**3B.**

**RUNNING LOOPS IN PARALLEL**

**2.1**

With the numbers of iterations divisible by the number of threads, the iterations will be divided evenly to the numbers of threads

./pLoop 4

A screenshot of a computer

Description automatically generated

./pLoop 2

A screen shot of a computer

Description automatically generated

With the numbers of iterations not divisible by the number of threads, each thread will evenly take the largest number of iterations possible, the remainder will be distributed from thread 0 -> last thread one by one iteration.

./pLoop 3

A screenshot of a computer

Description automatically generated

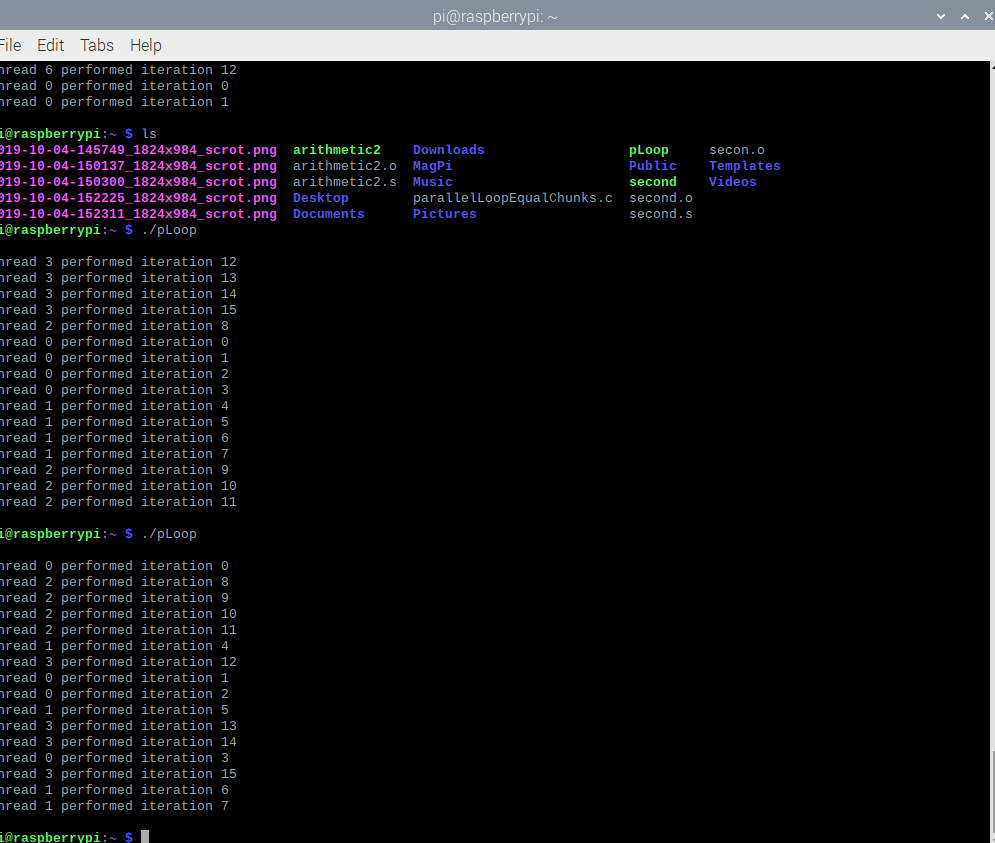
./pLoop 10

A screenshot of a computer

Description automatically generated

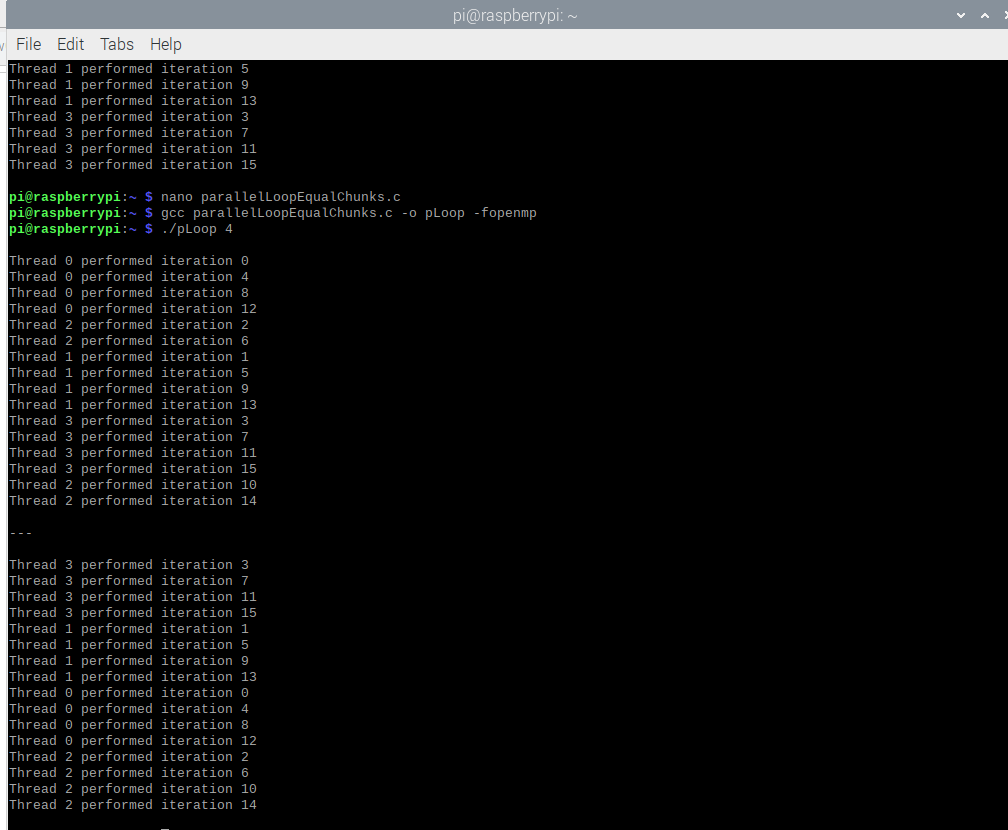
If we leave off the number, it will automatically run as 4 threads since the Raspberry Pi got 4 cores.

