Praveen Doluweera

Project 3 Task 3

**Part A:  Foundation**

**(5p) Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization. (in your own words)**

Task- This is a set of instructions that is executed by the processor. In parallel programming, there are usually multiple tasks that are run by multiple processors.

Pipelining- An operation that breaks down a task into steps that are performed by different processor units

Shared Memory- this is a common physical memory that is used by all the processors. In this model, the parallel tasks directly address and access logical memory locations regardless of where the physical memory exists.

Communications – This is a data exchange by the parallel tasks by way of memory bus or network

Synchronization – This is the coordination of parallel tasks in real time. The link together to commit a certain sequence or action.

**(8p) Classify parallel computers based on Flynn’s taxonomy. Briefly describe every one of them.**

1. SIMD (Single instruction and multiple data stream)
   1. The same instruction will be executed by all units at any clock cycle.
   2. Operates on a different data element
2. MISD (Multiple instruction stream and single data stream)
   1. Processing units work on data independently
   2. A single data stream is fed into multiple processing units.
3. MIMD (Multiple Instruction, Multiple Data)
   1. Every processor may be working with a different data stream
   2. Execution can be synchronous or asynchronous

**(7p) What are the Parallel Programming Models?**

* Shared memory (without threads)
* Threads
* Distributed Memory / Message Passing
* Data Parallel
* Hybrid
* Single Program Multiple Data (SPMD)
* Multiple Program Multiple Data (MPMD)

**(12p) List and briefly describe the types of Parallel Computer Memory Architecture. What type is used by OpenMP and why?**

Uniform Memory Access (UMA)

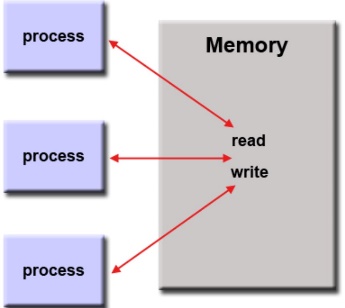
* This is also known as CC-UMA (Cache Coherent UMA)
* Identical processors which have equal and time access to the memory

Non-Uniform Memory Access (NUMA)

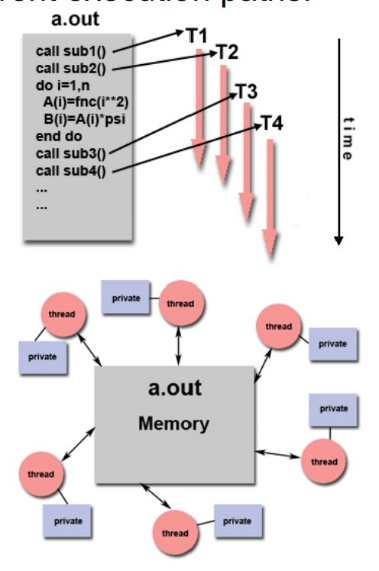
* Often made by linking two or more SMPs which share memory to each other. If cache coherency is maintained, then may also be called CC-NUMA -Cache Coherent NUMA
* NUMA is used by OpenMP. Because it is an API that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran, on most platforms including Solaris, AIX, HP-UX, Linux, macOS and Windows.

**(10p) Compare Shared Memory Model with Threads Model? (in your own words and show pictures)**

Shared Memory model (without threads) processes share a common address space that can read and write to an asynchronous way. An advantage of this is that it is easy to write an application. All processes have equal access to the shared memory. A disadvantage is it is hard to manage and understand data.



The next type of model is the threading model. It is a type of shared memory programming which has a single “heavyweight” process which can have multiple concurrent execution paths. An advantage of this is that is more suitable for applications based on the multiple data. A disadvantage of this is that it is harder to write applications using this model because it usually has complex instructions.



**(5p) What is Parallel Programming? (in your own words)**

Parallel Programming is using multiple processing elements to solve problems simultaneously

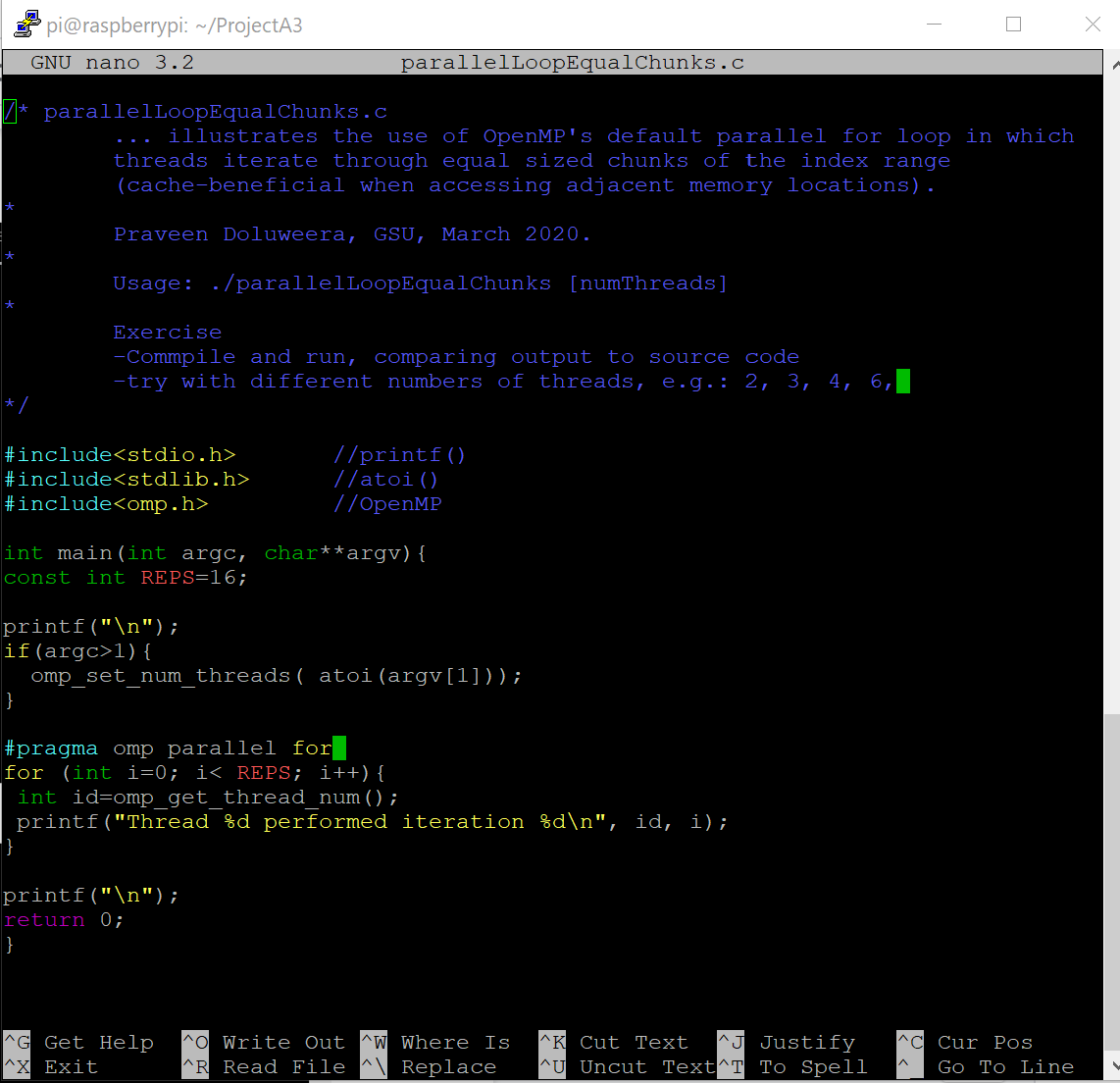
**(5p) What is system on chip (SoC)? Does Raspberry PI use system on SoC?**

A SoC is a system on chip. This integrates components including the CPU, GPU and a memory on a single chip. The Raspberry Pi is an example of such a chip.

**(5p) Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.**

The SoC is smaller which allows computers and other devices to become smaller and smaller. It also uses less power and is cheaper to produce and assemble.

**Part B: Parallel Programming Basics**

****

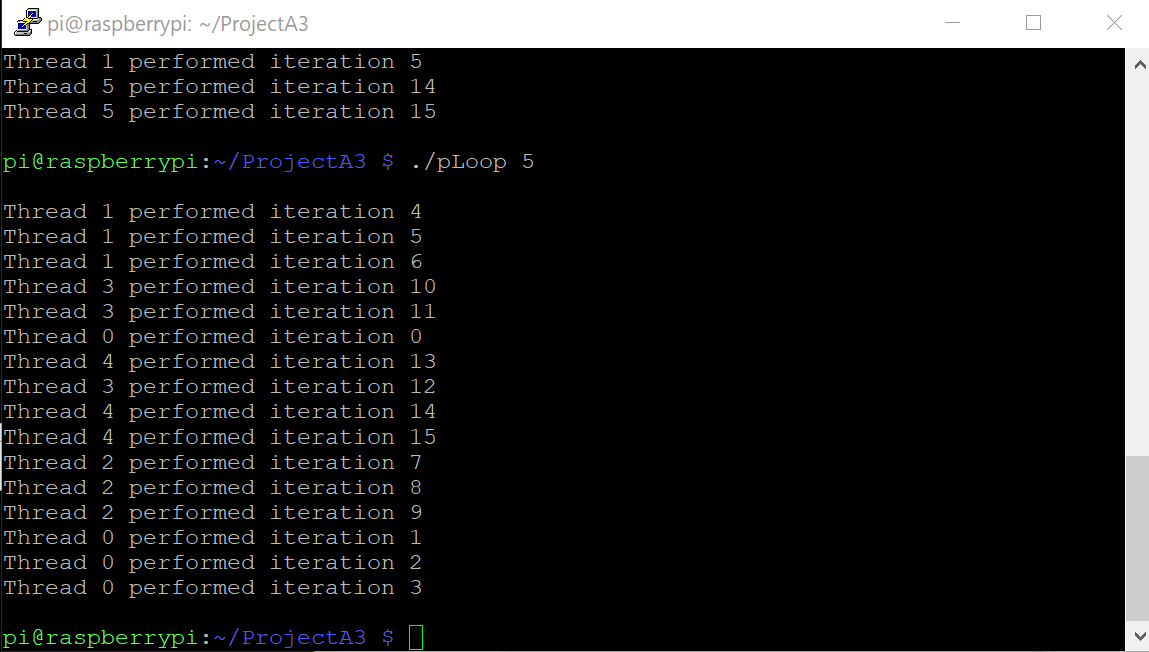
*Figure 1 – the contents of parallelLoopEqualChunks.c*

I first used PUTTY to ssh into my raspberry pi from my laptop and typed out the following code for parallelLoopEqualChunks.c (Figure 1). I then compiled the code and ran it as shown in Figure 2.

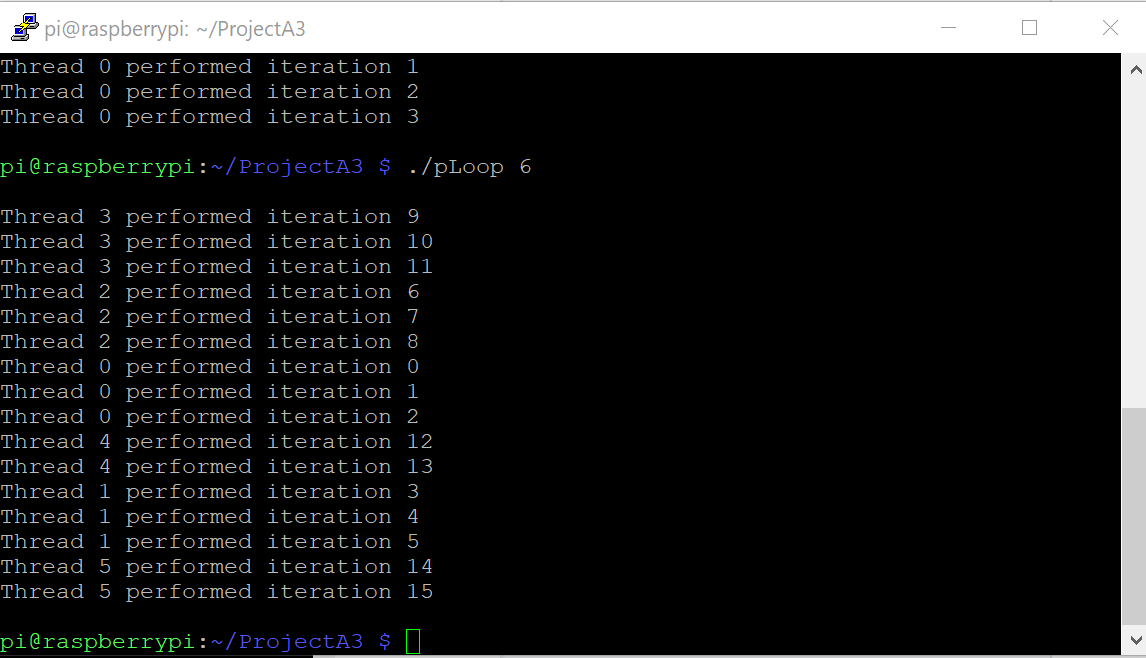
****

*Figure 2 – Running the code with 4 threads*

I then ran the program several more times with different thread numbers as shown in Figures 3 and 4.

