Developing Soft and Parallel Programming Skills Using Project Based Learning

Semester: Fall 2018

Group Name: Dunder Mifflin

Members: Anli Ji, Brian Cabau, Jimmy Petit Homeis, Manuel Lipo, Riyazh Dholakia

Task 1 Planning and Scheduling:

Name's	Email	Task	Duration (hours)	Dependency	Due date	Note
Brian Cabau	bcabau1@student.gsu.edu	Team coordinator, task assignment table, final written report	3 hours	Table must be done before next meeting on 9/24	9/21/18	Includes parallel programming skills and basics
Anli Ji	aji1@student.gsu.edu	Writing the lab report, final written report	3 hours	Lab report written during group meeting	10/4/18	Document the lab report as we're all working on it
Jimmy Petit Homeis	jpetithomeis1@student.gsu.edu	Raspberry pi setup, video editing (uploading video), final written report, filming	4 hours	Pi setup must be complete before group work can begin.	9/24/18	Includes parallel programming skills and basics
Manuel Lipo	mlipo1@student.gsu.edu	Writing the lab report, final written report	3 hours	Lab report written during group meeting	10/4/18	Document the lab report as we're all working on it
Riyazh Dholakia	rdholakia1@student.gsu.edu	Github, Slack invitation, final written report	3 hours	Github must be ready by 9/24	9/24/18	Includes parallel programming skills and basics

Task 2 Parallel Programming Skills:

a) Foundation

Identifying the components on the raspberry PI B+

Ethernet/Ethernet control, Camera, Power, HDMI, USB plug, CPU/RAM, Display

How many cores does the Raspberry Pi's B+ CPU have

4 Cores

List four main differences between X86 (CISC) and ARM Raspberry PI (RISC) Justify you answer and use your own words (do not copy and past)

The four main differences are:

1. The ARM architecture use BI-endian and the X86 use Little-endian.

2. ARM has less instruction than X86 which make it faster than X86.
3. X86 allow allows many complex instructions to access memory.
4. X86 has many modes, while ARM has only two.
5. X86 has less Register than ARM.

What is the difference between sequential and parallel computation and identify the practical significance of each?

The main issue is about data distribution. A sequential module encapsulates the code that implements the functions provided by the module's interface and the data structures accessed by those functions. In parallel programming, we need to consider not only code and data but also the tasks created by a module, the way in which data structures are partitioned and mapped to processors, and internal communication structures.

Identify the basic form of data and task parallelism in computational problems.

Data parallelism is used with multiple data items, and the potential parallelism is proportional to the size of the input, so it gives a lot of flexibility to programmers. *Task parallelism* is organized around the functions rather than the data items. It deals with breaking down tasks into functional pieces. It is not as well adapted for scalability.

Explain the differences between processes and threads.

Processes can made up of multiple threads; processes are single, while threads can be many to make up a whole process. It is essentially a process broken down into different independent parts.

What is OpenMP and what is OpenMP pragmas?

OpenMP is an API which is used for multiprocessing. Pragmas tells the compiler how to process its input.

What applications benefit from multi-core (list four) ?

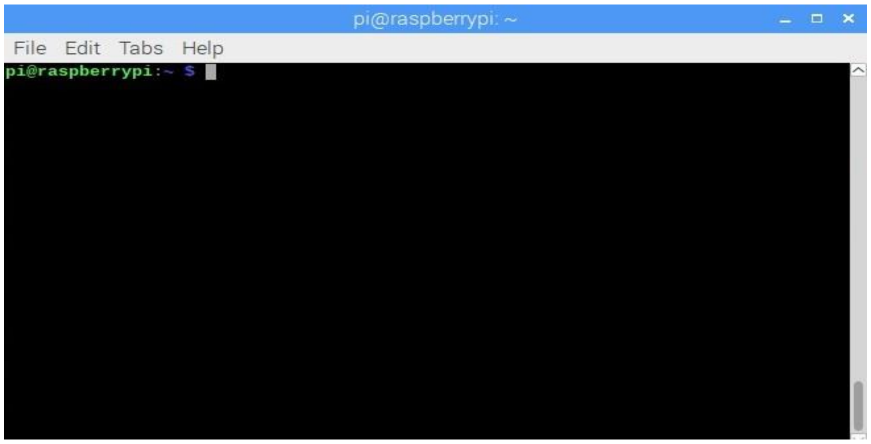
1. Database servers
2. Web servers (Web commerce)
3. Compilers
4. Multimedia applications
5. Scientific applications, CAD/CAM

