Developing Soft and Parallel Programming Skills Using

Project-Based Learning

semester (Spring-2019)

Group name: Kenny and the Rest

Members name: Nish Patel, Abdiaziz Dakane, Soojie Chin, Brett Krokoff and Kehinde Adedara

Planning and scheduling

Name	Email	Task	Duration (Hours)	Dependency	Due Date	Note
Abdiaziz Dakane (Group coordinator)	Adakane1 @student.gsu.edu	Task 4 and 3b:	4 Hours	Carrying out experiment on raspberry pi	10/16/20	Follow all the instructions before the deadline and make sure all the screenshots are taken.
Brett Krokoff	Bkrokoff1 @student.gsu.edu	Task 1 and 2:	2 – 3 Hours	(Group Co-Ordinator) Planning and scheduling.	10/16/19	Make sure there is communication, among the teammate, checking on everyone and ensuring they are on track.
Soojie Chin	Schin6 @student.gsu.edu	Task 3a:	3 Hours	Answering questions regarding	10/16/20	All questions asked

				foundation in parallel programming.		about the Raspberry Pi project is answered. Communicate and help Abdiaziz who oversees the experiment.
Kehinde Adedara	Kadedara1@student.gsu.edu	Task 6:	2-3 hours	Making sure everyone is there for the video as well as editing and uploading.	10/17/20	Make sure everyone is prepared for the video and it is uploaded and edited for visibility and clear sound
Nish Patel	Npatel1161@student.gsu.edu	Task 5:	2 hours	Reporting on observation and experimentations made on soft and Parallel programming using raspberry pi.	10/17/20	Report, screenshots, as well as other required documents should be included in the report by Thursday.

Parallel Programming Skills

Part A

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

Task – a set of instructions for the processor to execute

Pipelining – a type of parallel computing where a task is broken up for different processors to execute

Shared memory – computer architecture where processors have direct access to physical memory (hardware POV); parallel tasks can have direct address and same logical memory (programming POV)

Communications – data exchange

Synchronization – coordination of parallel tasks where, most of the times, one task needs to reach the same logical point

Classify parallel computers based on Flynn's taxonomy. Briefly describe every one of them.

Flynn's taxonomy helps distinguish multi-processor computer architecture through instruction stream and data stream. Instruction and data stream can be separated through single or multiple state. There are four possible classifications: SISD (single instruction, single data), SIMD (single instruction, multiple data), MISD (multiple instruction, single data), MIMD (multiple instruction, multiple instruction). SISD works with the oldest type of computers where one instruction is processed by the CPU in one clock cycle with only one data input. SIMD computers process one instruction through multiple processing units and work best for graphics and image processing. There are two types of SIMD: processor arrays and vector pipelining. MISD has each processing unit process data independently through separate instruction streams with singular data as input. Not many MISD computers exist. MIMD computers have multiple instructions being processed through multiple processing units. Most modern day computers fall into this category such as supercomputers, gaming computers, multicore PC's.

What are the Parallel Programming Models?

Parallel Programming models are abstractions above hardware and memory architectures. The models include shared memory (without threads), threads, distributed memory/message passing, data parallel, hybrid, SPMD (single program multiple data), and MPMD (multiple program multiple data).

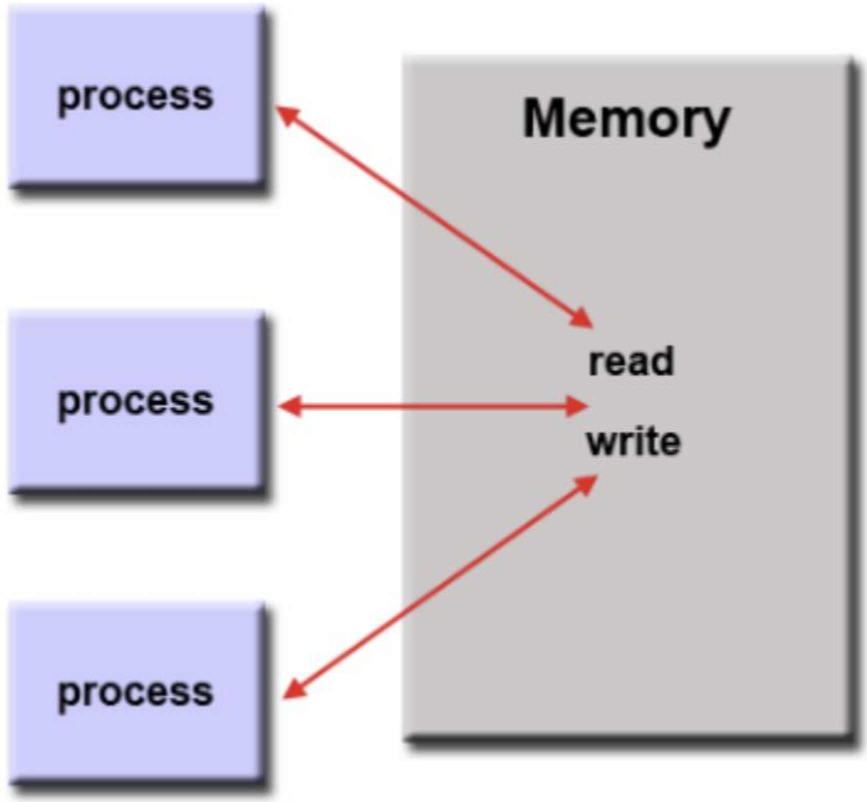
List and briefly describe the types of Parallel Computer Memory Architectures. What type is used by OpenMP and why?