./pLoop

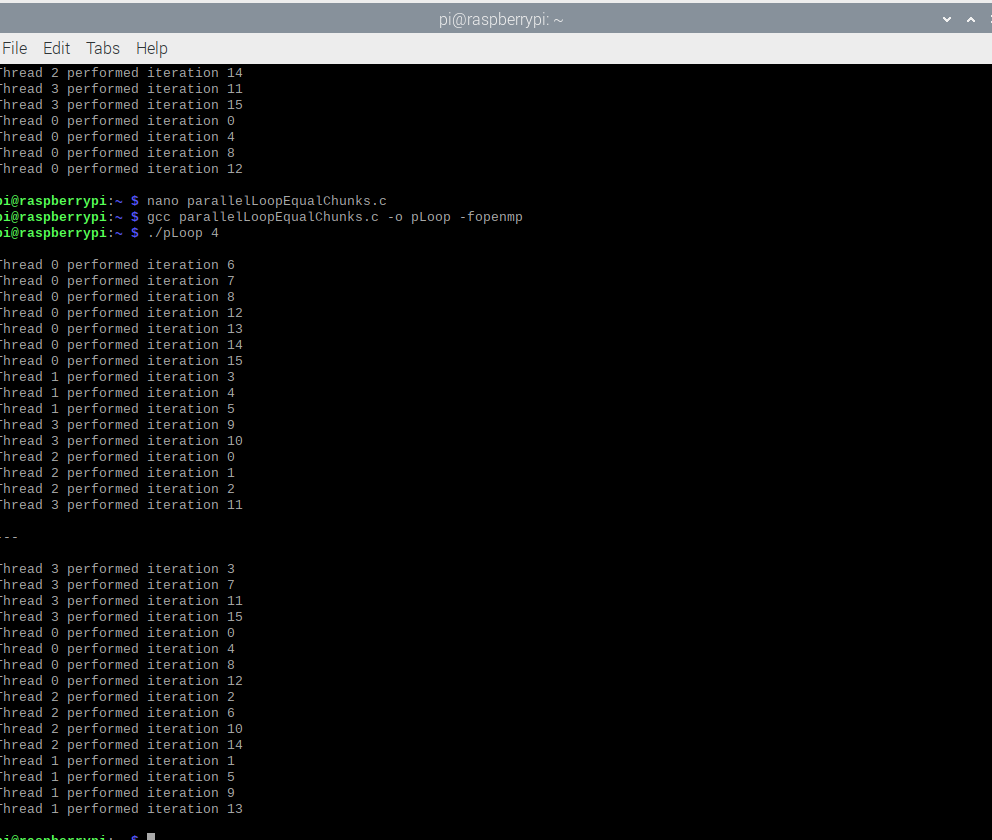


**3.3.1**

Both loops producing the same output



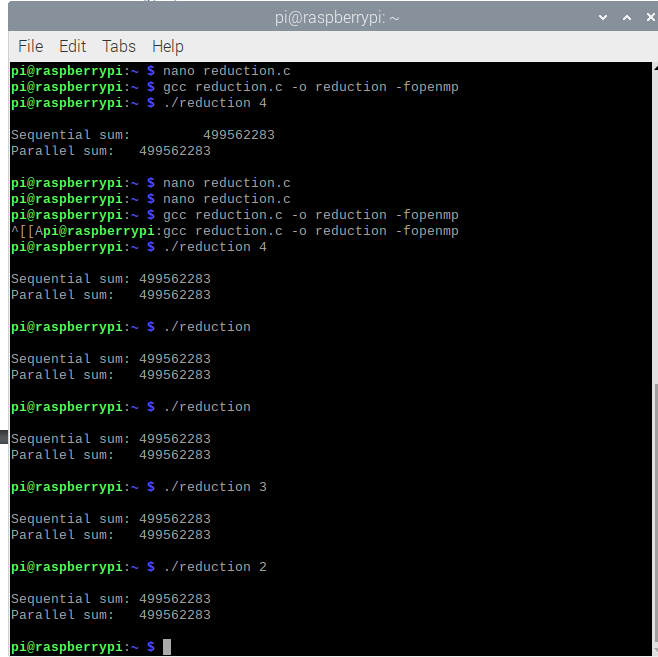
changing (static,1) to (dynamic,3)



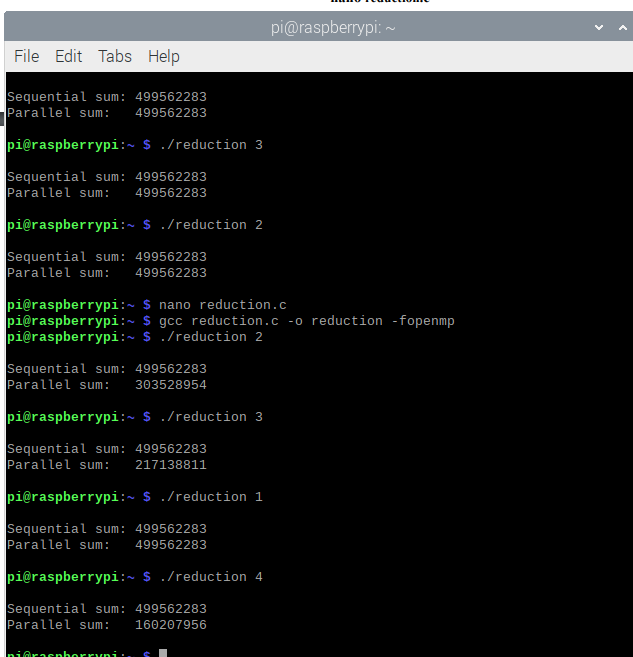
**WHEN LOOPS HAVE DEPENDENCIES**

After running the program, the output of sequential sum and parallel sum are the same with 4 or different number of threads, because we have commented out the statement #pragma omp parallel for reduction(+:sum) in the parallelSum function

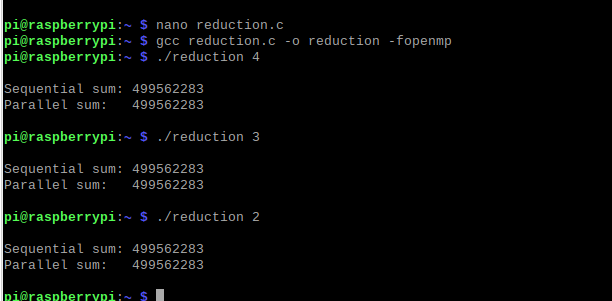
./reduction 4; ./reduction; ./reduction 2; ./reduction 3



Removing the first comment: the sequentialSum function results is different from the parallelSum function results. Because the accumulator called sum is not private, they return their individual sum of each thread.



Removing the second comment: the sequentialSum function results is the same with the parallelSum function results ( with 4 or different threads). Because we have uncommented the reduction clause, it makes the accumulator private to each thread and sum up their individual sums.



**4.**

**Part 1: Third Program**

Error:

Change code: