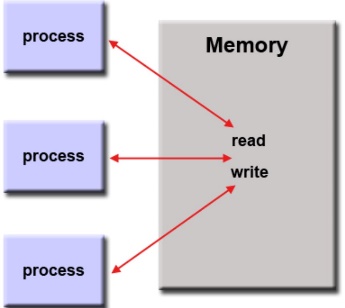
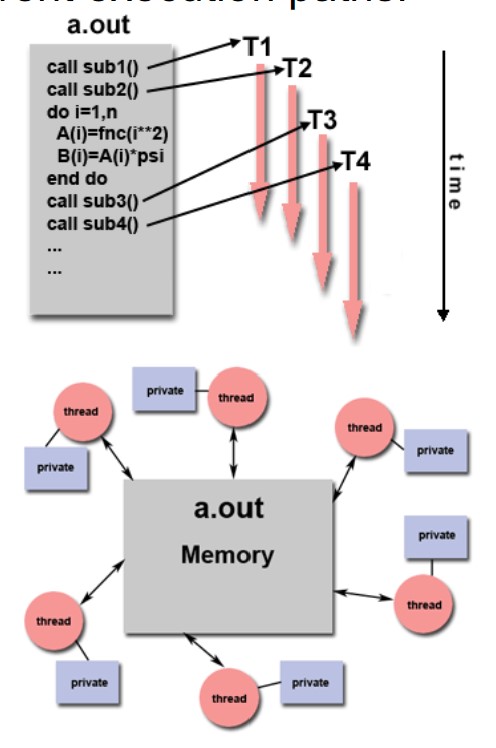
**Victoria Routon**

1. Define the following: Task, Pipelining, Shared Memory, Communications, Synchronization. (in your own words)
   * Task – A set of instructions that is executed by a processor. There are usually multiple tasks that are run by multiple processors in parallel programming.
   * Pipelining – A form of operation that breaks down a task into steps that is performed by different processor units.
   * Shared memory – common physical memory that is used by all the processors; a model where parallel tasks can directly address and access logical memory locations regardless of where the physical memory exists.
   * Communications – data exchange by parallel tasks, which can be accomplished by memory bus or network.
   * Synchronization - The coordination of parallel tasks in real time. It creates a waiting time for application’s wall clock as the tasks need to reach the same or logically equivalent point.
2. Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them.
3. SIMD - (single instruction and multiple data stream):
   * operates on a different data element
   * the same instruction will be executed by all processing units at any clock cycle
4. MISD - (multiple instruction stream and single data stream):
   * single data stream is fed into multiple processing units
   * processing units work on the data independently
5. MIMD - (multiple instruction stream and multiple data stream):
   * each processor is executed with a different instruction stream
   * every processor may work with different data stream
6. 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)
7. List and briefly describe the types of Parallel Computer Memory Architectures.
   * Uniform Memory Access (UMA):
     + 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 physically linking two or more SMPs and share memory with each other. But different processors might have different access time to memories, it will take more time when cross memories. If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA.
8. What type is used by OpenMP and why?
   * NUMA is used by OpenMP. Because OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. In order to achieve that, it requires different types of processors involved, only NUMA can make it possible.
9. Compare Shared Memory Model with Threads Model? (in your own words and show pictures)
10. Shared memory model (without threads): processes(tasks) share a common address space that they can read and write to in an asynchronous way
    * Advantage: Easily to write an application. All processes have equal access to the shared memory.
    * Disadvantage: It is hard to manage and understand data.



*Figure 1 – Shared Memory Model*

1. Threading model: It is a type of shared memory programming which has a single “heavy weight” process can have multiple concurrent execution paths.
   * Advantage: More suitable for applications based on the multiple data.
   * Disadvantage: It is hard to write applications using threading models since it usually has complex instructions.