There two common types of shared memory parallel computer memory architectures: UMA (uniform memory access) and NUMA (non-uniform memory access). UMA uses identical processors and has similar access and access times to memory. UMA is also commonly called

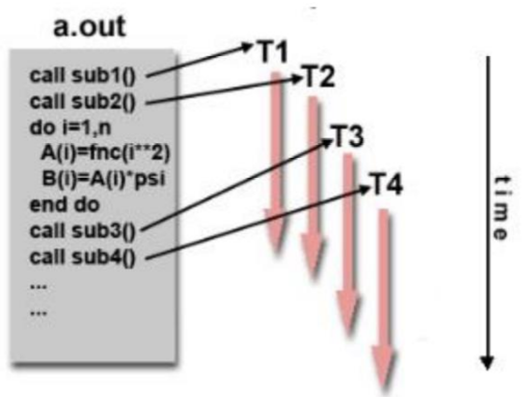
Cache Coherent UMA, which means that if one processor updates a location in shared memory, all the other processors are also aware of the update. All of this is executed at the hardware level. NUMA works by mostly physically linking two or more SMP's (symmetric multiprocessor). One SMP can directly access memory to another SMP, BUT not all processors have the same access time to the rest of the memories and the link time is slower. OpenMP uses UMA because each processor can have a private cache memory.

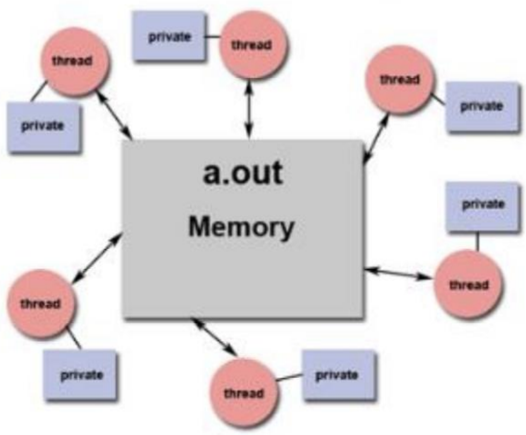
Compare Shared Memory Model with Threads Model? (in your own words and show pictures)

With shared memory, tasks share the same address and is the simplest parallel programming model. An advantage a programmer has with this model is that there is no need to explicitly communicate data between the tasks since they have shared memory. However, it becomes increasingly difficult to manage and understand local data.



The threads model uses a single "heavy weight" process that can have multiple "light weight" execution paths. Threads usually require synchronization and communication through global memory. a.out is an example of using threads to execute a program by communicating through global memory. Each thread contains local data, but then shares all contents and memory space of a.out. Some operating system such as Unix/Linux use threads like POSIX threads. POSIX threads are C language only and require close attention to detail. OpenMP is an example of portable threads and is available only multiple platforms and available in C, C++, and Fortran languages.





What is Parallel Programming? (in your own words)

Parallel programming is using multiple processors to execute a single or set of instructions. It ensures quicker execution time and is useful for executing complex programs. However, a slight disadvantage to parallel programming is that if the programmer wants to tweak the code, they would have to target different points of the code to improve it. It is not as straightforward as serial programming.

What is system of chip (SoC)? Does Raspberry PI used system on SoC?

