

Name: Minh Binh Nguyen
PantherID: 002-46-4288

Project 2 – Task 3

a. Foundation:

1. Identifying the components on the Raspberry PI B+:
 - Single Board Computer Quad-Core Multicore CPU
 - 1 GB of RAM
 - I/O ports: display port, 2 USB ports, Ethernet port, HDMI port, camera port.
 - Ethernet Controller
 - Power port and power button
2. The Raspberry Pi's B+ CPU has 4 cores.
3. The differences between X86 (CISC) and ARM Raspberry PI (RISC):
 - Instruction Set: The CISC has larger instruction set, more features. The ARM has 100 or less instruction sets, they are simplified instruction sets.
 - Registers: CISC has less registers than ARM. ARM has more general-purpose registers than CISC.
 - Memory Access: ARM uses instructions that only operate only on registers using a Load/Store memory model for memory access. CISC allows memory access to many complex instructions.

4. The difference between Sequential computation and parallel computation is:

Sequential Computation	Parallel Computation
<ul style="list-style-type: none">- A problem is broken down into a discrete series of instructions which will be executed sequentially one after another.- The instructions are executed only on a single processor.- Only one instruction is only executed at 1 time.	<ul style="list-style-type: none">- A problem is broken down into discrete parts that can be solved concurrently. These parts will be broken down to a series of instructions.- The instructions are simultaneously executed on different processors.- Use control/coordination mechanism.

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

- Data Parallelism is a broad category of parallelism. The amount of available parallelism is proportional to the input size, which leads to big amounts of potential parallelism, because the same computation is applied to multiple data items.
- Task Parallelism is the broad classification of task parallelism which parallelism is organized around the functions to be performed instead of around the data input size.

6. The differences between processes and threads

Processes	Threads
<ul style="list-style-type: none">- Process is the abstraction of a running program.- Processes don't share memory with each other.- A single-core CPU only executes 1 process at a time.	<ul style="list-style-type: none">- Thread is a lightweight process which allows a single executable/process to be broken down to smaller, independent parts.- All threads share the common memory of the process which they belong to.- OS will schedule threads on separate cores/CPU's, as available.

7. What is OpenMP and what are OpenMP pragmas?

OpenMP is a standard that compilers who implement it must adhere to. Open MP uses an implicit multithreading model in which the library controls thread creation and management. This can benefit the programmers by making their tasks much simpler and less error prone.

OpenMP pragmas are compiler directives which allow compiler to generate threaded code.

8. What applications benefit from multi-core?

There are applications that benefit from multi-cores, such as:

- Compilers
- Multimedia Applications
- Scientific applications (CAD/CAM)
- Web servers

