Lab Session Week 3

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student Name** | **Student Email** |
| 32844700 | Teh Jia Xuan | Jteh0015@student.monash.edu |

Task 1 - Code

A screen shot of a computer program

Description automatically generated

A computer screen with text

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A black screen with white text

Description automatically generated

Serial code took 0.533 seconds to run

Task 2 - Code

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of code

Description automatically generated

A black screen with white text

Description automatically generated

Task 2 – Explanation

A white background with black text

Description automatically generated

My serial code run 0.533 seconds and my parallel code run 0.442950 seconds. Which is speed up by 16.88%, 0.09 seconds. For my parallel code I created 8 threads to speed up as I have 8 cores in my CPU. In my expectation, the code is supposed to speed up 8 times faster as I am using 8 threads compared to using 1 thread in my serial code.

1. I/O operations

The reason it is not meeting the expectations is the code performs a lot of I/O operations like printing to the console which can be a bottleneck as printing to console is slow and it not easily be parallelised. Even with multiple threads the console only handles the output operations once at a time.

1. Synchronization overhead

Another reason is each time a thread needs to update the array when they found a prime number. So each time when a thread needs to access one shared memory, it must lock the shared memory to prevent other threads update the data and cause data corruption or confusion. After finish updating it will unlock the shared memory for other threads to access. Thus, the cost of locking and unlocking will significantly reduce the expected speed up as other threads need to wait for the shared memory to be unlocked.

1. Workload imbalanced

Besides that, the workload is not equally divided among the threads some threads might finish their tasks earlier and remain idle while others are still working. In my implementation, I have divided tasks to the threads for example, thread 1 calculating prime number 0 – 250000, thread 2 calculating 250001 – 500000. So thread 1 will be faster than thread 2 as its data can be calculated faster. So thread 1 will remain idle.

Task 3 – Code

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Task 3 - Explanation

Task 3 I have used openMP implementations to parallelized the code and it speeds up by 0.26 seconds. The threads is set to 8 to make the program run as fast as possible as there is 8 cores in my machine.

The expected speed up is supposed to be 8 times faster than the serial one. But due to several reasons it does not meet with the actual time.

1. Synchronization Overhead

When the thread is updating the array we need to lock the shared memory to prevent race condition. In other words, this is to prevent conflicts between two threads trying to write data to the same location. Thus, we lock the shared memory, but locking the shared memory will cause threads to wait for it to be unlock. Hence, reduces the speed. Although in my implementation, I have declared each local array for each thread, so they can update in their own array without locking and unlocking. But after the calculation they need to write to the global array so it still introduce synchronization overhead.

1. Memory bandwidth limitation

When multiple threads are accessing data from RAM the bandwidth is not sufficient to support the CPU’s processing speed. So CPU may spent more time to wait for the data and cause it to be slow.