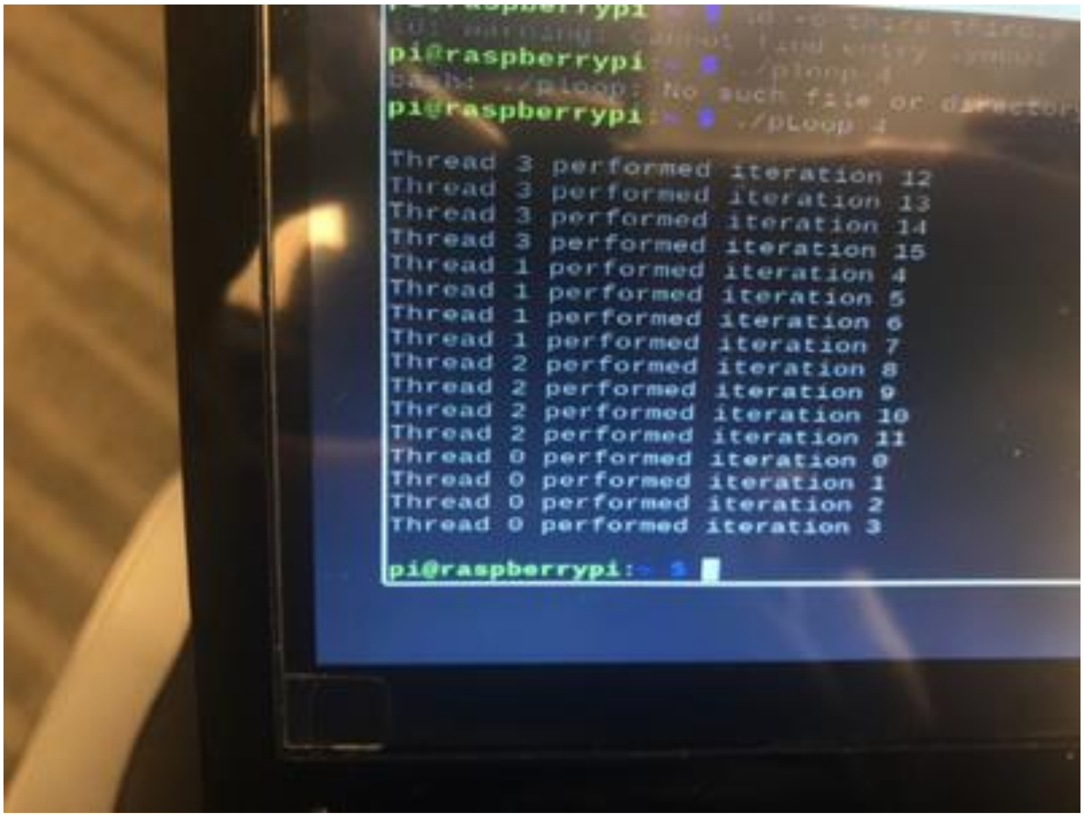
A SoC contains all the computer components into a single silicon chip. It usually contains a CPU, GPU, memory, USB controller, power management circuits, and wireless radios. A system that uses SoC is Raspberry PI.

