

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

Project Report

Fall-2018

Submitted By PAJ\_VR

Prashant Vemulapalli, Anthony Davis, Jaiana Butler, Vincent Lee, Ryan Barrett

## Planning and Scheduling

Name:	Email:	Task(s):	Duration :	Dependency:	Due Date:	Note:
Prashant Vemulapalli	pvemulapalli1@student.gsu.edu	Installing the raspberry pi, Collaborating on Parallel Programming skills questions, Collaborating on Lab report	6 hours	Raspberry Pi	9/30/2018	
Ryan Barret	rbarrett7@student.gsu.edu	Creating the Github, Collaborating on Parallel Programming skills questions, Collaborating on Lab report	6 hours	Raspberry Pi	9/30/2018	
Anthony Davis	adavis183@student.gsu.edu	Collaborating on Lab report, Installing the raspberry Pi, Collaborating on Parallel Programming Skills questions, Sending Slack invitation	6 hours	Slack, Raspberry Pi	9/30/2018	
Jaiana Butler	jbutler38@student.gsu.edu	Video editing, Collaborating on Parallel Programming Skills questions, Collaborating on Lab report	6 hours	YouTube channel, Raspberry Pi	10/3/2018	
Vincent Lee	vlee18@student.gsu.edu	Organizing the main report, Collaborating on Parallel Programming Skills questions, Collaborating on Lab report, Creating the task table	6 hours	Parallel Programming Skills questions, Lab report, Github, YouTube Channel, Raspberry Pi	9/30/2018	

## Parallel Programming Skills

*Identifying the components on the raspberry PI B+*

The raspberry pi has nine components: Display, power, ethernet controller, ethernet port, camera, CPU/RAM, HDMI, two USB ports, and a 3.5 audio jack.

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

The Raspberry Pi B+ has four cores.

*List four main differences between X86 (CISC) and ARM Raspberry PI (RISC).*

The x86 has an instruction set that is composed of more complex operations, and may take more cycles to execute, while the ARM Raspberry pi uses an instruction set composed of simpler instructions that reduce the amount of cycles needed to execute them.

ARM uses a load/store memory model in which arithmetic operations such as add, or sub are executed exclusively in registers while the x86 uses the register/memory model that doesn't have these restrictions.

The x86 uses little endian while the ARM is bi-endian, meaning it can use either little endian or big endian, depending on the situation.

Because the ARM uses RISC, it can also utilize pipelining more effectively than x86, due to it being able to execute most instructions in one clock cycle, so it can use a pipeline composed of single instructions segments to complete a task.

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

In parallel computation, processes/threads are executed simultaneously, while in sequential computation, processes are executed one by one. The practical significance of parallel computation is that tasks can be executed in a shorter amount of time, and the practical significance of sequential computation is that it may have a lower cost compared to parallel computation.

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

Examples of data and task parallelism are application creating threads for doing parallel processing where each thread is responsible for performing a different operation, or a client server assigning some tasks the job of making requests and others the jobs of servicing requests.

*Explain the differences between processes and threads.*

A process is an abstraction of a program that does not share memory with other processes. A process can be decomposed into smaller individual process called threads, and these threads can share memory with other threads.

*What is OpenMP and what is OpenMP pragmas?*

OpenMP is an API to write multi-threaded processes in parallel processing. OpenMP pragmas are compiler directives that allow for the generation of threaded code.

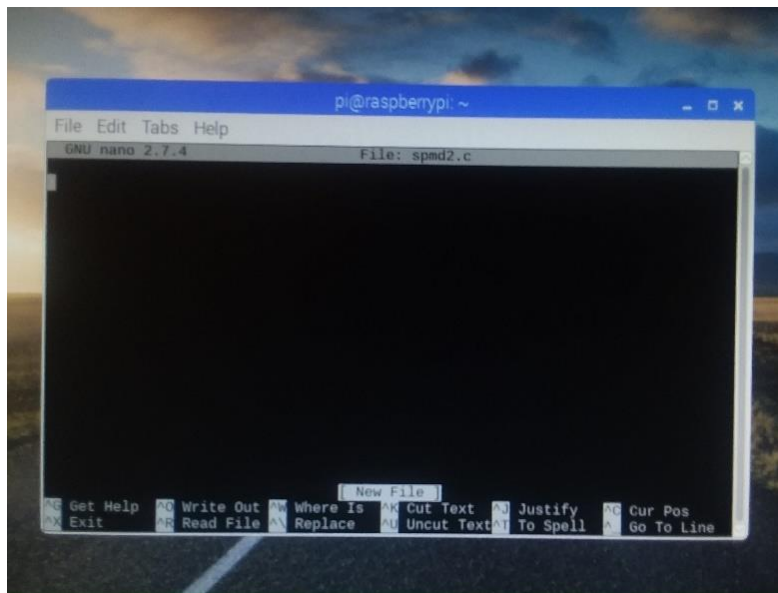
*What applications benefit from multi-core (list four)?*

Database servers, web servers, compilers, and multimedia applications.

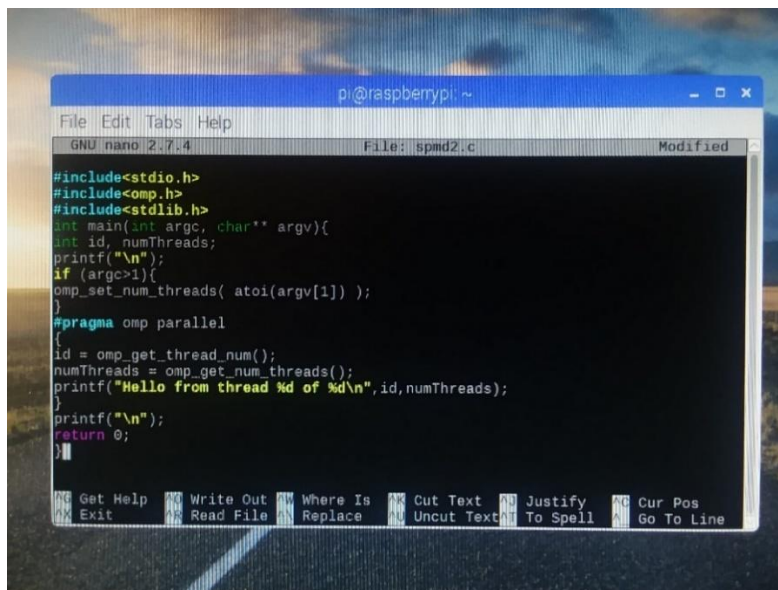
*Why Multicore? (why not single core, list four)*

Heat and speed issues with single core, higher clock frequency in multicore compared to single-core, many newer applications use multiple cores, multi core allowing for multithreading, and a general trend towards multicore in recent times.

## Parallel Programming Basics

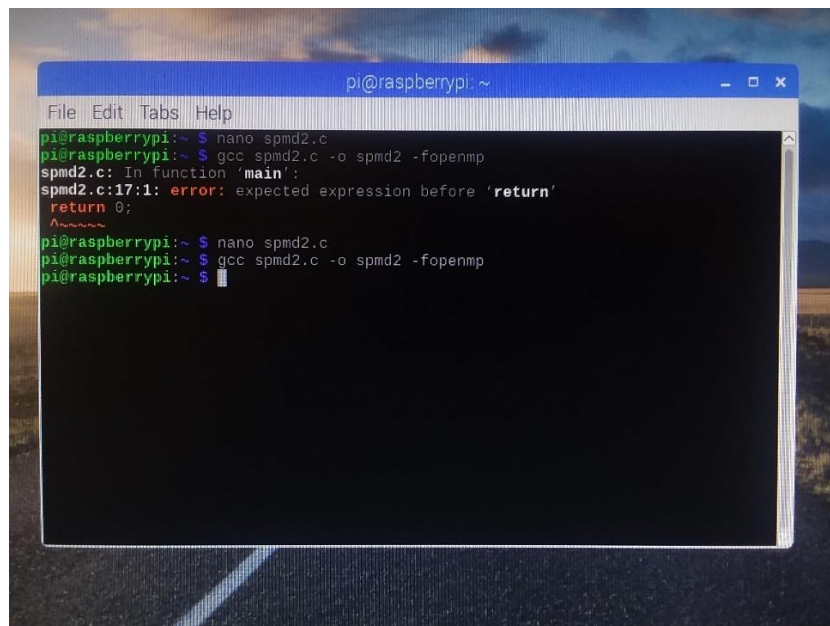


Explanation: The code “**nano spmd2.c**” was used to open the nano editor and begin writing the spmd2.c code.



Explanation: The spmd2.c code is then written in the nano editor.

Observation(s): The code appears to be including some sort of libraries, similar to how importing libraries are used in java. The “**id**” and “**numThreads**” also seem to be returning what thread number and id the printf in line 14 will print.

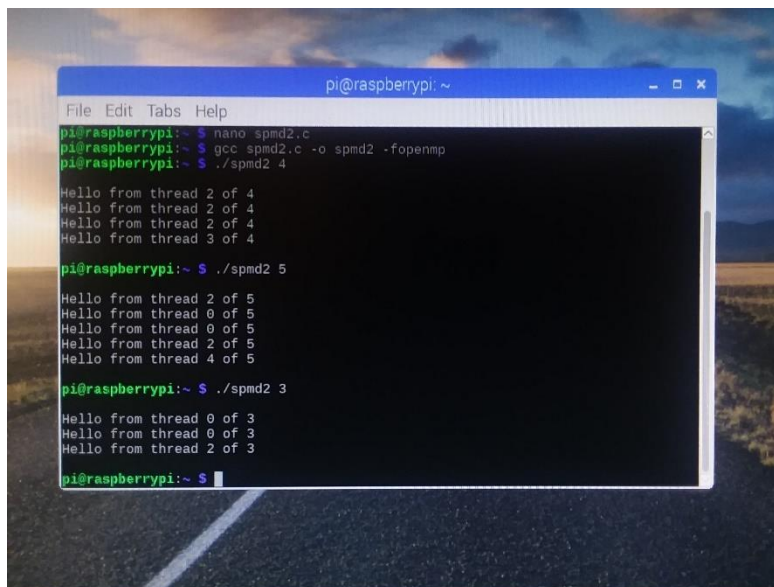


```

pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
spmd2.c: In function 'main':
spmd2.c:17:1: error: expected expression before 'return'
  return 0;
  ~~~~~
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $

```

Explanation: The executable for the spmd2.c program is created using the code “**gcc spmd2.c -o -fopenmp**”.



```

pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
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:~ $ ./spmd2 5

Hello from thread 2 of 5
Hello from thread 0 of 5
Hello from thread 0 of 5
Hello from thread 2 of 5
Hello from thread 4 of 5

pi@raspberrypi:~ $ ./spmd2 3

Hello from thread 0 of 3
Hello from thread 0 of 3
Hello from thread 2 of 3

pi@raspberrypi:~ $

```

Explanation: Multiple values in place of 4 are tried for the spmd2.c program, and there are repeating thread ids in the result.

Observation(s): After more testing, it appears that the threads with an id of 0 occur whenever the number of threads to fork is less than or greater than 4, while a thread with an id of 0 will never occur if the number of threads to fork is 4.

```

pi@raspberrypi: ~
File Edit Tabs Help
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
{
int id = omp_get_thread_num();
int numThreads = omp_get_num_threads();
printf("Hello from thread %d of %d\n",id,numThreads);
}
printf("\n");
return 0;
}
Get Help Write Out Where Is Cut Text Justify Cur Pos
Exit Read File Replace Uncut Text To Spell Go To Line

```

Explanation: The incorrect code is corrected by using “**nano spmd2.c**” then commenting out line 5 by adding “//” then placing “**int**” at the beginning of lines 12 and 13.

```

pi@raspberrypi: ~
File Edit Tabs Help

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:~ $ ./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

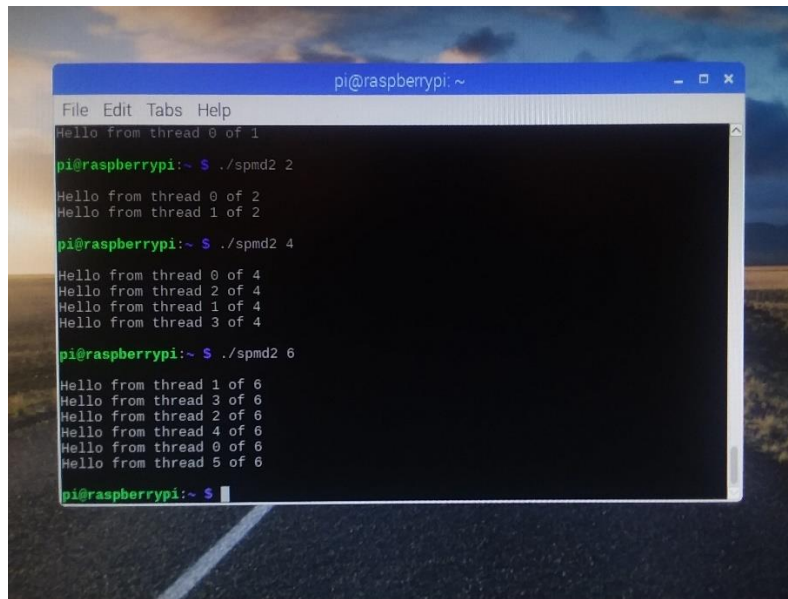
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ nano spmd2.c
pi@raspberrypi:~ $ gcc spmd2.c -o spmd2 -fopenmp
pi@raspberrypi:~ $ ./spmd2 4