****

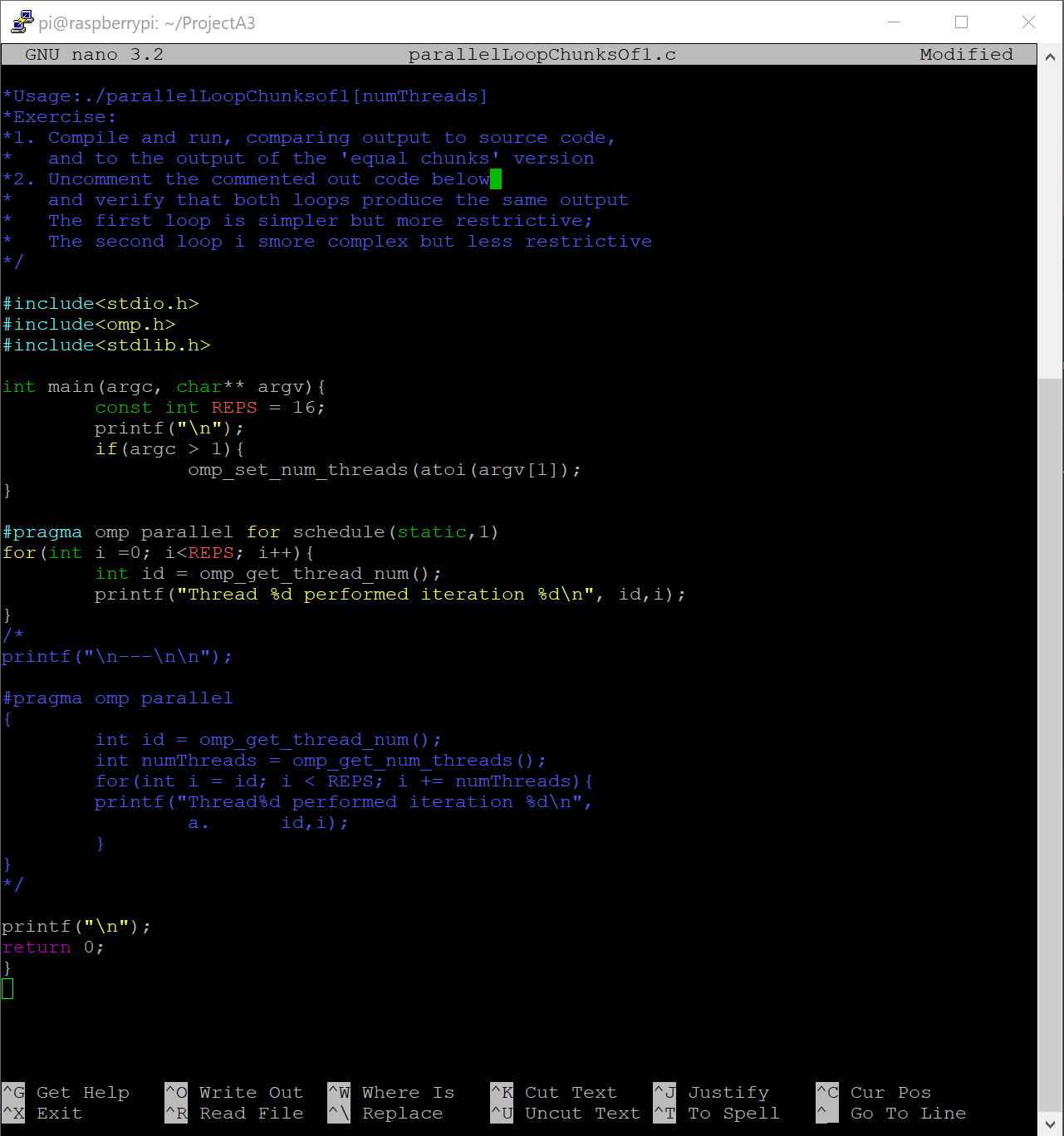
*Figure 3 – running with 5 threads*

****

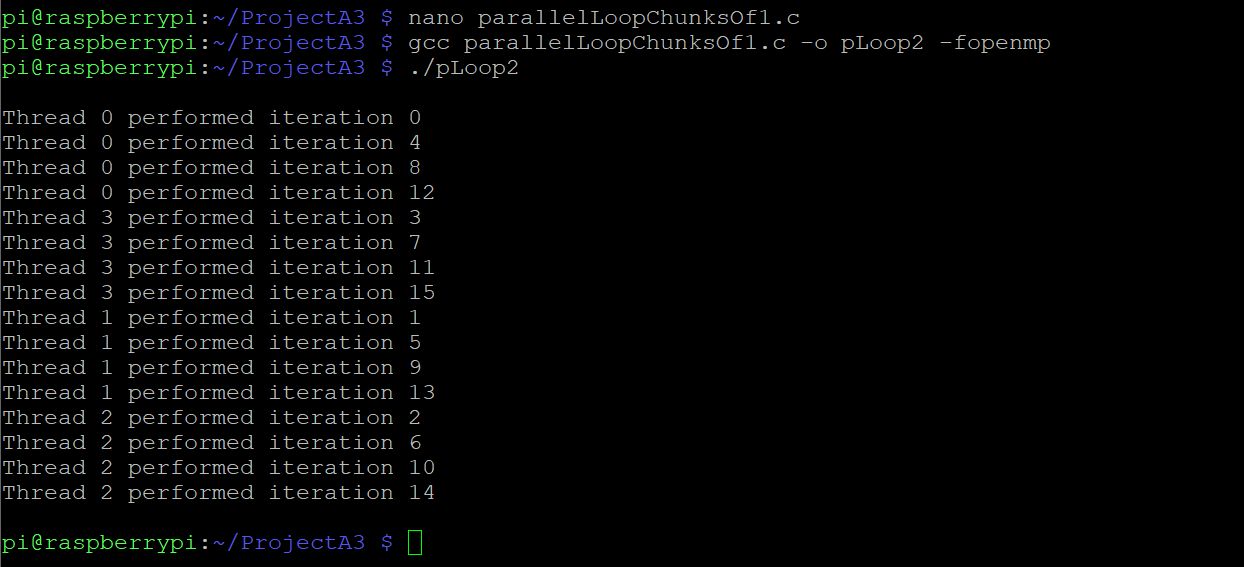
*Figure 4 – running with 6 threads*

This program follows the pattern of decomposition of work into threads as was shown in the instructions. The pattern can be seen for 4,5 and 6 threads.