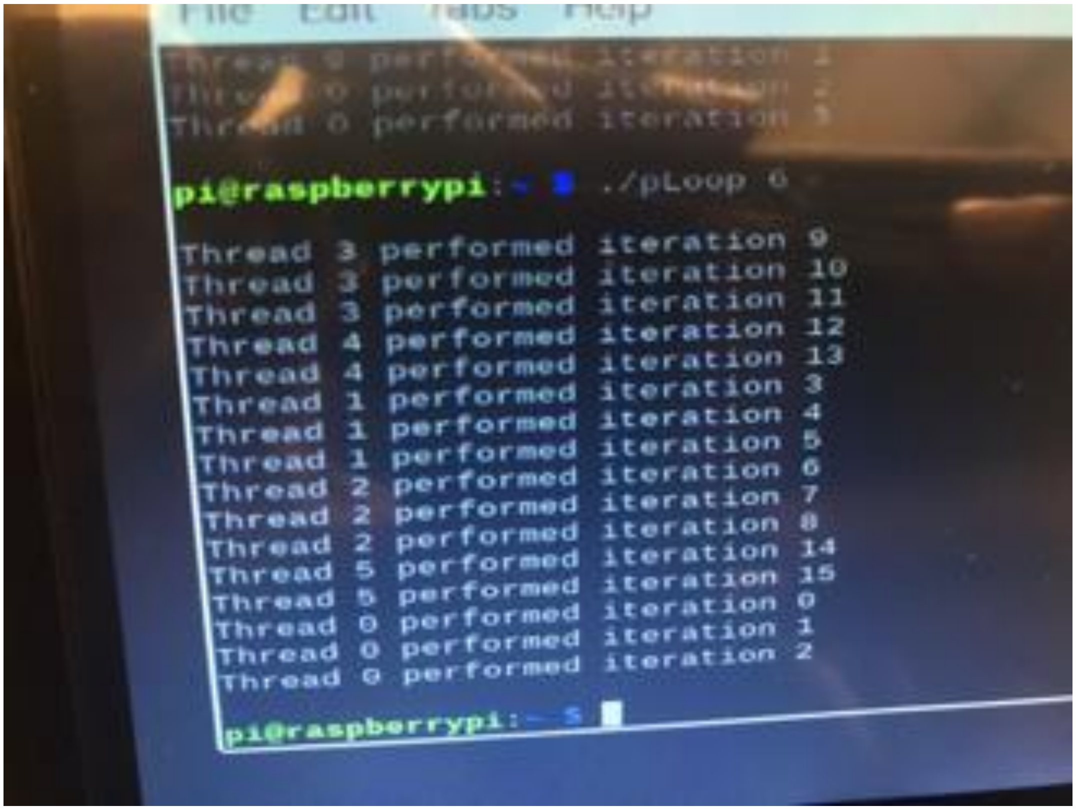
Explain what the advantages are of having a System on a Chip rather than separate CPU, GPU and RAM components.

Significant advantages of having a System on a Chip rather than a separate CPU, GPU, and RAM components are its size vs. functionality and requires less power. The SoC is only slightly bigger than a CPU, but it has a lot more functionality that just a separate CPU. Because of small size and high functionality, you can use SoC's to build smartphones and tablets. Also due to high integration and shorter wiring, the SoC requires less power. This is most convenient for mobile computing. Furthermore, the SoC uses less number of chips, so it is also cost effective.

Part B:

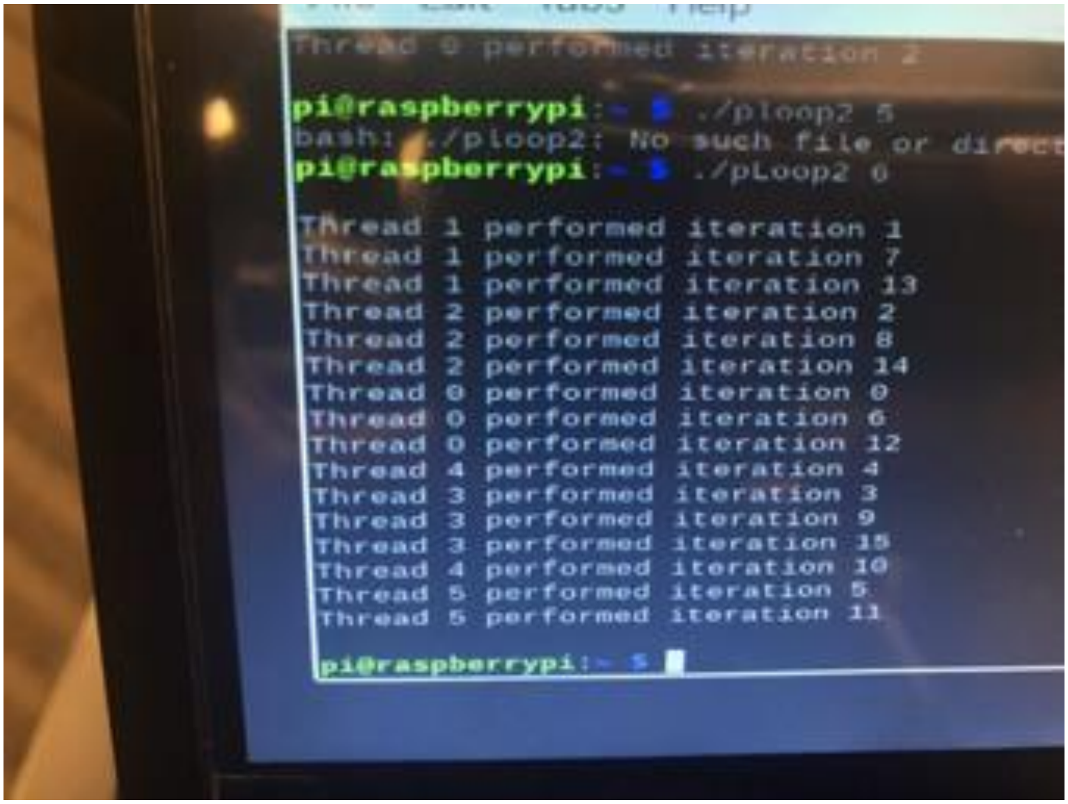
pLoop1:

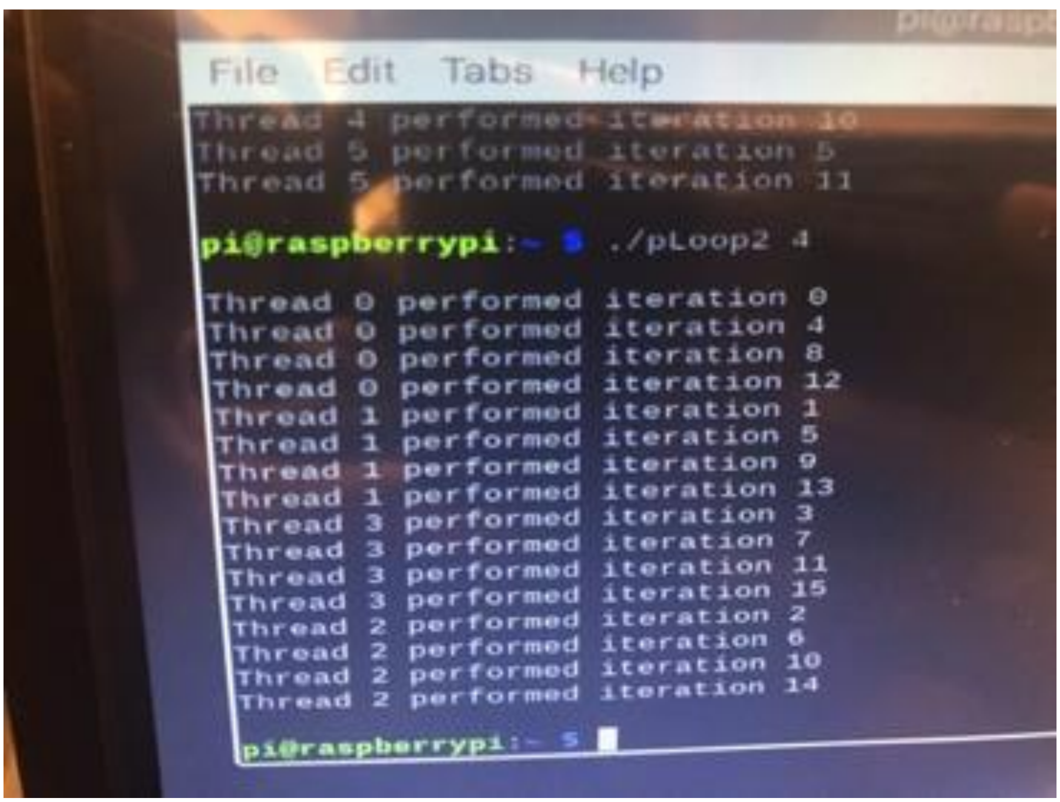




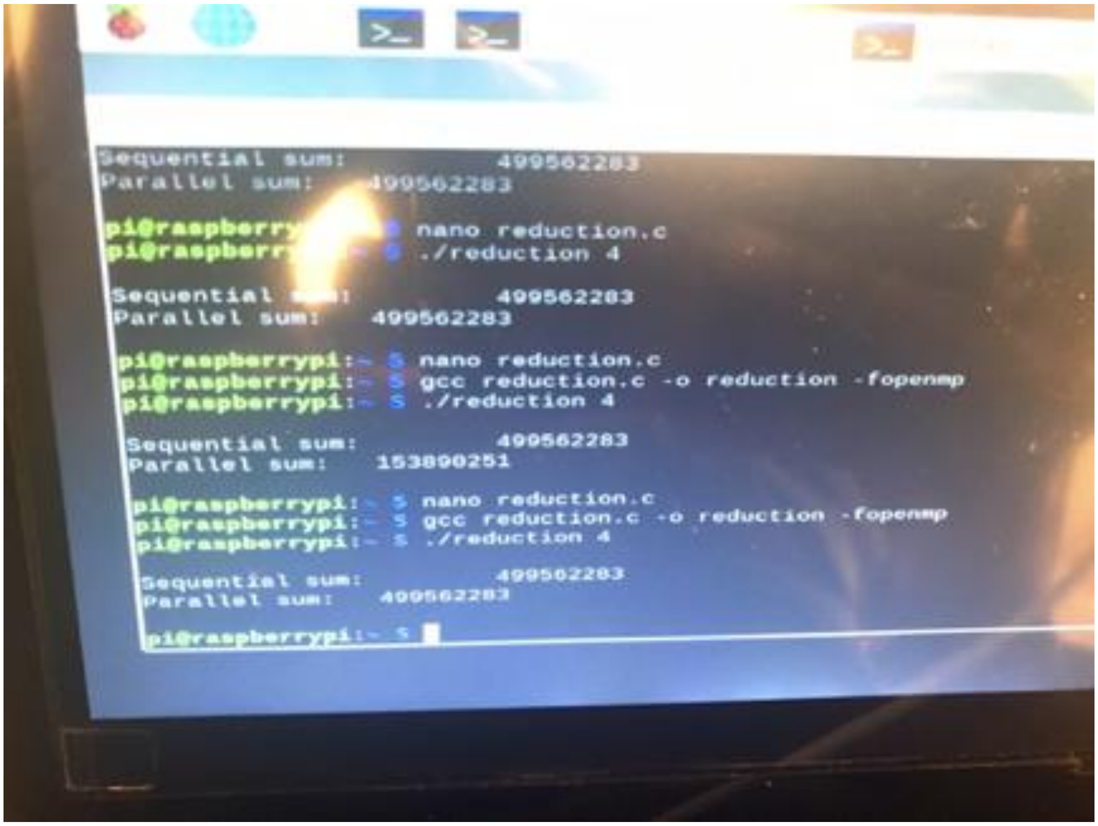
As you can see the difference between./pLoop 4 and ./pLoop 6 is that in pLoop 4 the number of thread is in a descending order but in pLoop 6 the threads are in random order.

Ploop2:





As you can see pLoop2 and pLoop are arranged in different orders. The difference was in first loop each thread performed the iteration is in order for example if thread 0 performed from iteration 1 to 3 then thread 1 had to perform from iteration 2 but In the second loop the iteration and the thread correlation was if thread 0 went from 0 to 4 iteration it meant it kept adding up by 4 so next