Why Multicore? (why not single core, list four)

1. It’s difficult to make single core work with a high clock frequency
2. Single core has some more deep problems (heat, speed of light, difficult design and verification
3. New applications are more likely to be multithreaded
4. A general trend shifts the computer architecture to be more parallelism

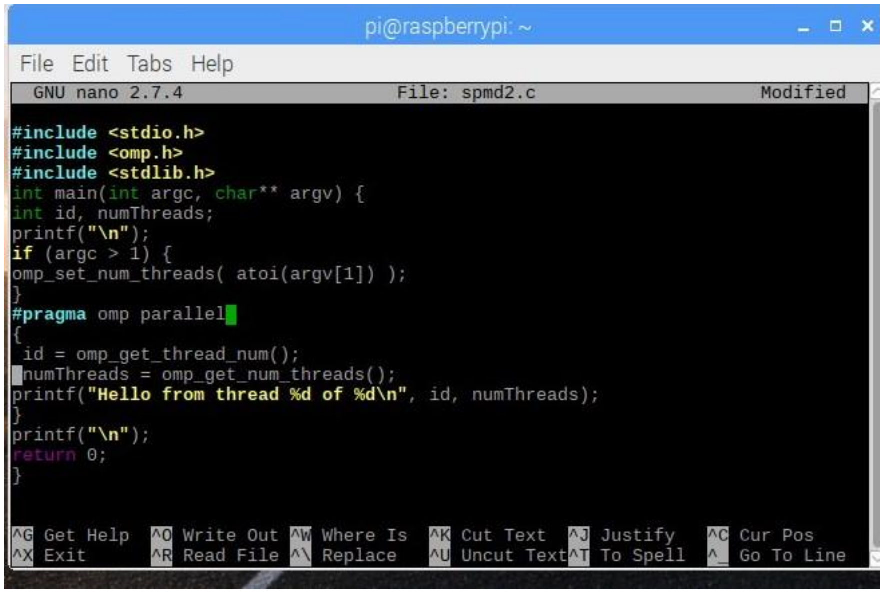
b) Pi Installation/ Parallel Programming Basics

1.1 After successfully booting up the Raspberry Pi system, the terminal window was selected. The blank terminal screen was displayed. It is interesting how such a small machine holds a close resemblance to larger machines with greater capacity for computation. A windows machine now does not seem so different from a Raspberry Pi



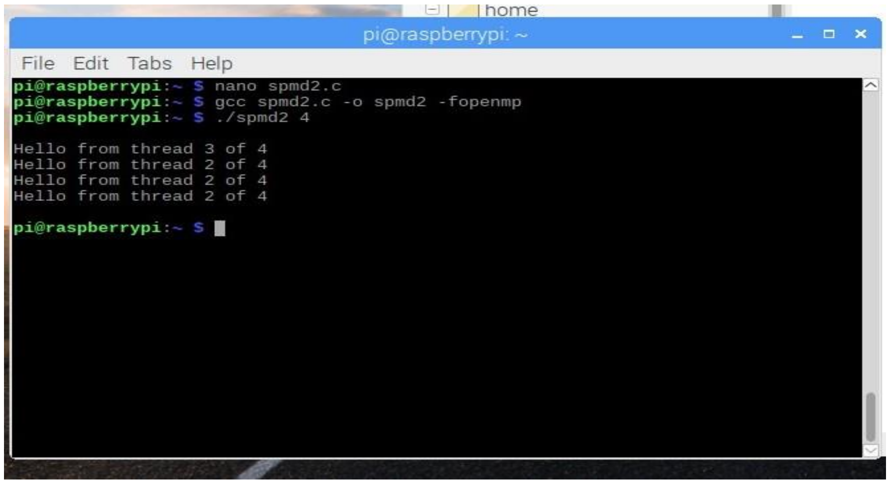
```
pi@raspberrypi:~ $
```

2.2 The first command was entered: *nano spmd.c*, which opens an editor within the terminal in order to copy the code provided. An interesting observation was how the editor does not look any different from the rest of the terminal, but colors keywords, and displays commands to modify the file.



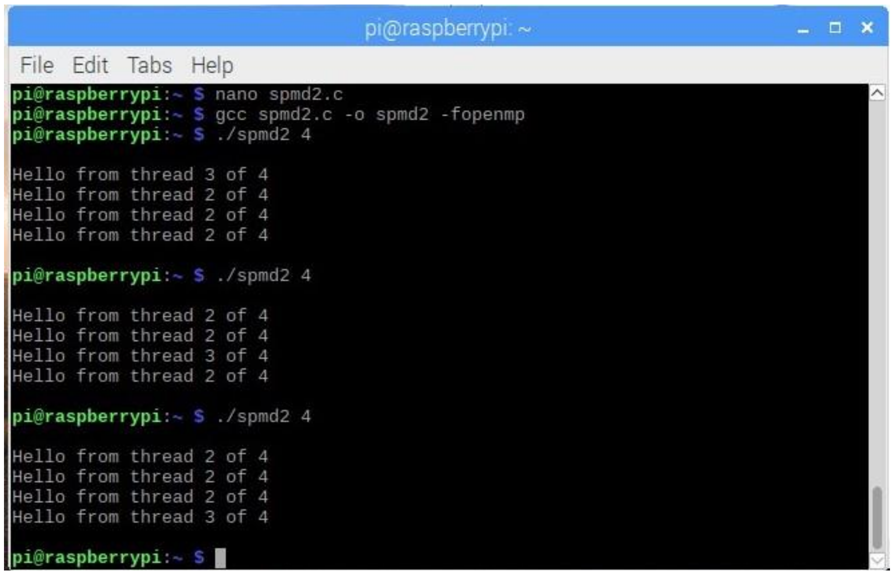
```
  GNU nano 2.7.4                    File: spmd2.c                    Modified

#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
int main(int argc, char** argv) {
int id, numThreads;
printf("\n");
if (argc > 1) {
omp_set_num_threads( atoi(argv[1]) );
}
#pragma omp parallel
{
 id = omp_get_thread_num();
 numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n", id, numThreads);
}
printf("\n");
return 0;
}

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

The file was saved, and an executable called spmd2 was created. The program was then executed. The interesting thing is how the program executed showing 4 lines of output; this shows that there are 4 cores to the CPU, and they each outputted a result.



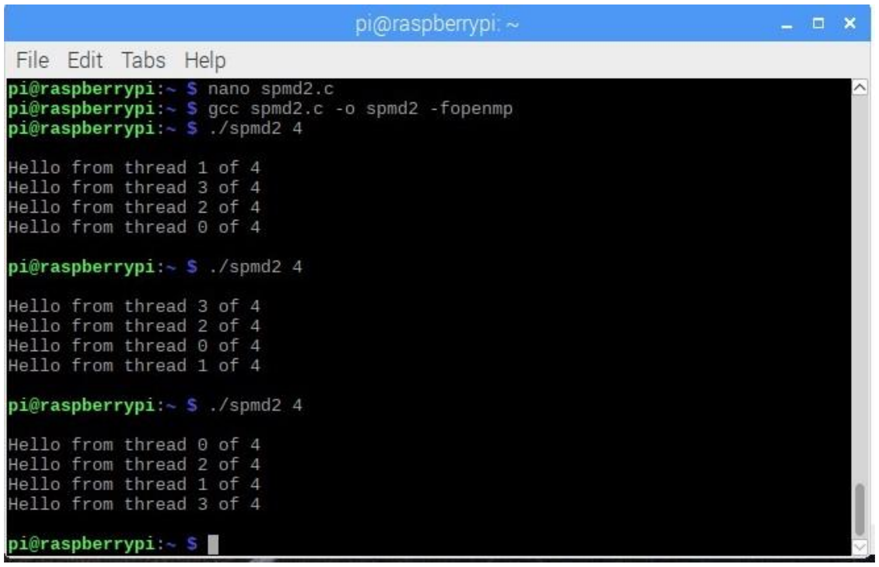
```
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4

pi@raspberrypi:~ $
```

2.3 There was a problem with the output; it should not have outputted from the same thread more than once, but typing the command *./spmd2 4* multiple times showed that the problem reoccurs. There was a problem with the code. The problem has to do with the scope of the variables. Each thread should keep its id separate from the others.



```
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4

pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4
Hello from thread 2 of 4

pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

pi@raspberrypi:~ $
```

2.4.1 The file was reopened in order to solve the problem of the recurring thread. According to the instructions two changes needed to be made: comment out line 5 and declare the variables down where they are initialized. This way each thread showed an output and meant that the problem was solved. It is interesting how such small tweaks can alter the output of a program so much. It shows all the more how important low-level programming is to prevent and solve problems.

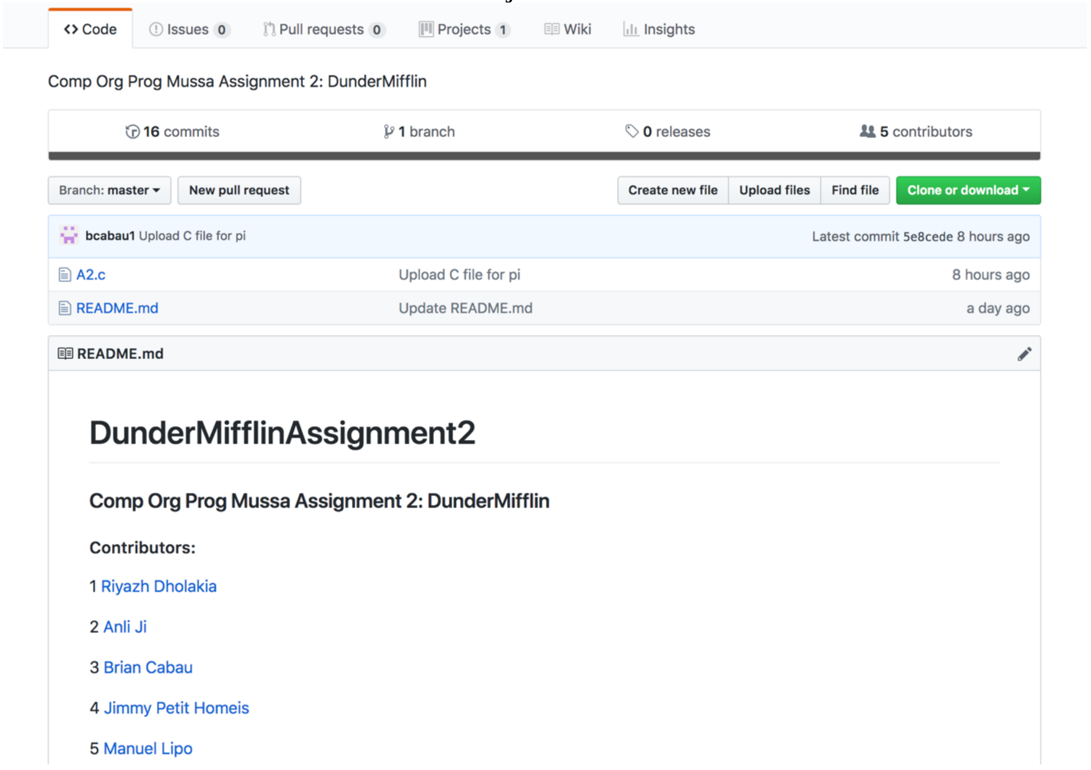


```
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 1 of 4
Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4

pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4
Hello from thread 1 of 4

pi@raspberrypi:~ $ ./spmd2 4

Hello from thread 0 of 4
Hello from thread 2 of 4
Hello from thread 1 of 4
Hello from thread 3 of 4