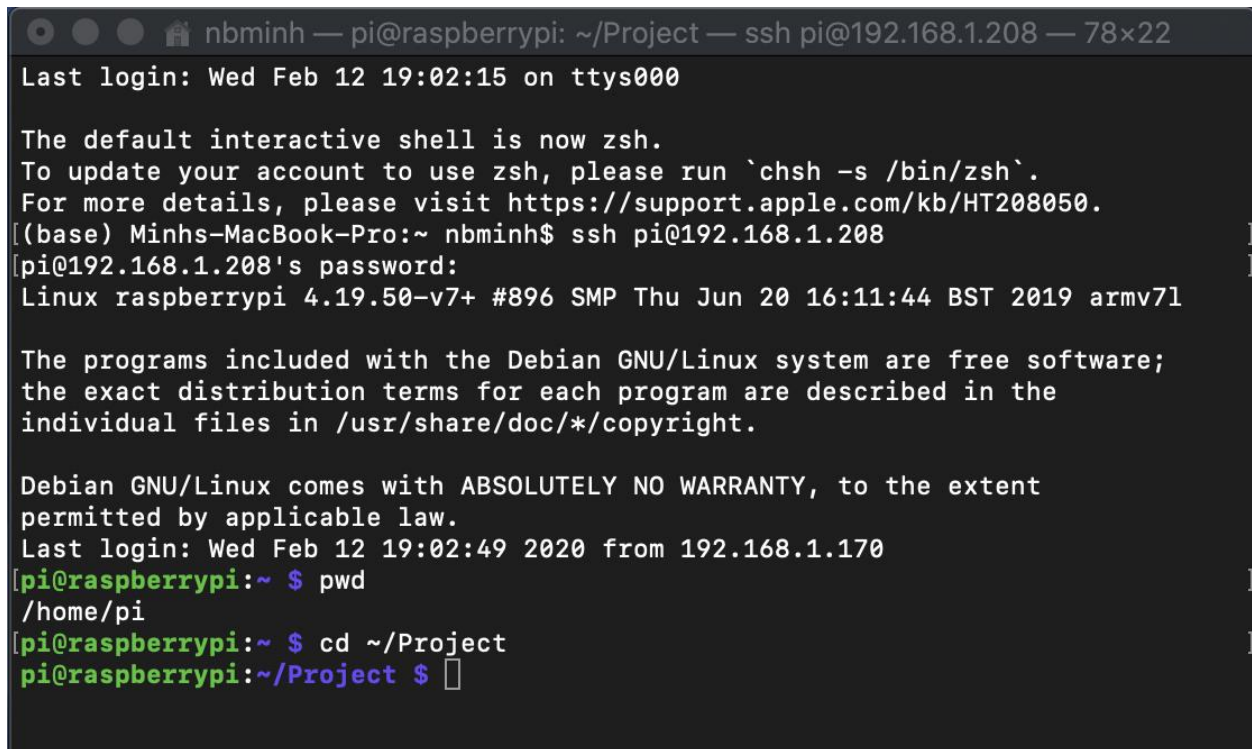
9. Why multicore?

There are reasons to apply multicore in the real life:

- It's more difficult to produce single-core clock frequencies even higher.
- Deeply pipelined circuits. It solves many problems such as heat problems, speed of light problems.
- As the quick growth of software, there are many new applications that are built multithreaded.
- This is a general trend when designing computer architecture.

b. Parallel Programming

Firstly, I started with connecting my Raspberry to my computer using IP address. Then I redirect my working directory to my “Project” folder where I save all the project files for this class (Figure 1).

A terminal window showing an SSH session from a Mac to a Raspberry Pi. The window title is 'nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x22'. The output shows the last login time, a message about switching to zsh, the SSH command used, the password prompt, the system version (Linux raspberrypi 4.19.50-v7+), Debian GNU/Linux version information, and the user's current directory being /home/pi. Finally, the user runs 'cd ~/Project' and the prompt changes to ~/Project.

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x22
Last login: Wed Feb 12 19:02:15 on ttys000

