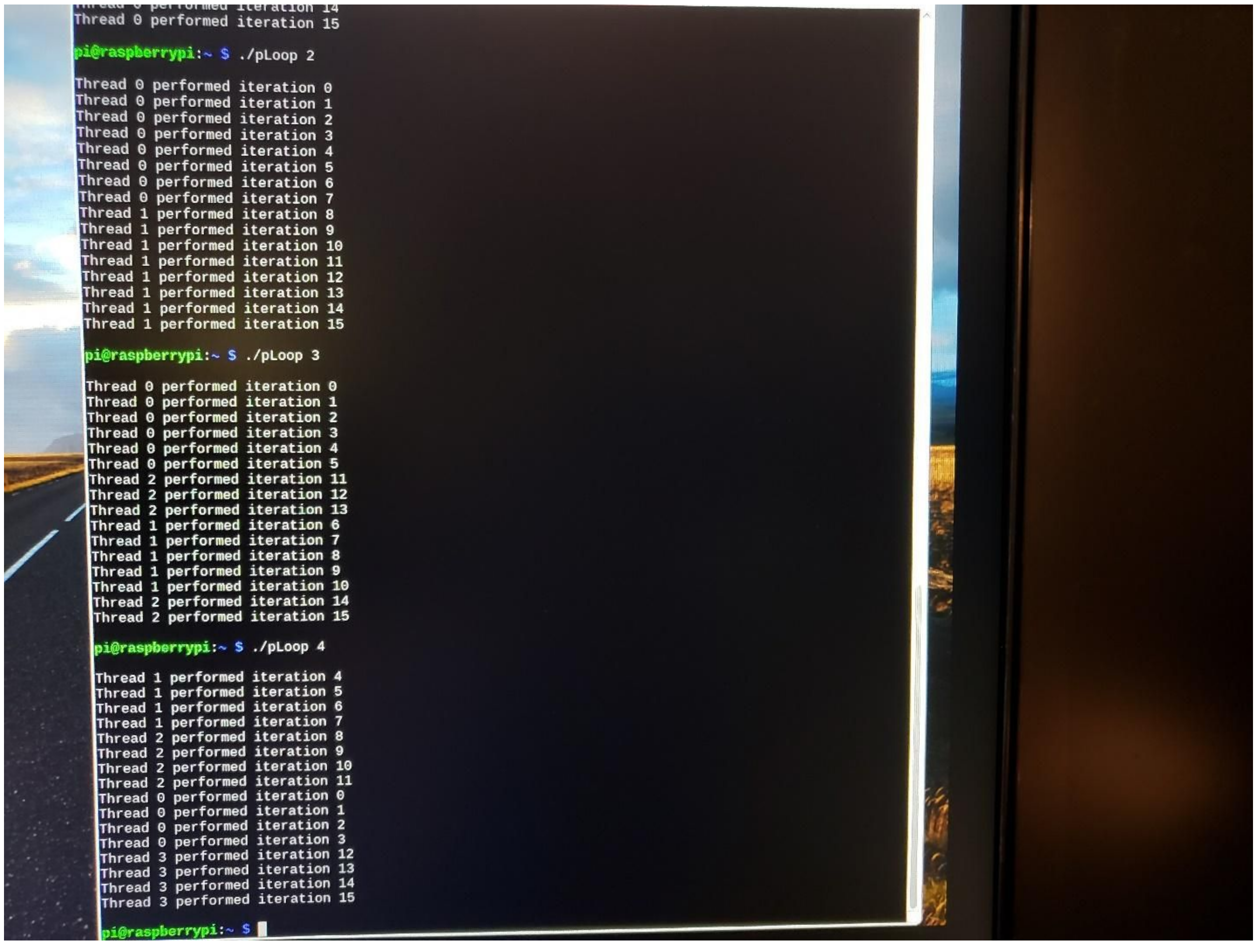
Lab Report

When running the program pLoop the value that is added to the right determines the number of threads that are going to be used. If a value of 1 is put after pLoop then all of the data will be processed by one thread, while an input of 4 will evenly divide the work across all 4 threads. If the number of threads is not an even number, then the data will be added to threads making their workload more than other threads. A value above 16 cannot be put since the number of iterations is 16. Also, when each thread will process consecutive data that it is assigned. For example, pLoop 4 has thread 1 doing iteration 4,5,6,7 each consecutively in order.



```
Thread 0 performed iteration 14
Thread 0 performed iteration 15

pi@raspberrypi:~ $ ./pLoop 2

Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 0 performed iteration 4
Thread 0 performed iteration 5
Thread 0 performed iteration 6
Thread 0 performed iteration 7
Thread 1 performed iteration 8
Thread 1 performed iteration 9
Thread 1 performed iteration 10
Thread 1 performed iteration 11
Thread 1 performed iteration 12
Thread 1 performed iteration 13
Thread 1 performed iteration 14
Thread 1 performed iteration 15

pi@raspberrypi:~ $ ./pLoop 3

Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 0 performed iteration 4
Thread 0 performed iteration 5
Thread 2 performed iteration 11
Thread 2 performed iteration 12
Thread 2 performed iteration 13
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 1 performed iteration 8
Thread 1 performed iteration 9
Thread 1 performed iteration 10
Thread 2 performed iteration 14
Thread 2 performed iteration 15

pi@raspberrypi:~ $ ./pLoop 4

Thread 1 performed iteration 4
Thread 1 performed iteration 5
Thread 1 performed iteration 6
Thread 1 performed iteration 7
Thread 2 performed iteration 8
Thread 2 performed iteration 9
Thread 2 performed iteration 10
Thread 2 performed iteration 11
Thread 0 performed iteration 0
Thread 0 performed iteration 1
Thread 0 performed iteration 2
Thread 0 performed iteration 3
Thread 3 performed iteration 12
Thread 3 performed iteration 13
Thread 3 performed iteration 14
Thread 3 performed iteration 15

pi@raspberrypi:~ $
```

Figure 1: Code running of pLoop

When running pLoop2 with a value of 4 then the iterations are divided evenly through the threads and the thread is responsible for iteration of the thread number then add the value after pLoop2. For example, with pLoop2 4, thread 1 is responsible for iteration 1, 5, 9, 13, increasing by 4 each time. The static,1 at line 10 is responsible for this. After commenting the following code back into the program

printf("\n---\n\n");

#pragma omp parallel

{

int id = omp_get_thread_num();

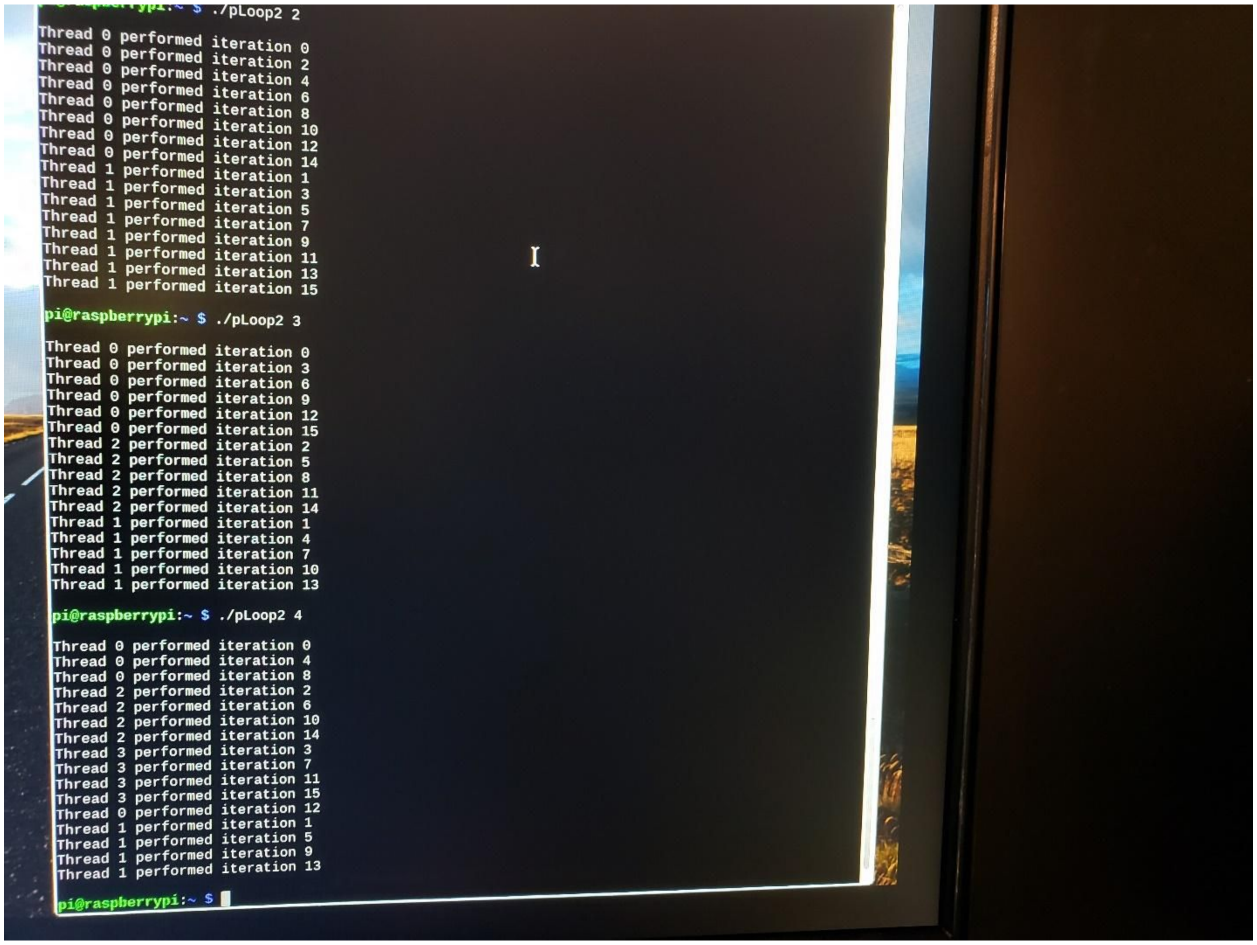
int numThreads = omp_get_num_threads();

for (int i = id; i < REPS; i += numThreads) {

printf("Thread %d performed iteration %dn" a.id, i);}

}

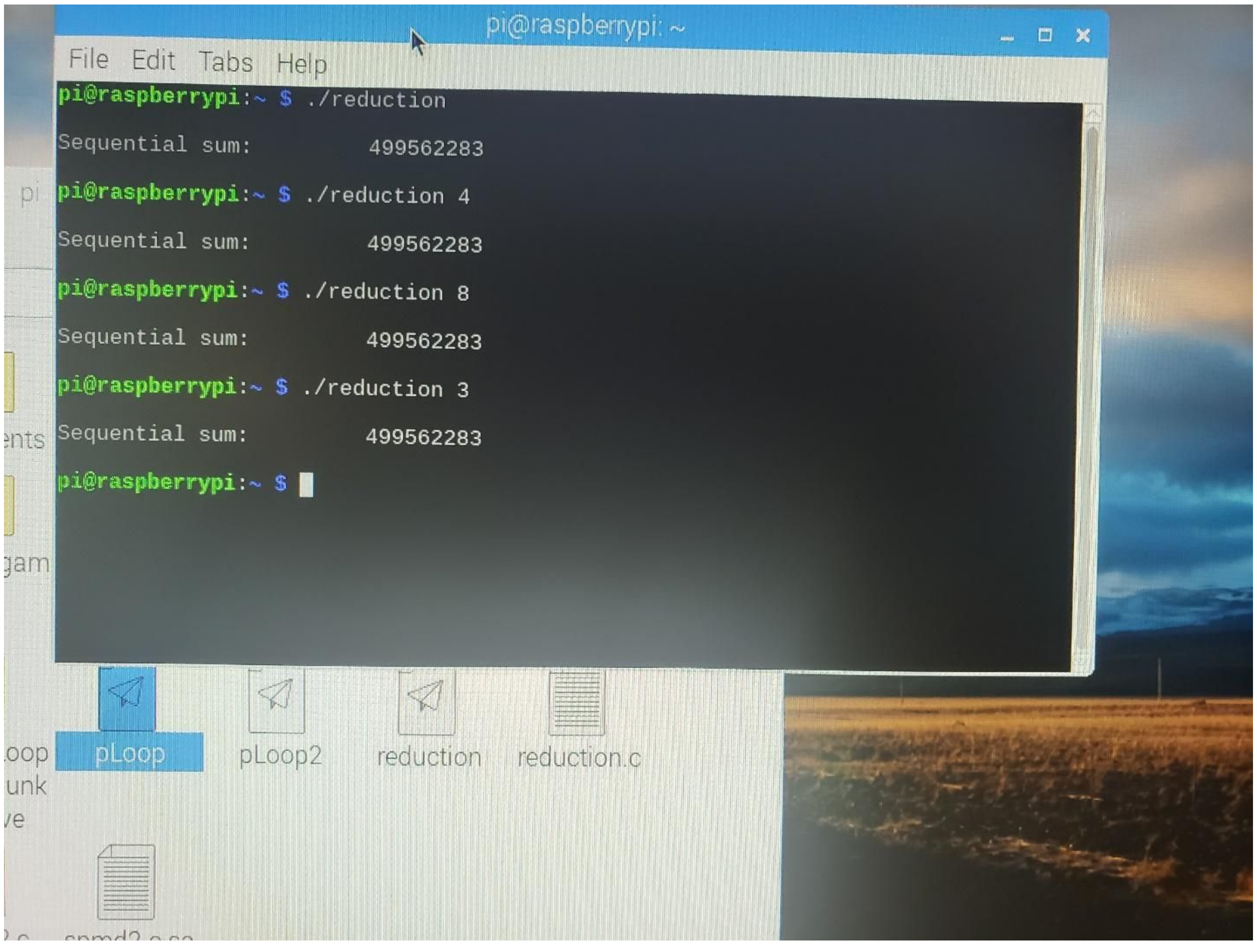
the output is the same as the original output.



```
ypi:~ $ ./pLoop2 2

Thread 0 performed iteration 0
Thread 0 performed iteration 2
Thread 0 performed iteration 4
Thread 0 performed iteration 6
Thread 0 performed iteration 8
Thread 0 performed iteration 10
Thread 0 performed iteration 12
Thread 0 performed iteration 14
Thread 1 performed iteration 1
Thread 1 performed iteration 3
Thread 1 performed iteration 5
Thread 1 performed iteration 7
Thread 1 performed iteration 9
Thread 1 performed iteration 11
Thread 1 performed iteration 13
Thread 1 performed iteration 15

pi@raspberrypi:~ $ ./pLoop2 3

Thread 0 performed iteration 0
Thread 0 performed iteration 3
Thread 0 performed iteration 6
Thread 0 performed iteration 9
Thread 0 performed iteration 12
Thread 0 performed iteration 15
Thread 2 performed iteration 2
Thread 2 performed iteration 5
Thread 2 performed iteration 8
Thread 2 performed iteration 11
Thread 2 performed iteration 14
Thread 1 performed iteration 1
Thread 1 performed iteration 4
Thread 1 performed iteration 7
Thread 1 performed iteration 10
Thread 1 performed iteration 13

pi@raspberrypi:~ $ ./pLoop2 4

