data:





- Functional decomposition lends itself well to problems that can be split into different tasks.

(5p)

What is MapReduce?

- Map Reduce is a programming model in addition to an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

(10p)

What is map and what is reduce?

- **Map stage** – The map or mapper's job is to process the input data. The input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
- **Reduce stage** – This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

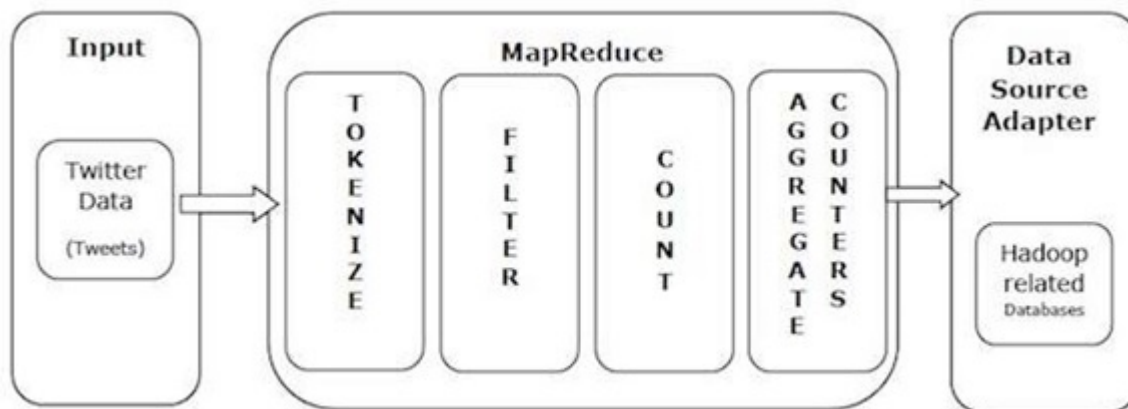
(5p)

Why MapReduce?

- MapReduce is a framework for long parallel computations that use large data sets and a lot of nodes. In addition to this, it also uses data that is stored locally on a node where the job is being executed. The computations are parallel because there is no communication between them. They run independent of one another.
- MapReduce is also Fault-Tolerant. This means that each node periodically reporting its status to a master node. If a node doesn't respond as expected, the master node re-assigns that piece of the job to other available nodes in the cluster. This creates resiliency and makes it practical for MapReduce to run on inexpensive commodity servers.

(5p)

Show an example for MapReduce.



- **Tokenize:** Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter:** It filters the unwanted words from maps of tokens.
- **Count:** Generates a token counter per word.
- **Aggregate counters:** Prepares a combination of similar counter values into small manageable units.

(10p)

Explain in your own words how MapReduce model is executed?

- MapReduce has two steps. The first step, the "Map" step, takes the input and breaks it into smaller sub-problems and distributes them to the worker nodes. The worker nodes then send their results back to the "master" node. The second step, the "Reduce" step, takes the results from the worker nodes and combines them in some manner to create the output, which is the output for the original job. Hadoop was designed to support MapReduce from the beginning.

(6p)

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

1)

configure

void **configure**([JobConf](#) job)

Initializes a new instance from a [JobConf](#).

Parameters:

job - the configuration

2) map

void **map**([K1](#) key,

[V1](#) value,

[OutputCollector](#)<[K2](#),[V2](#)> output,

[Reporter](#) reporter)

throws [IOException](#)

Maps a single input key/value pair into an intermediate key/value pair.

Output pairs need not be of the same types as input pairs. A given input pair may map to zero or many output pairs. Output pairs are collected with calls

to [OutputCollector.collect\(Object, Object\)](#).

Applications can use the [Reporter](#) provided to report progress or just indicate that they are alive.

In scenarios where the application takes an insignificant amount of time to process individual key/value pairs, this is crucial since the framework might assume that the task has timed-out and kill that task. The other way of avoiding this is to set [mapred.task.timeout](#) to a high-enough value (or even zero for no time-outs).

Parameters:

key - the input key.

value - the input value.

output - collects mapped keys and values.

reporter - facility to report progress.

Throws:

[IOException](#)

3) collect

void **collect**([K](#) key,

[V](#) value)

throws [IOException](#)

Adds a key/value pair to the output.

Parameters:

key - the key to collect.

value - to value to collect.