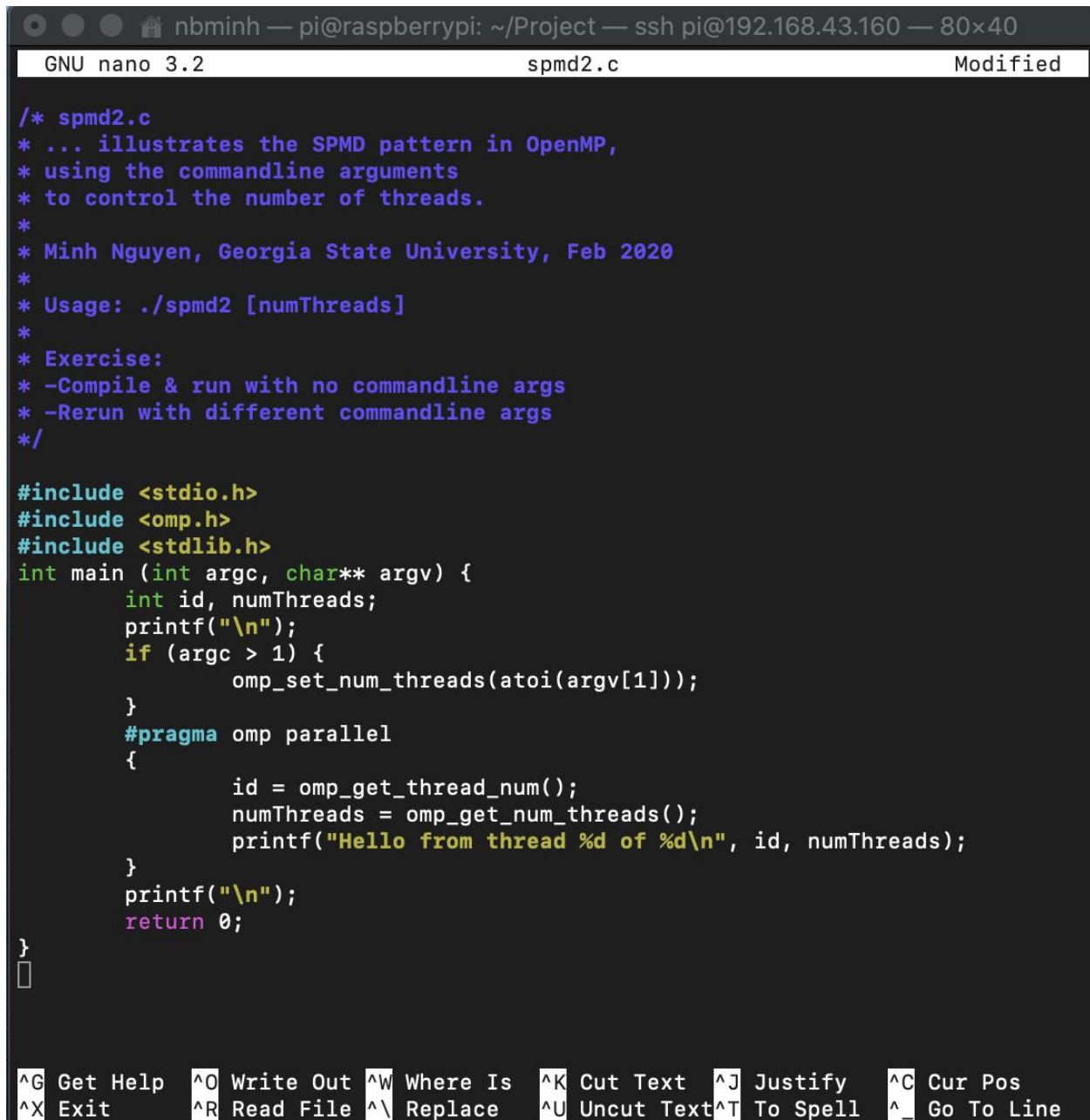
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) Minhs-MacBook-Pro:~ nbminh$ ssh pi@192.168.1.208
pi@192.168.1.208's password:
Linux raspberrypi 4.19.50-v7+ #896 SMP Thu Jun 20 16:11:44 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Feb 12 19:02:49 2020 from 192.168.1.170
[pi@raspberrypi:~ $ pwd
/home/pi
[pi@raspberrypi:~ $ cd ~/Project
pi@raspberrypi:~/Project $
```

Figure 1

Then I followed the instruction, created file called `spmd2.c`, then typed the given code. I learned that the line “`#pragma omp parallel`” makes all the statements inside the curly brackets run concurrently on a given number of threads (Figure 2).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x40
GNU nano 3.2                                spmd2.c                                Modified

/* spmd2.c
 * ... illustrates the SPMD pattern in OpenMP,
 * using the commandline arguments
 * to control the number of threads.
 *
 * Minh Nguyen, Georgia State University, Feb 2020
 *
 * Usage: ./spmd2 [numThreads]
 *
 * Exercise:
 * -Compile & run with no commandline args
 * -Rerun with different commandline args
 */

#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
```

Figure 2

Then I make executable program, then run it forking 1, 2, 3, 4 threads using commands (Figure 3), and 5, 6 threads (Figure 4).

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x23
[pi@raspberrypi:~/Project $ ./spmd2 1

Hello from thread 0 of 1

[pi@raspberrypi:~/Project $ ./spmd2 2

Hello from thread 0 of 2
Hello from thread 1 of 2

[pi@raspberrypi:~/Project $ ./spmd2 3

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

[pi@raspberrypi:~/Project $ ./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:~/Project $
```

Figure 3

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x18
[pi@raspberrypi:~/Project $ ./spmd2 5

Hello from thread 2 of 5
Hello from thread 3 of 5
Hello from thread 4 of 5
Hello from thread 0 of 5
Hello from thread 1 of 5

[pi@raspberrypi:~/Project $ ./spmd2 6

Hello from thread 3 of 6
Hello from thread 3 of 6
Hello from thread 0 of 6
Hello from thread 0 of 6
Hello from thread 5 of 6
Hello from thread 4 of 6

pi@raspberrypi:~/Project $
```

Figure 4

Notice that the number of threads is correct, but the IDs of each thread are repeated. This happened because Raspberry has multiple cores, but they all share a bank of memory. The id variable wasn't created inside the parallel block. To fix this, the id and numThreads need to be declared inside the parallel block so that each thread can have its own unique ID. Here is the fixed code (Figure 5).



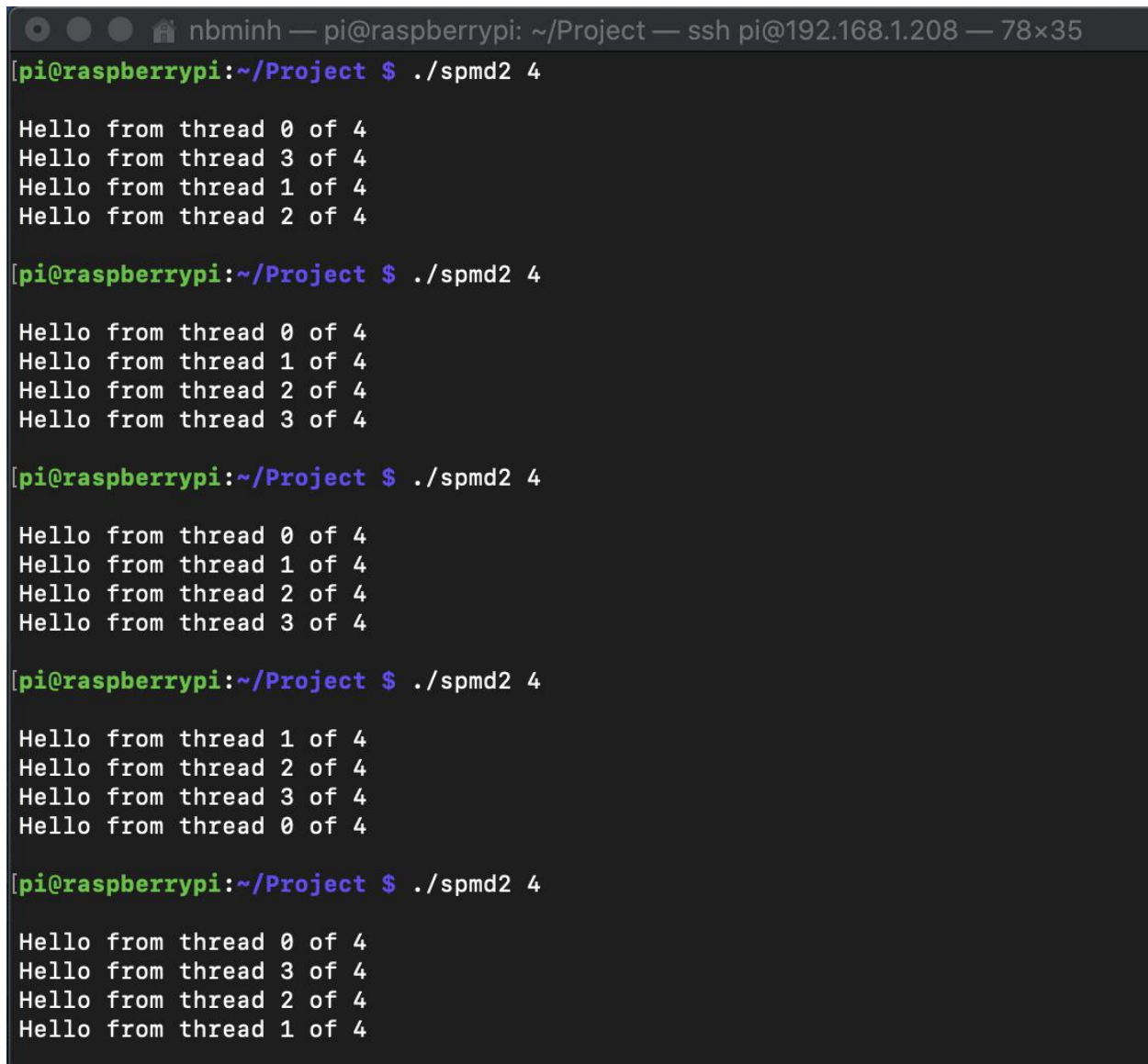
```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.43.160 — 80x40
GNU nano 3.2                                spmd2.c                                Modified

/* spmd2.c
 * ... illustrates the SPMD pattern in OpenMP,
 * using the commandline arguments
 * to control the number of threads.
 *
 * Minh Nguyen, Georgia State University, Feb 2020
 *
 * Usage: ./spmd2 [numThreads]
 *
 * Exercise:
 * -Compile & run with no commandline args
 * -Rerun with different commandline args
 */

#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;
}
^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
```

Figure 5

After I fixed it, the program runs as it expects. Each thread now has its own ID (Figure 6 and figure 7).



```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x35
[pi@raspberrypi:~/Project $ ./spmd2 4
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4

[pi@raspberrypi:~/Project $ ./spmd2 4
Hello from thread 0 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

[pi@raspberrypi:~/Project $ ./spmd2 4
Hello from thread 0 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

[pi@raspberrypi:~/Project $ ./spmd2 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4
Hello from thread 0 of 4

[pi@raspberrypi:~/Project $ ./spmd2 4
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 2 of 4
Hello from thread 1 of 4
```

Figure 6

```
nbminh — pi@raspberrypi: ~/Project — ssh pi@192.168.1.208 — 78x47
[pi@raspberrypi:~/Project $ ./spmd2 5

Hello from thread 2 of 5
Hello from thread 1 of 5
Hello from thread 3 of 5
Hello from thread 0 of 5
Hello from thread 4 of 5

[pi@raspberrypi:~/Project $ ./spmd2 6

Hello from thread 3 of 6
Hello from thread 5 of 6
Hello from thread 1 of 6
Hello from thread 4 of 6
Hello from thread 0 of 6
Hello from thread 2 of 6

[pi@raspberrypi:~/Project $ ./spmd2 7

Hello from thread 1 of 7
Hello from thread 4 of 7
Hello from thread 0 of 7
Hello from thread 2 of 7
Hello from thread 3 of 7
Hello from thread 5 of 7
Hello from thread 6 of 7

[pi@raspberrypi:~/Project $ ./spmd2 8

Hello from thread 3 of 8
Hello from thread 6 of 8
Hello from thread 7 of 8
Hello from thread 4 of 8
Hello from thread 0 of 8
Hello from thread 2 of 8
Hello from thread 5 of 8
Hello from thread 1 of 8

[pi@raspberrypi:~/Project $ ./spmd2 5

Hello from thread 1 of 5
Hello from thread 3 of 5
Hello from thread 2 of 5
Hello from thread 4 of 5
Hello from thread 0 of 5

pi@raspberrypi:~/Project $
```

Figure 7