I then moved on to the next section and wrote the code shown in Figure 5

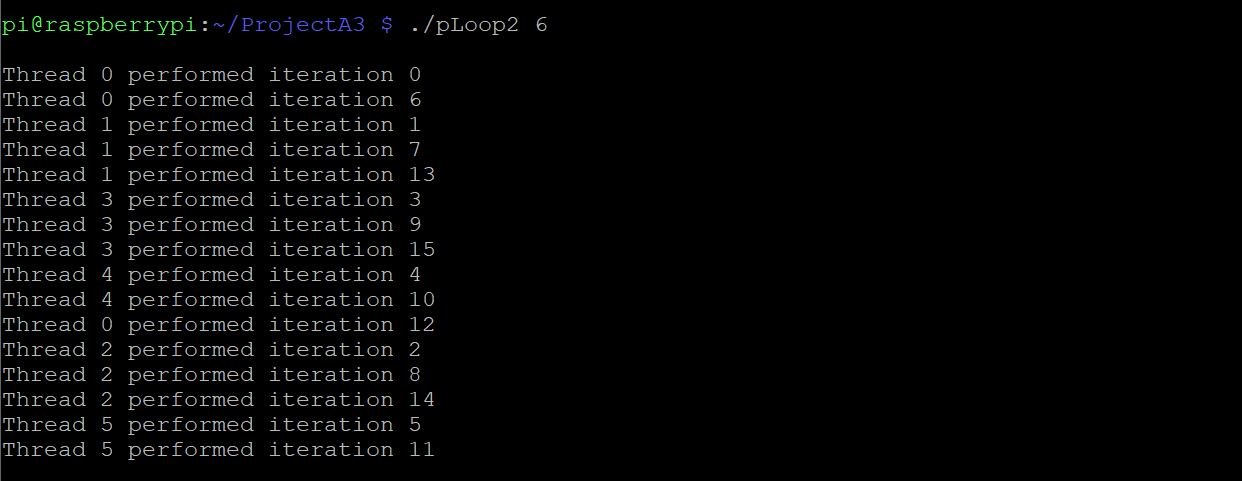


*Figure 5*

I then created an executable program and ran it as shown in Figure 6

*Figure 6*

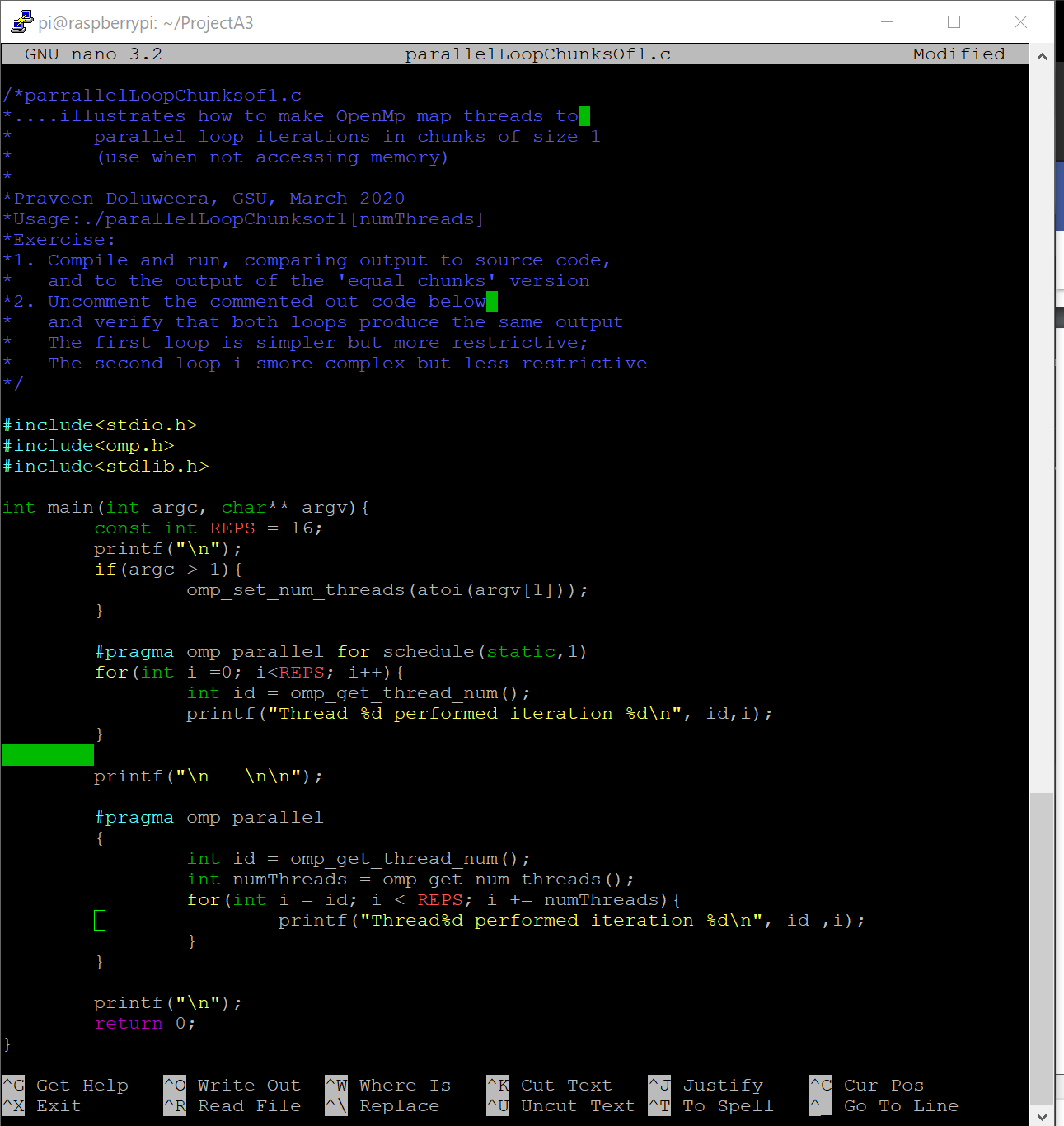
The iterations are ordered so that thread 0 gets the first iteration, thread 1 gets the next and so on until it goes to 3 and then back to 0. This is because there are 4 threads by default. This same pattern can be shown when a different amount of threads are used as in Figure 7 below



*Figure 7*

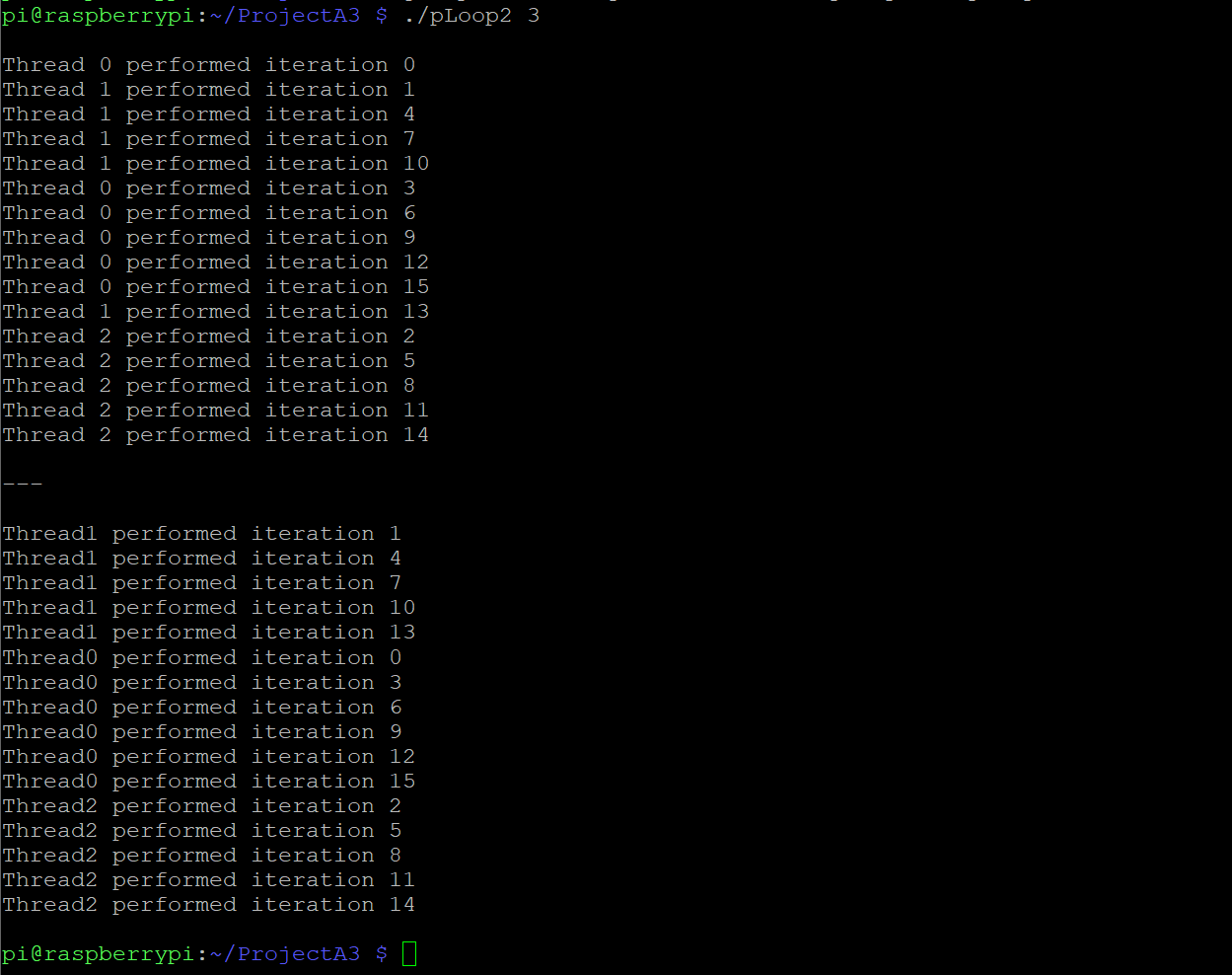
I then uncommented the code so the new parallelLoopChunks.c file now looked like this (Figure

8)

