CENG444 – HW1

Threads can be utilized for parallel computations. In this homework, I tried to calculate π by implementing **Monte Carlo** method into a function used by threads. Threads were distributed and executed by multicores (4 core in my case) and execution time was recorded for execution of each number of threads.

First of all, I prefer to use 10 million for number of toss. Although, I thought to use much bigger of it, I could not use due to segmentation faults. The reason behind the segmentation faults is that I created double arrays for both x and y which have the number of toss elements to store indexes for tosses. Using $2*10^6*8$ bytes corresponds to 16GB for one thread. For increasing number of threads, it results in bigger memory allocations from the memory, if double arrays are fully utilized. Even, since I cannot that much memory with static memory allocation. Therefore, I prefer to use dynamic memory allocation method. As a result, I used 10 million tosses for the π prediction.

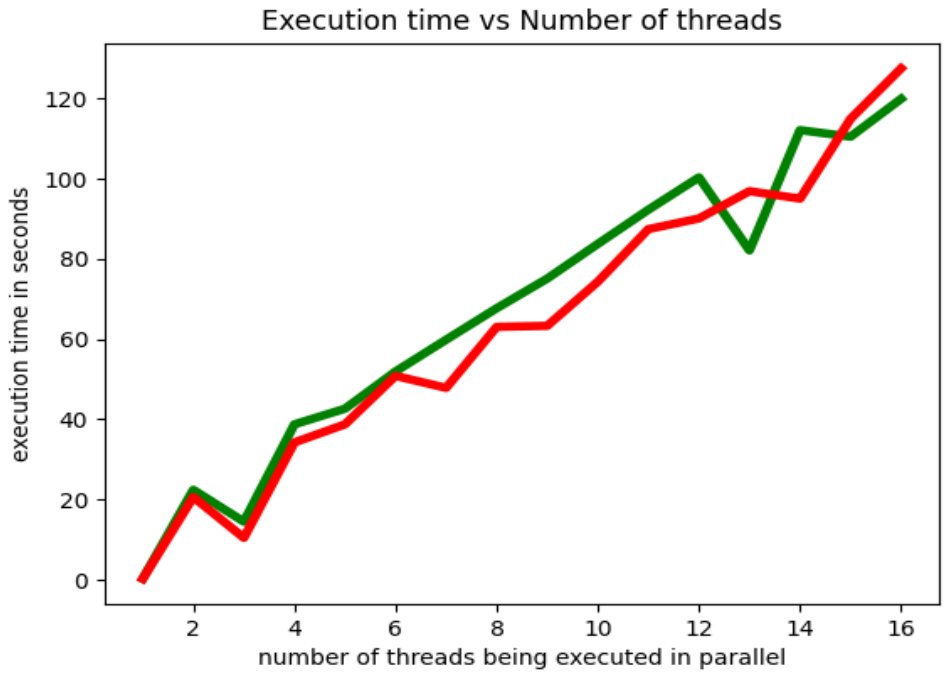

Figure 1: Execution time of 2 different runs. Each of these runs, executed with from 1 to 16 threads.

In the above figure 1, there are two results for this experiment. Before interpreting the results, I share the instructions below for both compiling and executions.

- **To compile**: gcc -o test -fopenmp 283078027.c

Gcc is the compiler used to compile the 283078027.c code. -fopenmp is used to enable openMP. test specifies executable output file by –o.

- **To execute**:./test

  By writing ./test, generated test executable is executed by the computer.

By using the threads which can be executed as parallel, it is expected that execution time does not increase linearly as increasing number of threads. For example, 4 threads will be executed into 4 cores instead of sequential execution, and execution performance will increase meaning that decreasing of execution time.

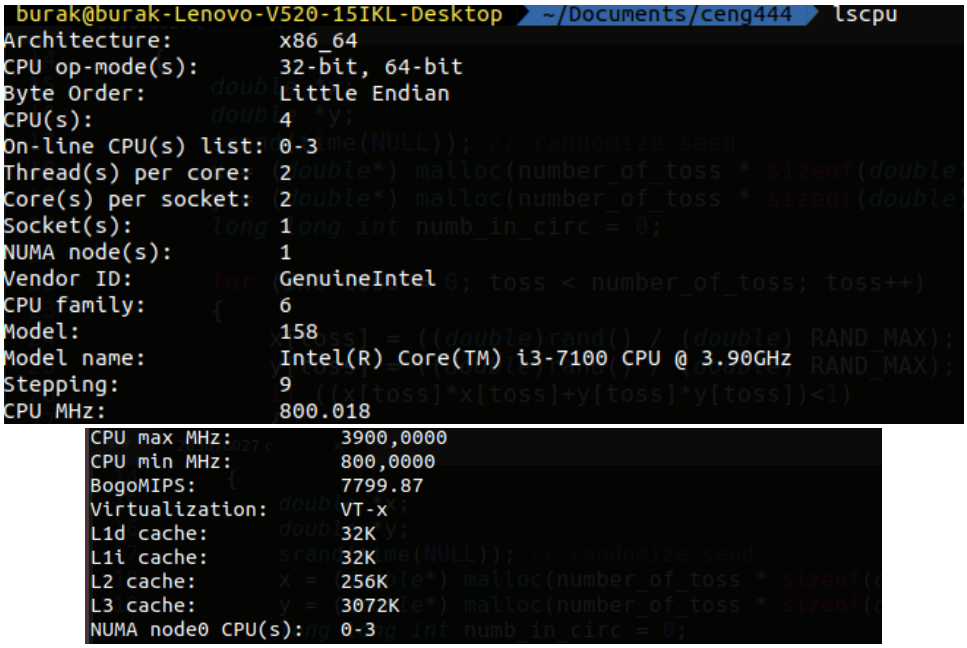

Figure 2: Computer hardware properties such as number of CPUs, memory subunits and so on.

In the figure 2, my computer properties are shared. There are 4 CPUs, 3 level caches and the first level cache is not uniform such that one of them is instruction cache while the other is data cache. For our problem, since we are storing huge memory arrays for indexing the toss; that is, each of the threads must read/write their data into memory. In other words, our problem has some memory boundary such that each CPU has to read/write their data into memory in very close times. By the way, each CPU can handle 2 threads. This means that there is no sequential increase for the execution time up to 8 threads.

Our CPUs operates with respect to RISC ISA so, one line of the code is executed in multiple of assembly instruction. For instance, '**x[toss] = ((double)rand() / (double) RAND_MAX) '** will be executed by multiple assembly instructions such that, random number will be generated, converted into double format, then divided by double converted RAND_MAX constant.

Afterwards, it will be stored into x array indexed by toss value. Thus, there are multiple and mixed instructions which will be executed in the sequentially for each core.

As a result, even if we expect that compute time to be very similar for executing 3-4 threads or 6-8 threads, since each of them has their memory overloads, the execution time increases. However, there is not proportional relationship between the number of threads and execution time for increased number of threads. For example, execution time is not doubled from 4 to 8 threads. However, the time amount of increase for execution with increasing number of threads is caused by initialization of the threads such as allocating memories, initialize the variables. Since computation will be operated in a parallel manner for multiple threads, the relationship between execution time and number of threads is not proportional. It can be observed from graph of figure 1 such that slope of the graphs is decreasing for increasing number of threads.

APPENDICES

Execution times for 2 runs of using multiple threads to calculate pi estimation by using Monte Carlo method.