thread 0 will have 8. If thread 1 had three threads and the first one was 1 and second one was 6 it will keep adding 5 so next thread 1 will have iteration of 11.

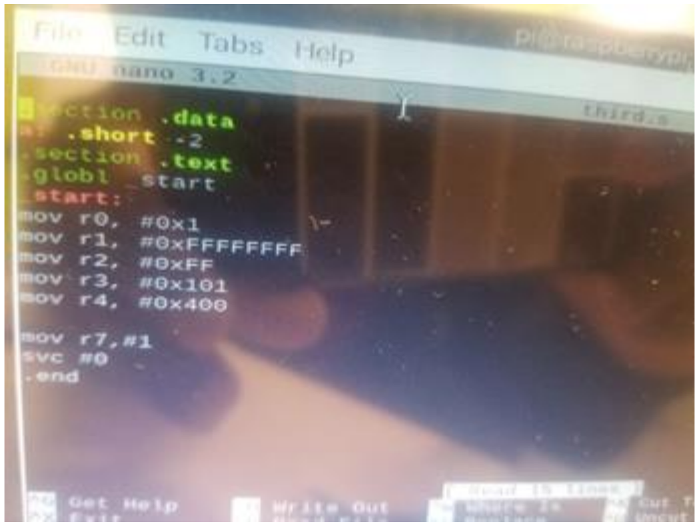


- In the first ./reduction 4 I ran I did not take out the // comment in line 39 and this was the output we got.
- In the second code we ran all we deleted was the “//” infront of pragma and we got a different output for parallel sum. The sequentialSum and parallelSum did not match to the first output when we deleted the first comment infront of “#pragma”
- In the third code we deleted all the comments from line 39 and we managed to get right output again.