Thread 0 performed iteration 0
Thread 0 performed iteration 4
Thread 0 performed iteration 8
Thread 2 performed iteration 2
Thread 2 performed iteration 6
Thread 2 performed iteration 10
Thread 2 performed iteration 14
Thread 3 performed iteration 3
Thread 3 performed iteration 7
Thread 3 performed iteration 11
Thread 3 performed iteration 15
Thread 0 performed iteration 12
Thread 1 performed iteration 1
Thread 1 performed iteration 5
Thread 1 performed iteration 9
Thread 1 performed iteration 13

pi@raspberrypi:~ $
```

Figure 2: Code running of pLoop2

In the program reduction the output of the sum is 499562283. This is always the output no matter what number if put after reduction. The number after only determines the number of threads that will be doing the work. After uncommenting **#pragma omp parallel for** on line 39 the result is the same. Then after uncommenting **reduction(+:sum)** on the same line the end result is still the same as it was before.



```
pi@raspberrypi:~ $ ./reduction
Sequential sum:          499562283
pi@raspberrypi:~ $ ./reduction 4
Sequential sum:          499562283
pi@raspberrypi:~ $ ./reduction 8
Sequential sum:          499562283
pi@raspberrypi:~ $ ./reduction 3
Sequential sum:          499562283
pi@raspberrypi:~ $
```

Figure 3: Code running of reduction