*Figure 2 – Threading Model*

1. What is Parallel Programming? (in your own words)
   * Parallel programming is the use of multiple processing elements simultaneously for solving problems.
2. What is system on chip (SoC)? Does Raspberry PI use system on SoC?
   * An SoC, or system-on-a-chip integrates components that include CPU, GPU (a graphics processor) and a memory on a single silicon chip. Example of such device that uses SoC is a Raspberry PI.
3. Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.
   * SoC has smaller size, it is only a little larger than a CPU and contains more functionality. Therefore, it can help to make smaller computers, like smartphones. SoC also uses less power, and it is cheaper to build a computer using SoC.

Part B

Parallel Programming

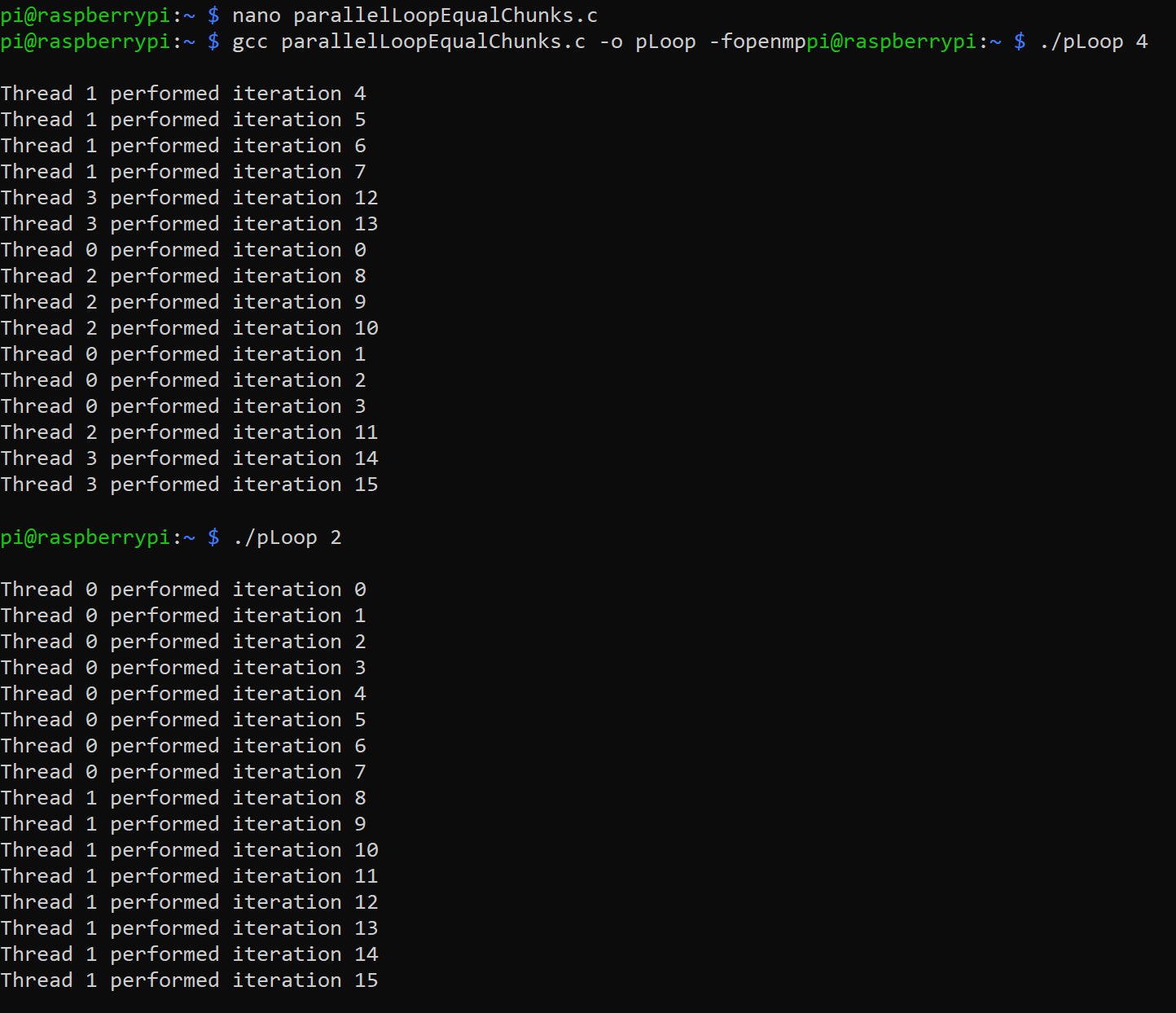
I began by typing the parallelLoopEqualChunks code into Raspberry Pi.