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

pi@raspberrypi:~ $

```

Explanation: After saving the changes in nano and exiting, “**gcc spmd2.c -o -fopenmp**” is used again to compile the corrected code, and “**./spmd2 4**” is used to execute the code, and this time there are no repeated thread ids in the result. The code is successful.

A terminal window titled 'pi@raspberrypi: ~' with a menu bar (File, Edit, Tabs, Help) and standard window controls. The terminal shows the output of a program for three different thread counts: 1, 2, and 4. For 1 thread, it prints 'Hello from thread 0 of 1'. For 2 threads, it prints 'Hello from thread 0 of 2' and 'Hello from thread 1 of 2'. For 4 threads, it prints 'Hello from thread 0 of 4', 'Hello from thread 2 of 4', 'Hello from thread 1 of 4', and 'Hello from thread 3 of 4'. For 6 threads, it prints 'Hello from thread 1 of 6', 'Hello from thread 3 of 6', 'Hello from thread 2 of 6', 'Hello from thread 4 of 6', 'Hello from thread 0 of 6', and 'Hello from thread 5 of 6'. The prompt 'pi@raspberrypi:~ \$' is visible at the end of each command line.

```
pi@raspberrypi: ~
File Edit Tabs Help
Hello from thread 0 of 1

pi@raspberrypi:~ $ ./spmd2 2
Hello from thread 0 of 2
Hello from thread 1 of 2

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:~ $ ./spmd2 6
Hello from thread 1 of 6
Hello from thread 3 of 6
Hello from thread 2 of 6
Hello from thread 4 of 6
Hello from thread 0 of 6
Hello from thread 5 of 6

pi@raspberrypi:~ $
```

Explanation: The code is then tested again for other values, and there are also no repeating thread ids for numbers of threads greater than or less than 4.

Observation(s): Now that the code is correct, a thread with id 0 will appear for any number of threads, including 4.

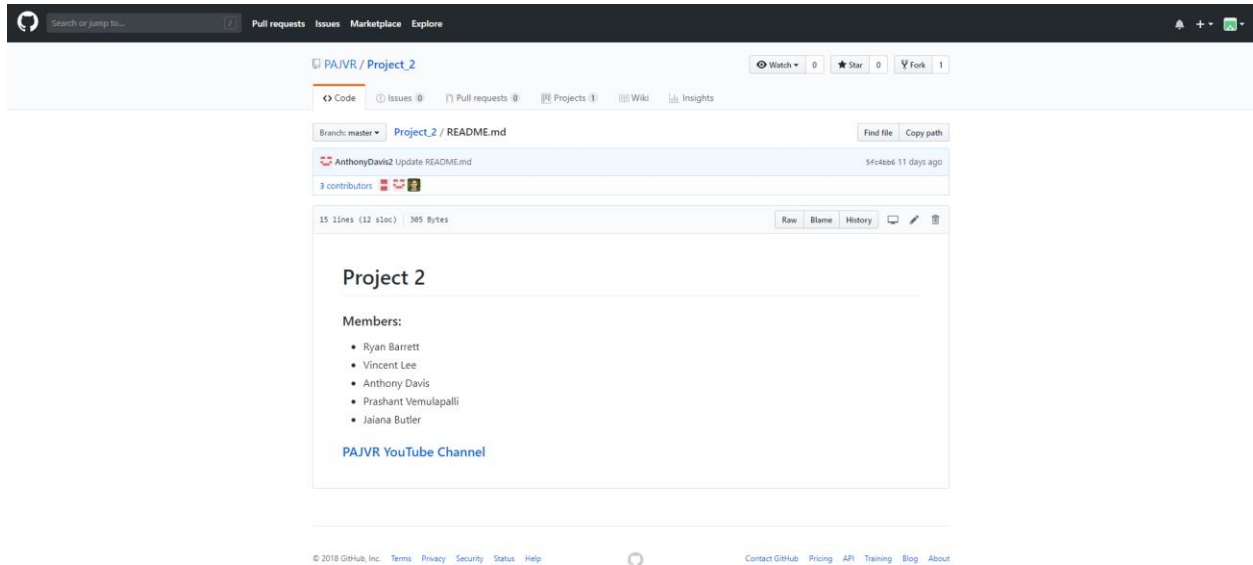
## Appendix

Video: <https://youtu.be/drT0E6N0-2M>

Slack: [pajvr.slack.com](https://pajvr.slack.com)

Github: [https://github.com/PAJVR/Project\\_2](https://github.com/PAJVR/Project_2)

Github readme page screenshot:



Github project page screenshot:

