Project A3

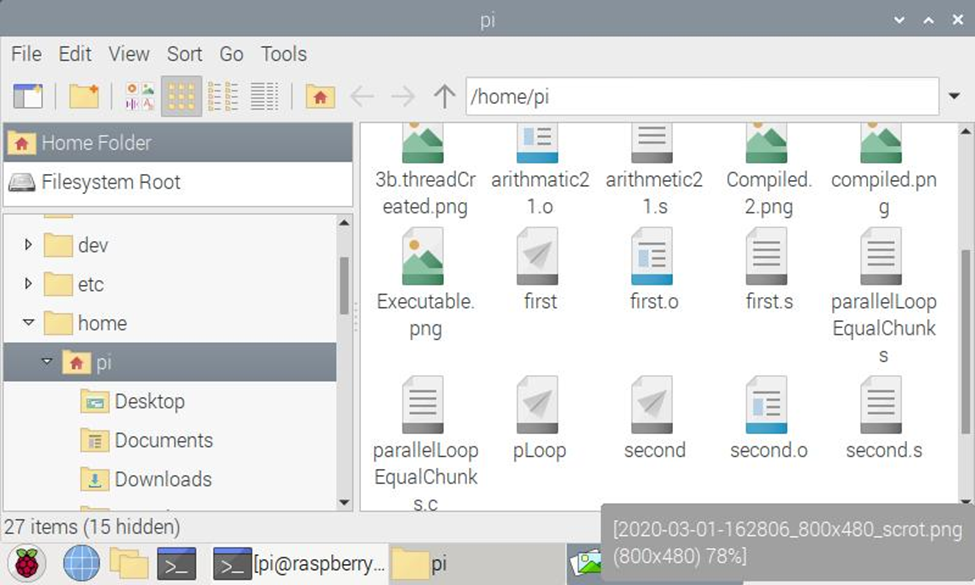
Tek Acharya

900758663

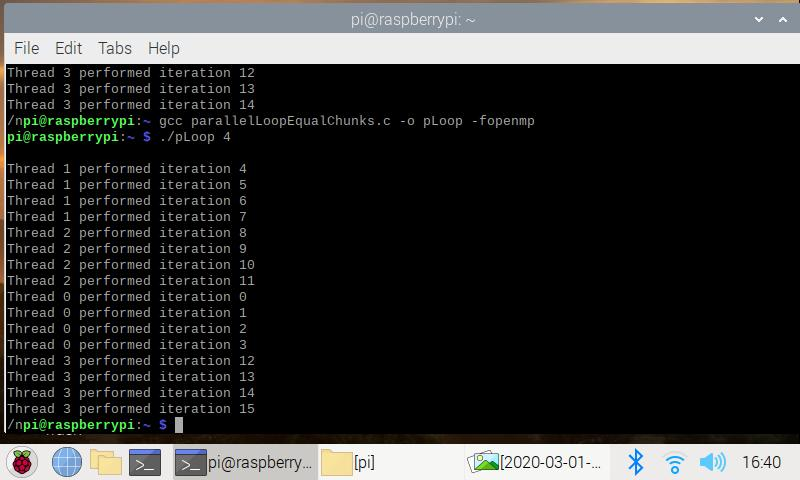
GSU-Spring 2020

**TASK3b. Parallel Programming Basics A3(80P)**

What happens when the number of iterations (16 in this code) is not divisible by the number of threads?

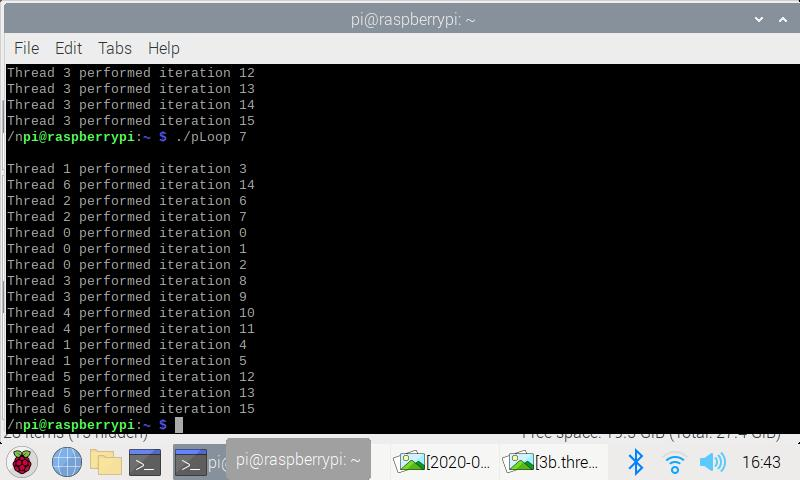


The executable pLoop file is created.

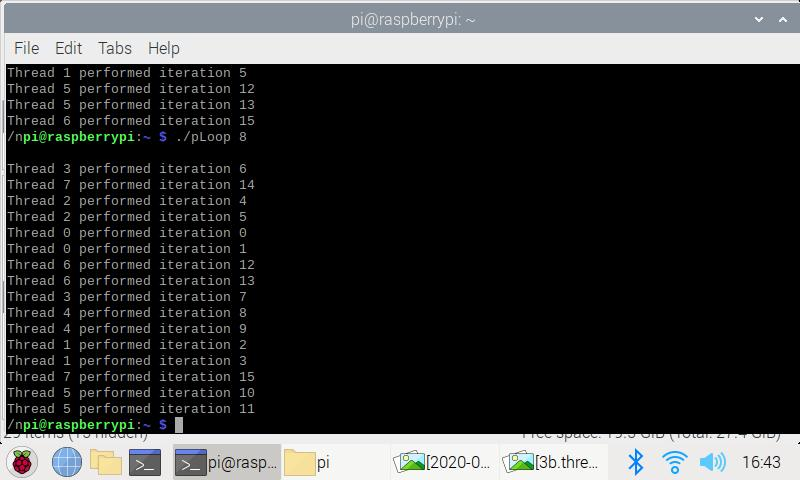


4 threads created and observed that 0 id performed iteration 0, 1, 2, and 3. Id 1 performed iteration 4, 5, 6, and 7, id 2 performed iteration 8, 9, 10, and 11. Similarly id 3 performed iteration 12, 13, 14, and 15. This shows that each thread id is involved 4 in iterating exactly 4 iterations. Also noticed that lower id number is used for initial iterations. And higher id numbers for the later iterations.

Also, the pattern of thread used per iteration has as given has been verified through code..



Tried several other numbers of threads to see a pattern of iteration assignments and noted that initial iterations are performed by lower threads and later ones by higher thread ids.

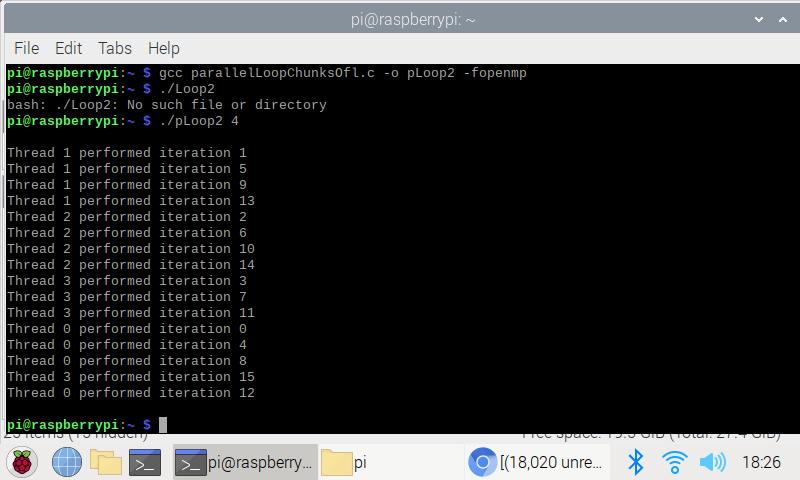


When the number of iterations is not evenly divisible by the number of threads, the even pattern of thread created is no longer possible which is shown in the above screenshots.

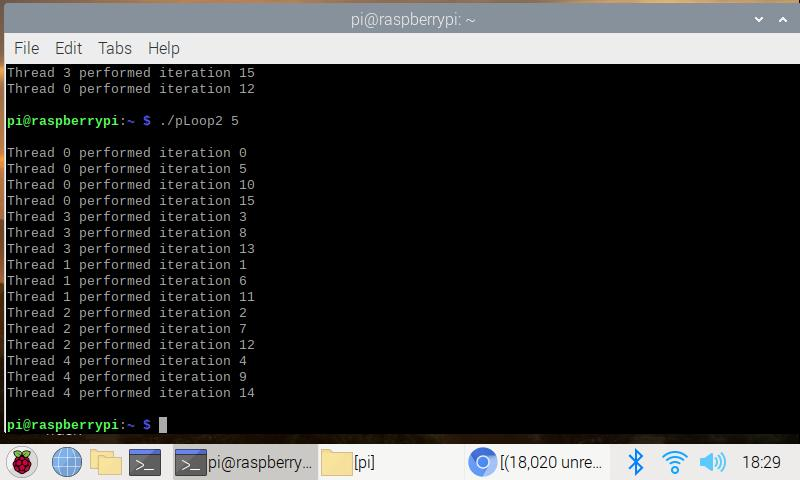
Thread Scheduling

In the first part of the code, we have used static scheduling system in which we set pragma to set schedule statically. We compared the thread performed iteration with the previous version and found no difference compared with this version except a different approach of thread implementation through scheduling.

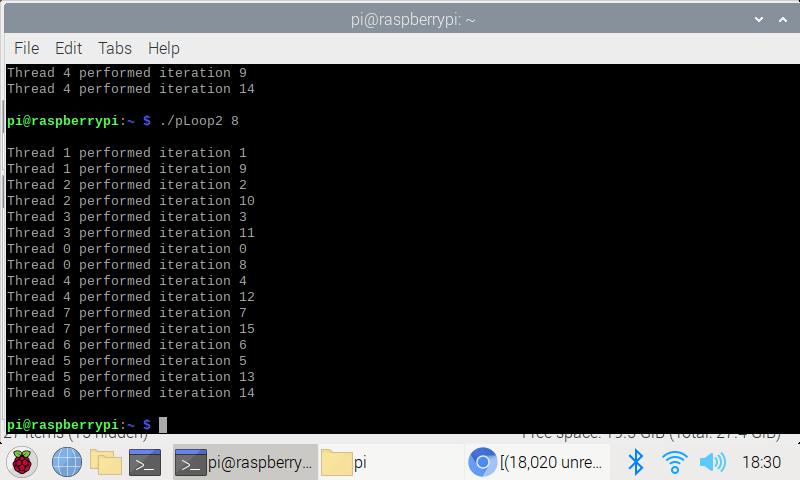
Next, we uncomment the section of code provided and study the iterations performed by the threads. We then use different cores to create threads and observed the iterations performed. We also noted that this time, the static version of the code is taken off from the pragma. Here we noticed that the pattern of output is the same, however, the output of iteration id is not what we input but any random ids.



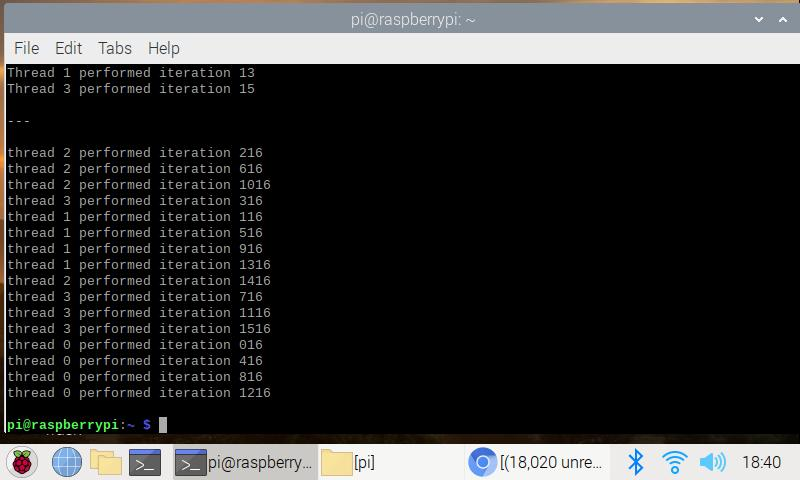
The pattern of thread creation and distribution of iteration formation is quite different to that without using static version.