A screenshot of a cell phone

Description automatically generated

*Figure 1 – ParallelEqualChunks program*

Then, I made an executable program and ran it (Figure 2). I used different threads numbers and observed an output. Since the threads are running in parallel, they all are being executed at the same time hence the thread numbers are not in order. However, the for loop starts from the element at the position 0, which explains why the thread 0 will be iterated first and after that the next thread 1 will be iterated and so on.



A screenshot of a computer

Description automatically generated

Figure 2 – running the code ParallelLoopEqualChunks

Next, I typed ParallelLoopChunksOf1 program in the Raspberry PI.

Parallel Loop Chunks Of 1

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

*Figure 3 – ParallelLoopChunksOf1 code*

Just like in the previous program, I made it an executable and ran it (Figure 4). I can see the similarities between those programs as it, it also iterates the thread 0 and then moves to thread 1 and then thread 2 and so on.

A screenshot of a computer

Description automatically generated

*Figure 4 – running ParallelLoopChunksOf1 program*

To check if both loops produce the same output, I uncommented the 2nd loop and ran the program again (Figure 5). I noticed that indeed, both loops create the same output.

A screenshot of a computer

Description automatically generated

*Figure 5 – running both loops of ParallelLoopEqualChunksOf1*

Lastly but not least, I created a program using a nano command and named it reduction (Figure 6).

A screenshot of a cell phone

Description automatically generated

A screen shot of a smart phone

Description automatically generated

*Figure 6 – reduction program*

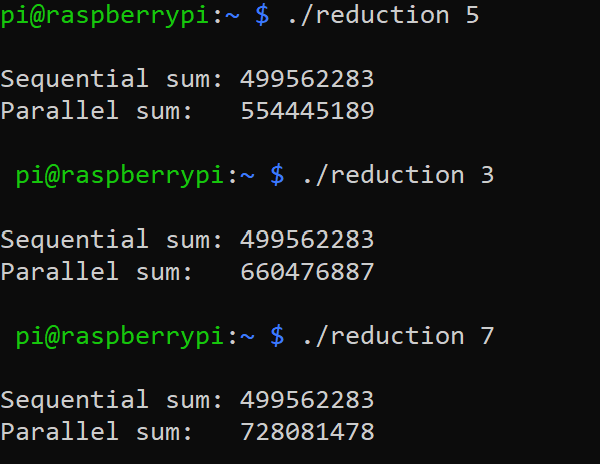
After I made it an executable, I ran the program. I noticed that the values of both Sequential and Parallel sums are the same (Figure 7).

A screenshot of a cell phone

Description automatically generated

*Figure 7 – running reduction program*

Then, I uncommented the “#pragma” part of the program and ran it (Figure 8).



*Figure 8 – reduction program after uncommenting “#pragma”*

Then, I uncommented “reduction(+:sum)” part of the program and ran it (Figure 9).

'A screenshot of a cell phone

Description automatically generated

*Figure 9 – reduction program after uncommenting “reduction(+:sum)”*

Reduction program uses clause on a parallel for loop program to calculate the sum. I noticed that when I removed reduction clause (“reduction(+:sum)”, it produced incorrect output. The reason why this happened because when we declare variables on parallel programing, they should be private to each thread to make sure the program will run correctly.