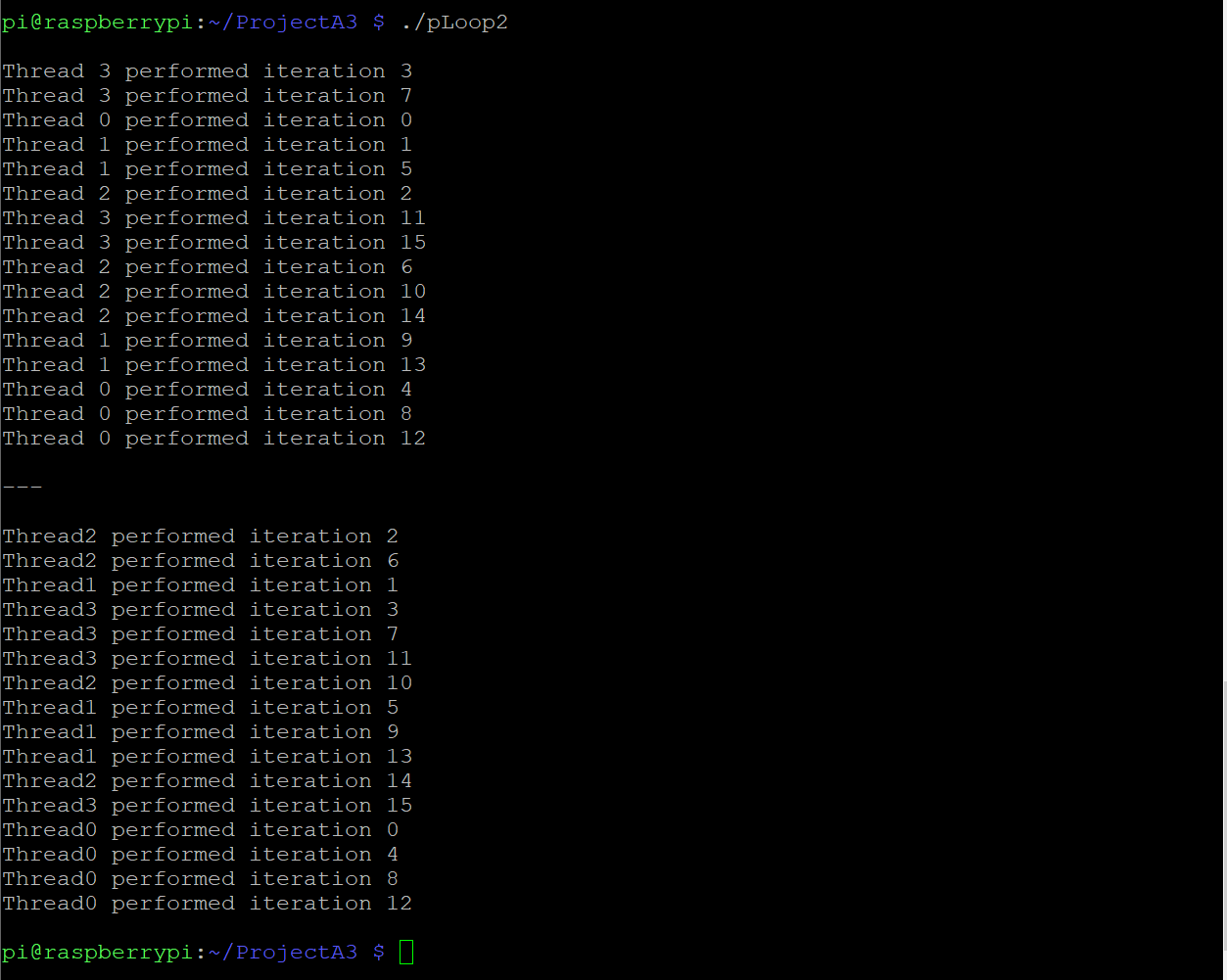


*Figure 8*

I then ran it and as shown in Figures 9 and 10 it has the same pattern shown in the output as before.

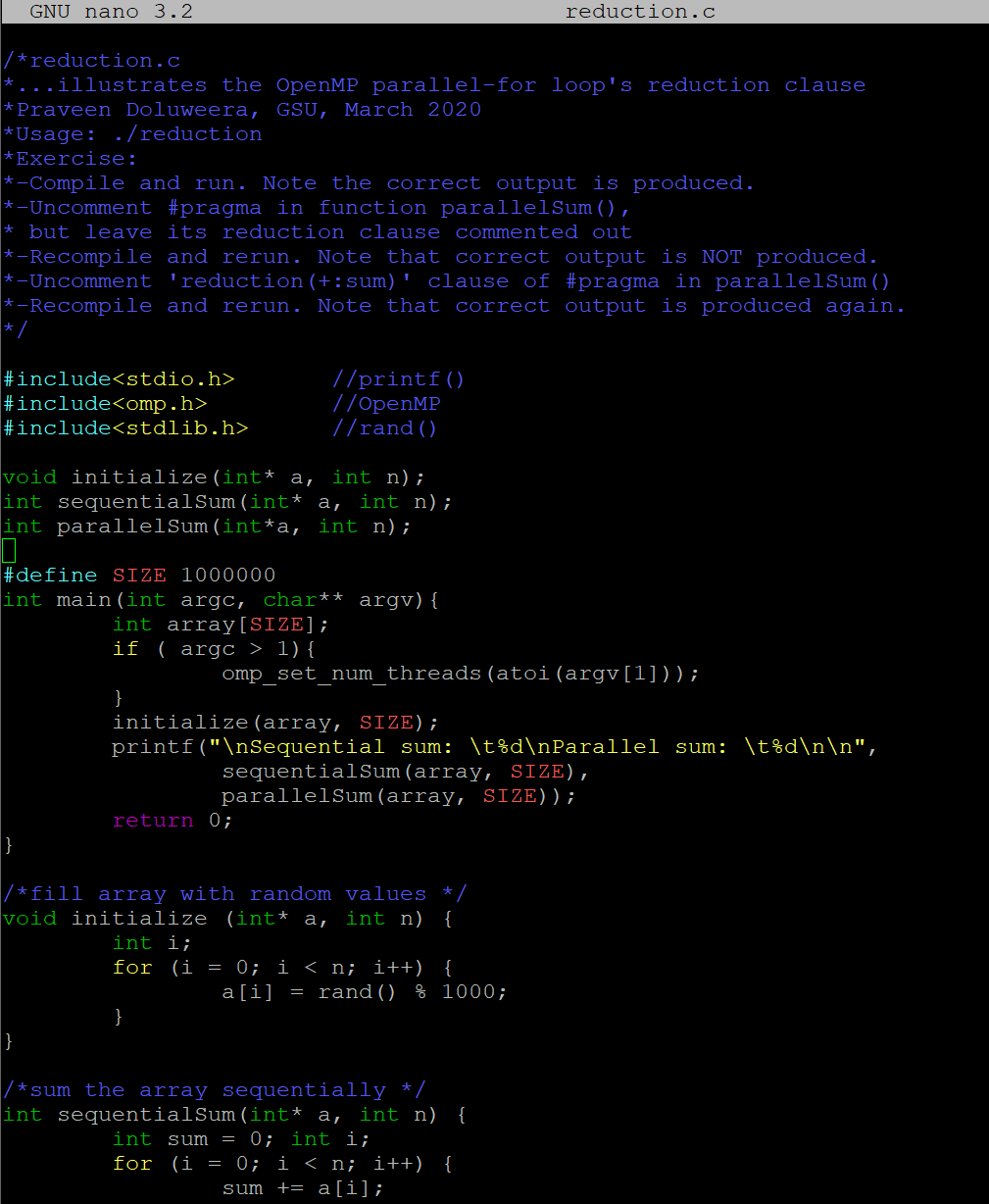


*Figure 9*



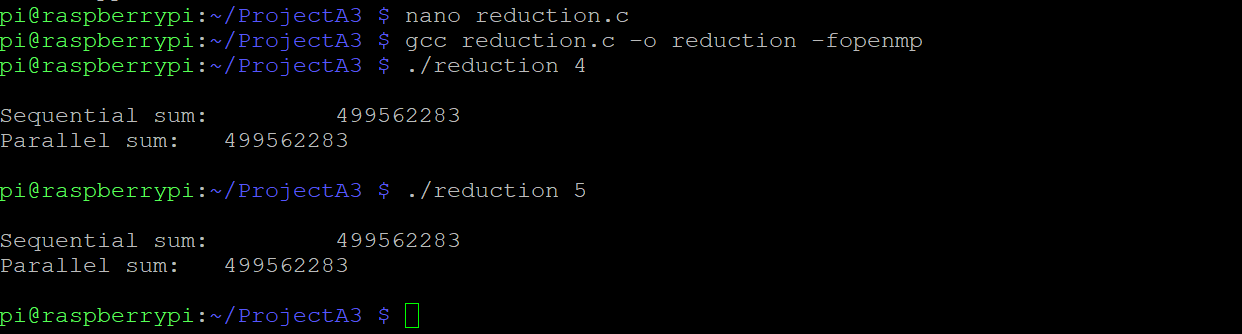
*Figure 10*

I then moved on to the next code and typed out the following for reduction.c as shown in Figure 11



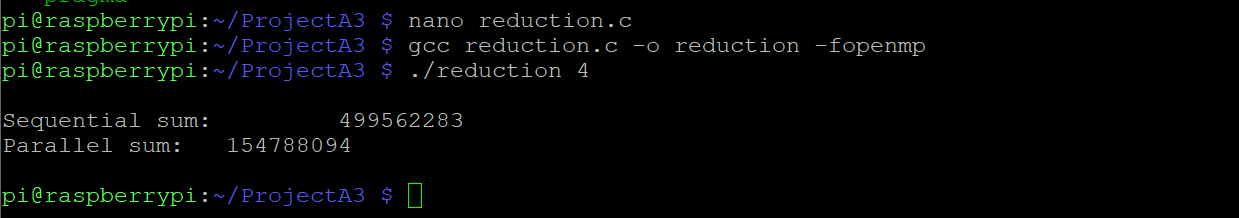
*Figure 11*

When I ran the code, I got the same value for sequential and parallel because the #pragma part of the code was commented. (Shown in Figure 12)



*Figure 12*

I then uncommented out the pragma part and got the following results as shown in Figure 13



*Figure 13*