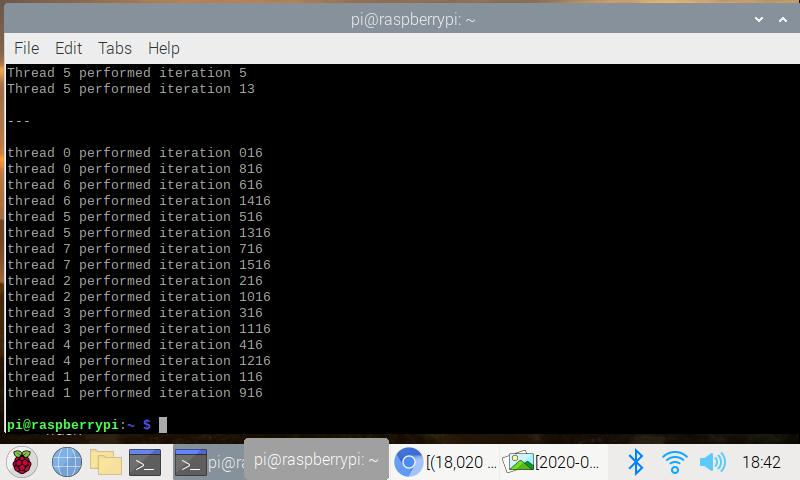
Observed the pattern using different cores



Observed using even multiple of 4



Output after the comment is uncommented and compared the output back-to-back



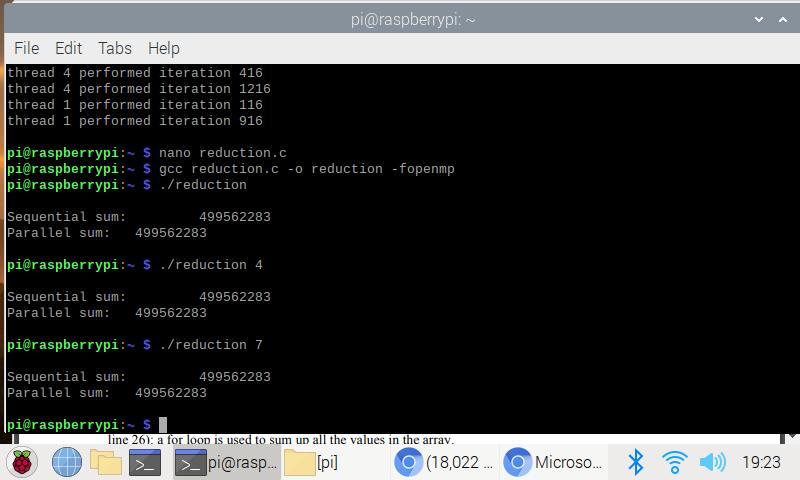
The output is seen the same except the iteration id.

**When loops has dependencies**

**3.4 Dynamic scheduling**

Compiled and ran the code as is (without uncommenting the comment) and looked into the output using different number of cores.

Observed that each time it gives the same answer as shown below.



The output is the same regardless of number of threads created. This is because the parallel computation has not taken into effect, yet.

Show and explain the output after running the program.

The sequential and parallel sum came up to be the same. However, we have not used the parallel computation yet as we have commented the pragma

1. To actually run it in parallel, per instruction, we uncommented the first comment and investigated the output.

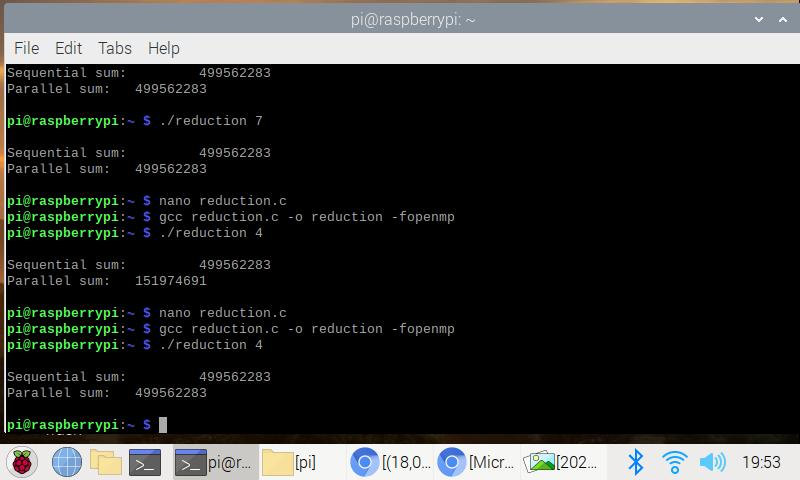
In this case we get different answers for the:

sequential sum (49562283), and

parallel sum (151974691)

1. This time we removed the second comment and let the parallel accumulate the total sum coming from all cores.

This time the answer on both the summation methods matched as shown in the screenshot



Code uncommented and investigated and each time output is studied.