pi@raspberrypi:~ $ 
```

Task 3 Appendix:

Slack Invite Link:
https://join.slack.com/t/dundermifflingang/shared_invite/enQtNDM0MzAzNTc0ODUyLTQwMDJlNDMyY2I0M2Q2MjY3MzA5ZjdiM2FmOWRiZjA0ZThmOGNkNGY1YWM4NGQ4ZDBmODllMWI5NjFkZDljYWU

Github Project Screen Shot:



Comp Org Prog Mussa Assignment 2: DunderMifflin

16 commits | 1 branch | 0 releases | 5 contributors

bcabau1 Upload C file for pi — Latest commit 5e8cede 8 hours ago

A2.c	Upload C file for pi	8 hours ago
README.md	Update README.md	a day ago

README.md

DunderMifflinAssignment2

Comp Org Prog Mussa Assignment 2: DunderMifflin

Contributors:

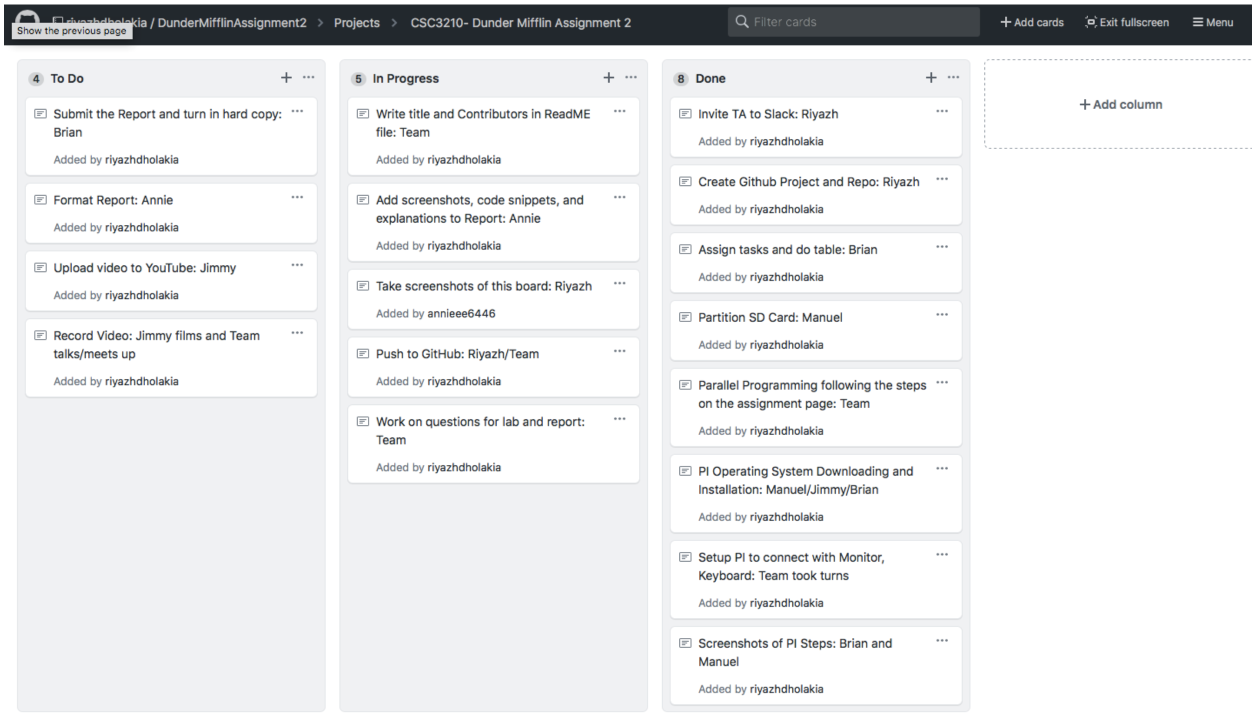
1 Riyazh Dholakia

2 Anli Ji

3 Brian Cabau

4 Jimmy Petit Homeis

5 Manuel Lipo


riyazhdholakia / DunderMifflinAssignment2 > Projects > CSC3210- Dunder Mifflin Assignment 2
Show the previous page
Filter cards
+ Add cards
Exit fullscreen
Menu
+ Add column

4 To Do

- Submit the Report and turn in hard copy: Brian — Added by riyazhdholakia
- Format Report: Annie — Added by riyazhdholakia
- Upload video to YouTube: Jimmy — Added by riyazhdholakia
- Record Video: Jimmy films and Team talks/meets up — Added by riyazhdholakia

5 In Progress

- Write title and Contributors in ReadME file: Team — Added by riyazhdholakia
- Add screenshots, code snippets, and explanations to Report: Annie — Added by riyazhdholakia
- Take screenshots of this board: Riyazh — Added by annieee6446
- Push to GitHub: Riyazh/Team — Added by riyazhdholakia
- Work on questions for lab and report: Team — Added by riyazhdholakia

8 Done

- Invite TA to Slack: Riyazh — Added by riyazhdholakia
- Create Github Project and Repo: Riyazh — Added by riyazhdholakia
- Assign tasks and do table: Brian — Added by riyazhdholakia
- Partition SD Card: Manuel — Added by riyazhdholakia
- Parallel Programming following the steps on the assignment page: Team — Added by riyazhdholakia
- PI Operating System Downloading and Installation: Manuel/Jimmy/Brian — Added by riyazhdholakia
- Setup PI to connect with Monitor, Keyboard: Team took turns — Added by riyazhdholakia
- Screenshots of PI Steps: Brian and Manuel — Added by riyazhdholakia

YouTube link:

https://www.youtube.com/channel/UCbAorx7CYVlyDs7_38edPRg