Throws:

[IOException](#)

(6p)

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

- We use OpenMP when we need to split the loop into multiple threads, so each of them can handle a specific chunk of the loop's iterations.
- We use MPI when we want to develop any parallel code that runs over multiple machines. If the programmer knows how to, they can mix MPI with OpenMP to use threads within machines. This is usually known as hybrid programming.
- We use MapReduce (Hadoop) usually when you have terabytes of data that you want to do ETL (extract, transform and load) like operations. One of the most important features when using Hadoop is something called **fault tolerance**. but Hadoop is really designed to use MapReduce as the primary method of interaction.

(14p)

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

- DNA is used as the set of instruction set (this is kind of like a computer science program code to complete a task). When it comes to medicines, the proteins shape usually determines the function that it does in the human's body. In order create a drug, we must find a way to fix or change the proteins shape. There are many steps that we need to take to create a drug design software. You generate ligands, compute a score for all of them and then identify the highest scoring ones and then use them for testing. It usually doesn't take long to generate the ligands. However, it does take a long time to score the ligands. The process for identifying the highest scoring ligands is fast as well.

Task 3: Parallel Programming Basics (Part-B)

A Sequential Solution Compilation:

```
pi@raspberrypi:~ $ mkdir sequential
pi@raspberrypi:~ $ cd Drug_DNA_Solutions
pi@raspberrypi:~/Drug_DNA_Solutions $ cd serial
pi@raspberrypi:~/Drug_DNA_Solutions/serial $ cp ~/dd_serial.cpp /home/pi/sequential
pi@raspberrypi:~/Drug_DNA_Solutions/serial $ cp ~/Makefile /home/pi/sequential
pi@raspberrypi:~ $ g++ -o dd_serial dd_serial.cpp
```

Made the directory using -mkdir

-cd changes the current directory

We cd into the directory we wanted to copy files from by using -cp

Compiled by using g++ -o dd_serial dd_serial.cpp

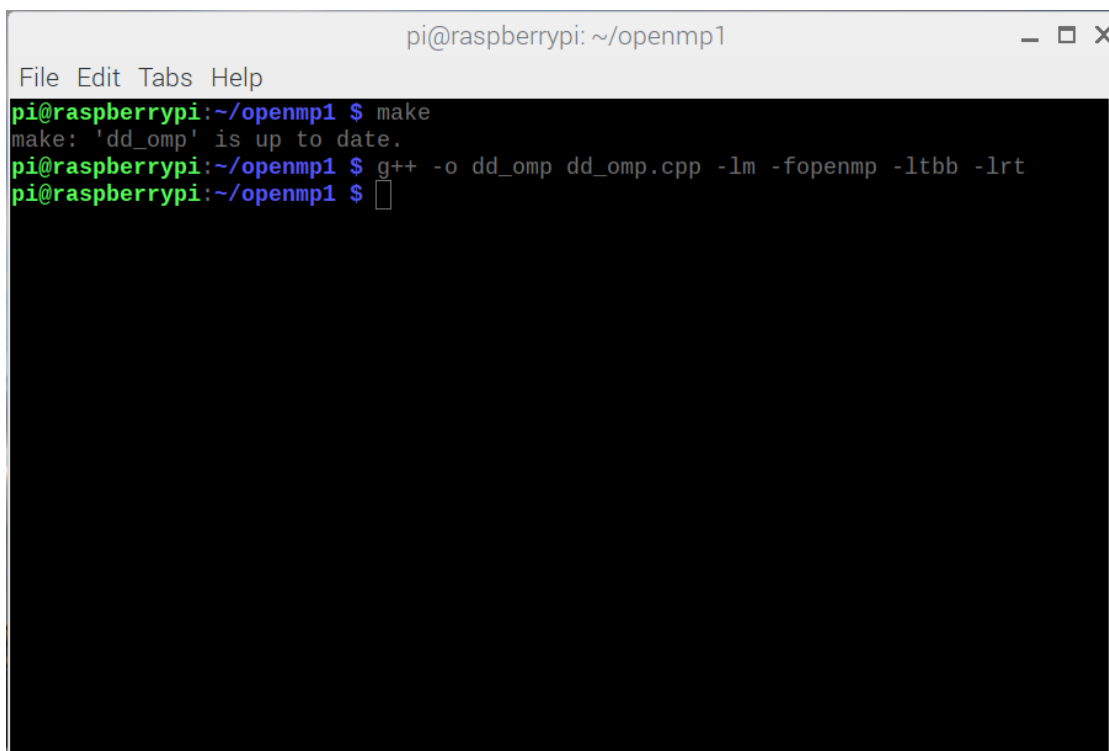
-We have encountered the following error when trying to compile the OpenMP solution and C++ 11 threads solution:

/usr/local/include/tbb/machine/gcc_armv7.h:31:2 error: #error compilation requires an ARMv7-a architecture.

To resolve the error, we followed the steps in these three websites:

- 1) <https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/>
 - To pip install OpenCV on Raspberry Pi
- 2) <https://www.theimpossiblecode.com/blog/intel-tbb-on-raspberry-pi/>
 - To build a TBB package for Raspberry Pi

3. OpenMP Solution Compilation



```
pi@raspberrypi: ~/openmp1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ make
make: 'dd_omp' is up to date.
pi@raspberrypi:~/openmp1 $ g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/openmp1 $
```

4. C++11 Threads Solution Compilation

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~ $ cd cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ make
make: 'dd_threads' is up to date.
pi@raspberrypi:~/cplusthreads1 $ g++ -o dd_threads dd_threads.cpp -lm -std=c++11
-lpthread -ltbb -lrt
pi@raspberrypi:~/cplusthreads1 $
```

5.

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 135.29
user 135.25
sys 0.00
pi@raspberrypi:~/sequential $ 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 w w r r o n n e r c c h o c i y y c c t e a y e w p c a i c h y n r r r o n o p
i p w p h t r p
real 0.02
user 0.01
sys 0.01
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
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 n r y w r i y a
l o c r h e w y r p
real 0.02
user 0.01
sys 0.01
```

Measure Run-Time

```
pi@raspberrypi: ~/cplusthreads1
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
op no hc tn to ac ar or ta hh rr ap
real 0.02
user 0.01
sys 0.01
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=2 nligands=120 nthreads=4
maximal score is 2, achieved by ligands
op or ap rr tn ta no ac hh ar hc to
real 0.02
user 0.00
sys 0.03
pi@raspberrypi:~/cplusthreads1 $
```

While measuring the running time of each implementation, we have found that the argument of the program is changing the maximum ligand length, and not the number of threads. To make the number of threads as the first argument for both programs, we have edited some codes in cpp files as follows and compile them again.

dd_threads.cpp	dd_omp.cpp	dd_serial
<pre>60 61 62 63 // Main program 64 int main(int argc, char **argv) { 65 int max_ligand = DEFAULT_max_ligand; 66 int nligands = DEFAULT_nligands; 67 int nthreads = DEFAULT_nthreads; 68 string protein = DEFAULT_protein; 69 70 if (argc > 1) 71 nthreads = strtol(argv[1], NULL, 10); 72 if (argc > 2) 73 max_ligand = strtol(argv[2], NULL, 10); 74 if (argc > 3) 75 nligands = strtol(argv[3], NULL, 10); 76 if (argc > 4) 77 protein = argv[4]; 78 // command-line args parsed 79 }</pre>	<pre>65 66 67 68 // Main program 69 int main(int argc, char **argv) { 70 int max_ligand = DEFAULT_max_ligand; 71 int nligands = DEFAULT_nligands; 72 int nthreads = DEFAULT_nthreads; 73 string protein = DEFAULT_protein; 74 75 if (argc > 1) 76 nthreads = strtol(argv[1], NULL, 10); 77 if (argc > 2) 78 max_ligand = strtol(argv[2], NULL, 10); 79 if (argc > 3) 80 nligands = strtol(argv[3], NULL, 10); 81 if (argc > 4) 82 protein = argv[4]; 83 // command-line args parsed 84 }</pre>	

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 143.24
user 143.14
sys 0.01
pi@raspberrypi:~/sequential $ cd ~/openmp1
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 1
max_ligand=7 nligands=120 nthreads=1
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 144.16
user 144.11
sys 0.01
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 1
max_ligand=7 nligands=120 nthreads=1
maximal score is 5, achieved by ligands
acehch ieehkc
real 143.91
user 143.86
sys 0.01

```

Implementation	Time (s)
dd_serial	143.24
dd_omp	144.16
dd_threads	143.91

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 114.79
user 141.83
sys 0.01
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 2
max_ligand=7 nligands=120 nthreads=2
maximal score is 5, achieved by ligands
ieehkc acehch
real 78.34
user 142.06
sys 0.02
pi@raspberrypi:~/cplusthreads1 $ 

```

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 3
max_ligand=7 nligands=120 nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 98.93
user 143.43
sys 0.15
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 3
max_ligand=7 nligands=120 nthreads=3
maximal score is 5, achieved by ligands
acehch ieehkc
real 55.04
user 143.72
sys 0.03
pi@raspberrypi:~/cplusthreads1 $
```

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4
max_ligand=7 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 81.14
user 143.92
sys 0.00
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4
max_ligand=7 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 42.27
user 143.74
sys 0.07
pi@raspberrypi:~/cplusthreads1 $
```

Implementation	Time (s) 2 Threads	Time (s) 3 Threads	Time (s) 4 Threads
dd_omp	114.79	98.93	81.14
dd_threads	78.34	55.04	42.27

Discussion Questions

- 1) Which approach is the fastest?
 - C++11 threads solution is the fastest approach.
- 2) Determine the number of lines in each file. How does the C++11 implementation compare to the OpenMP implementations?

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/sequential $ wc -l dd_serial
67 dd_serial
pi@raspberrypi:~/sequential $ cd ~/openmp1
pi@raspberrypi:~/openmp1 $ wc -l dd_omp
187 dd_omp
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ wc -l dd_threads
173 dd_threads
pi@raspberrypi:~/cplusthreads1 $
```

- Sequential solution: 67 lines
- OpenMP solution: 187 lines
- C++11 Threads solution: 173 lines
- C++11 solution has fewer lines in the file than OpenMP solution.

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

```
pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 5
max_ligand=7 nligands=120 nthreads=5
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 74.06
user 143.52
sys 0.05
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 5
max_ligand=7 nligands=120 nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 44.30
user 143.95
sys 0.01
pi@raspberrypi:~/cplusthreads1 $
```

Implementation	Time (s) 5 Threads
dd_omp	74.06
dd_threads	44.30

- Since the Raspberry Pi have a 4-core processor, the maximum number of threads available is 4. They are similar to the runtimes measured when nthreads was set to 4.

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

```

pi@raspberrypi: ~/cplusthreads1
File Edit Tabs Help
pi@raspberrypi:~/openmp1 $ time -p ./dd_omp 4 11
max_ligand=11 nligands=120 nthreads=4
OMP defined

^Z
[2]+  Stopped                  ./dd_omp 4 11
real 8452.46
user 0.00
sys 0.00
pi@raspberrypi:~/openmp1 $ cd ~/cplusthreads1
pi@raspberrypi:~/cplusthreads1 $ time -p ./dd_threads 4 11
max_ligand=11 nligands=120 nthreads=4