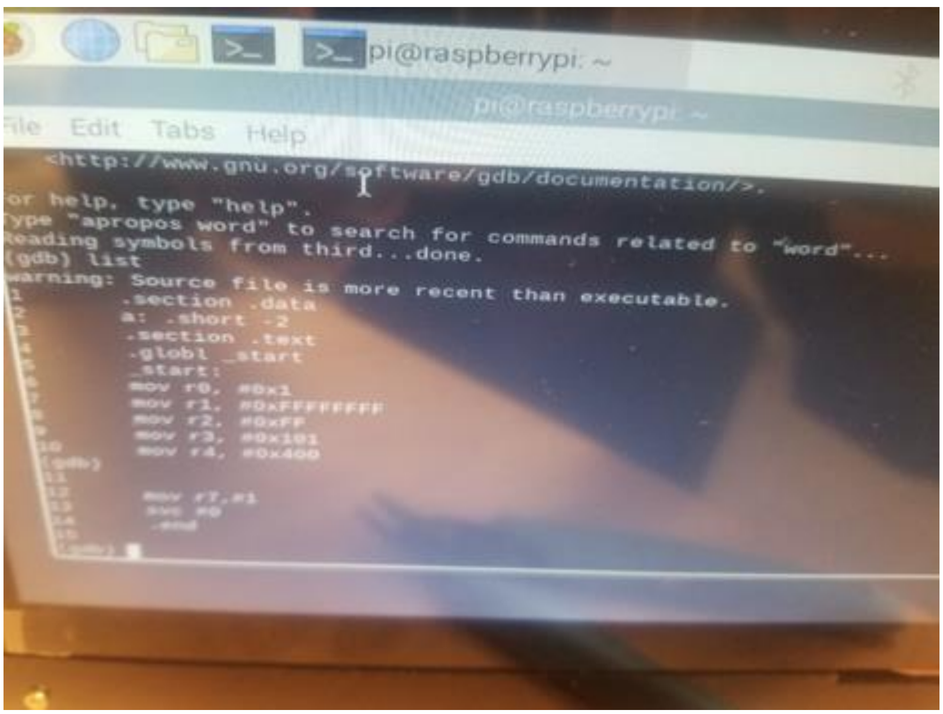
ARM Assembly Programming

1. Our issue is compiling the code and loading the file for execution. However, we are not able to achieve that because of the errors we get.

Solution: we can replace .shalfword, in the code, with .short.

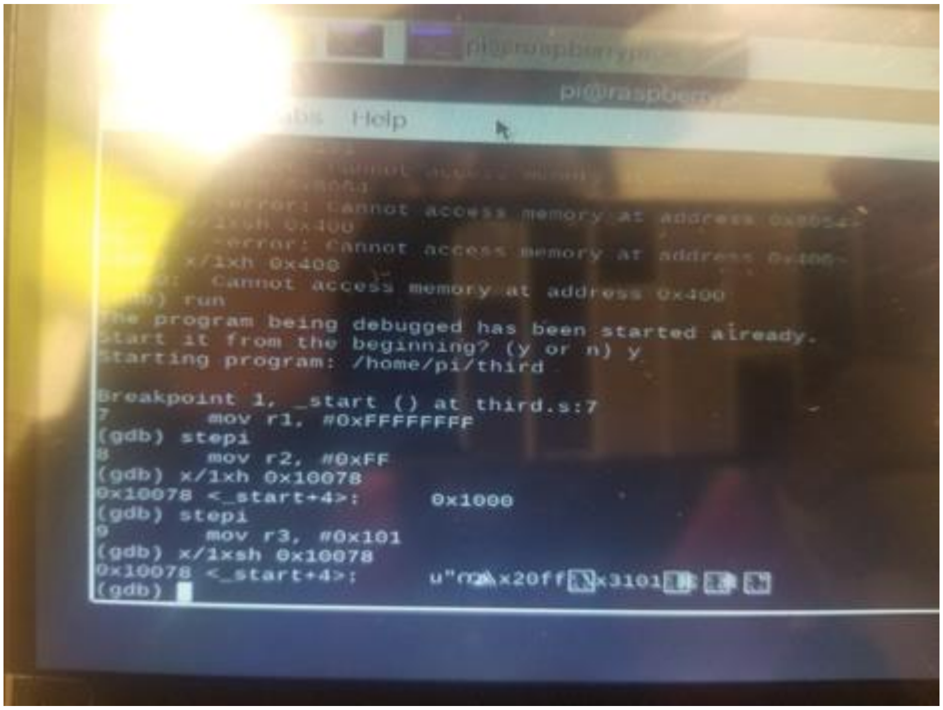


And now when we run and execute the code, we have this:

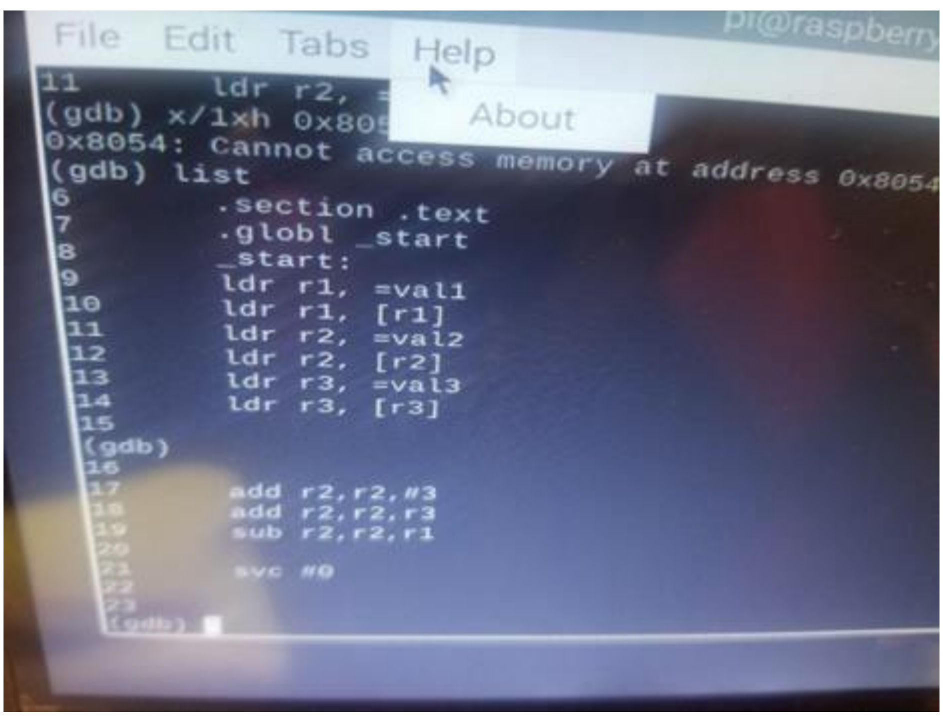


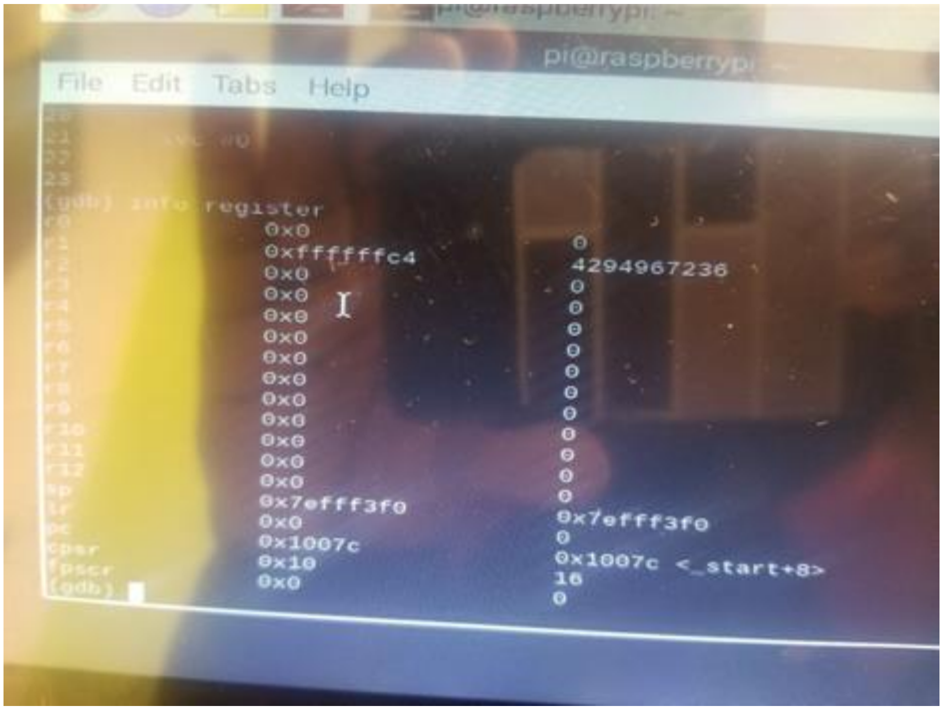
we get an error with just the h:

But with sh, we get this: The values are encoded however.



1. Adding the registers to give us an answer
2. When we ran and compile the program, the value of the register gives us the answer 90.
3. If we offset by every statement in the line, we get the updated value as it is being added and subtracted from the registers.





Appendix

YouTube video link:- https://youtu.be/KHDsyK9EeKs