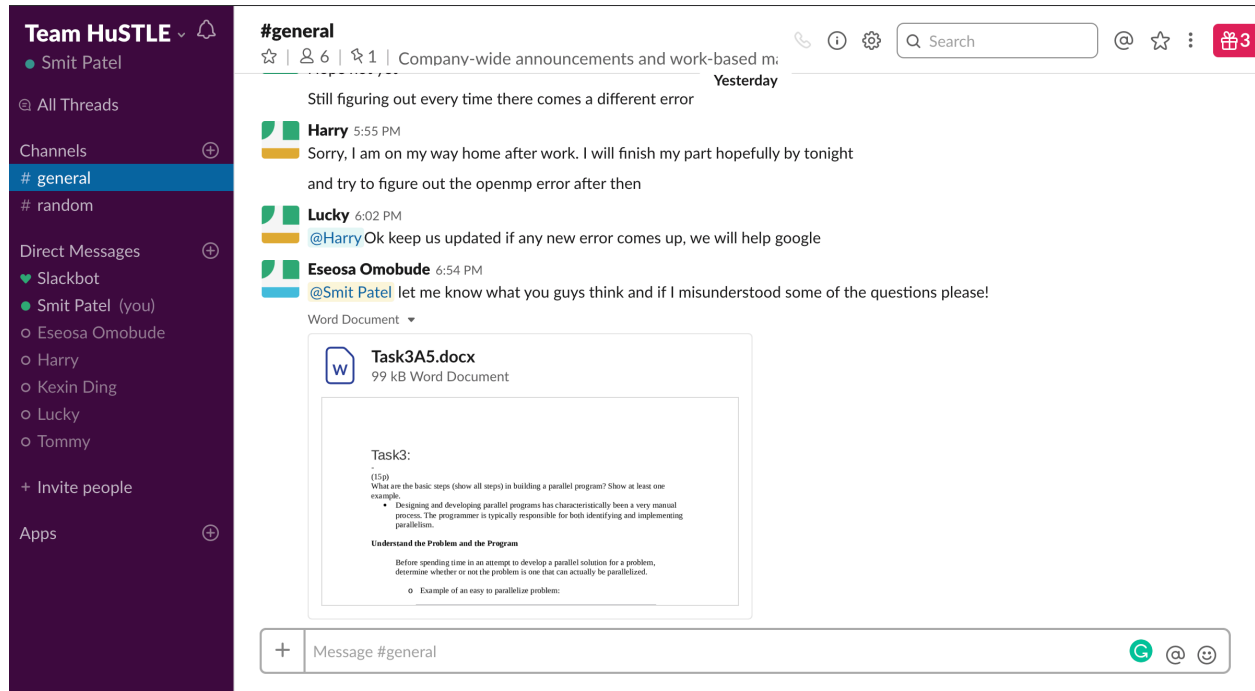
^Z
[3]+  Stopped                  ./dd_threads 4 11
real 8087.23
user 0.00
sys 0.00
pi@raspberrypi:~/cplusthreads1 $ 

```

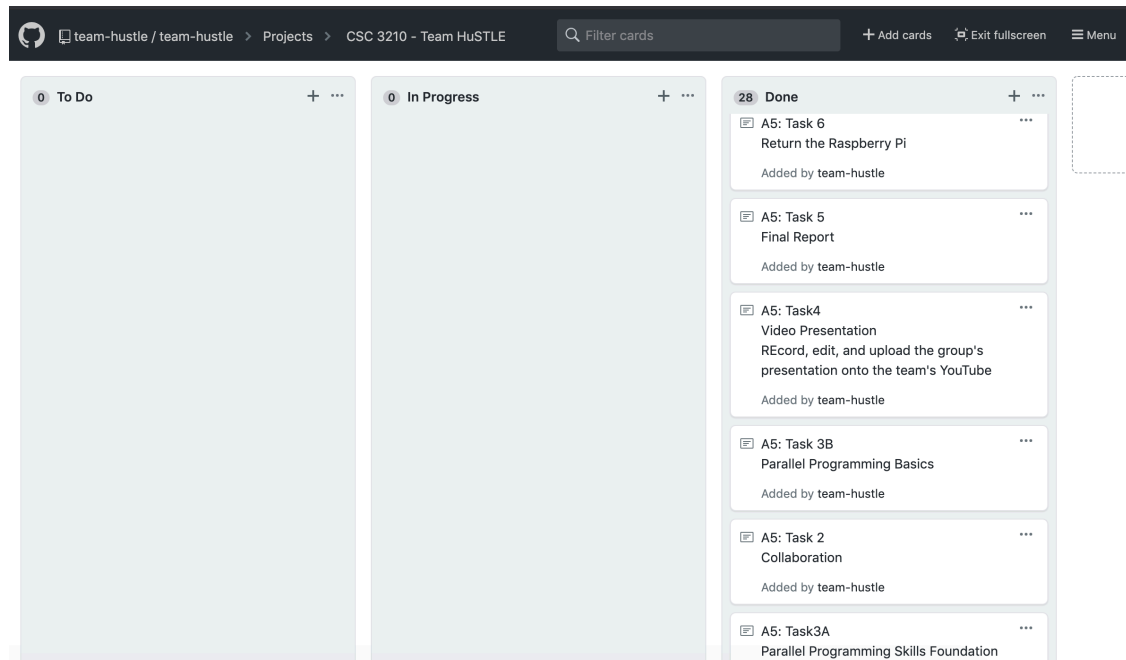
- OpenMP solution could not finish the task up until 8452.46 seconds (2.35 hours).
- C++11 solution could not finish the task up until 8087.23 seconds (2.25 hours).

Appendix

Slack: <https://teamhustle-gsu.slack.com/>



Github: <https://github.com/team-hustle/>



Youtube: <https://youtu.be/3e-o4AEKAHY>