CENG444 - Programming Assignment #2

In this homework, we are expected to implement a function to solve maximum consecutive sum problem. In this problem, we try to capture the sub-array where the sum of it is the largest for the given 1D array.

Firstly, I create an 1D array that has 320k randomly generated integer elements range from -15 to 15. After generating 1D input array, below steps are implemented:

- I calculate '**s**' array where s consists of the prefix-sums of a given input array.
- I calculate **'m'** array where m consists of prefix-minimums of the **s** array.
- Then, I calculate the **'s_m'** array with the help of s_m[j] = s[j] - m [j-1] formula.
- At last, I calculate the **'index'** array with respect to index[j] = max{i: 0 <=i <= j and s[i]=m[j]}

After calculating above arrays, **MCS** is the maximum element of the s_m array. Also, it is required to identify the interval of the maximum consecutive sum of the sub-array that belongs to the given input 1D array. Range of the sub-array is calculated as lower limit is max(index)+1 and upper limit is the minimum index of s_m array which holds the MCS value. In this way, both MCS value and corresponding subarray interval can be found.

To execute the program, I have compiled the c code with '**gcc 283078027.c -fopenmp 283078027**' command and run the resultant 283078027 executable by writing '**./283078027**' command. I enabled the omp library with –fopenmp statement.

I have used 320k for the input array size. Also, s, m, s_m, and index have the 320k integer elements. To create each of these arrays, we need to use 320k iteration. In order to see the effect of parallelization by introducing threads, I have recorded the run time of whole program for varying number of threads from 1 to 16 as shown in figure 1.

My computer properties are shared in the appendix section. In my computer, there are 12 cores and each of my cores can compute 2 threads. I have also 192kB L1 instruction and data caches, 3MB L2 unified and 8MB L3 unified caches. Since each of my arrays is integer arrays, each of the arrays (input, s, m, s_m, index) requires 320k*4byte memory. As far as I understood, each thread fetches its own required data to the shared memory and uses them respectively. In other words, spatial locality is used not to fetch the neighbor data of each array for each consecutive iteration. Also, since I have sequentially implemented the calculation of each intermediate arrays (s, s_m, m, index), the memory that is closure to the CPU can load data and CPU can use them from at least 3rd level of cache (320k*4byte = 1,28MB for each array)

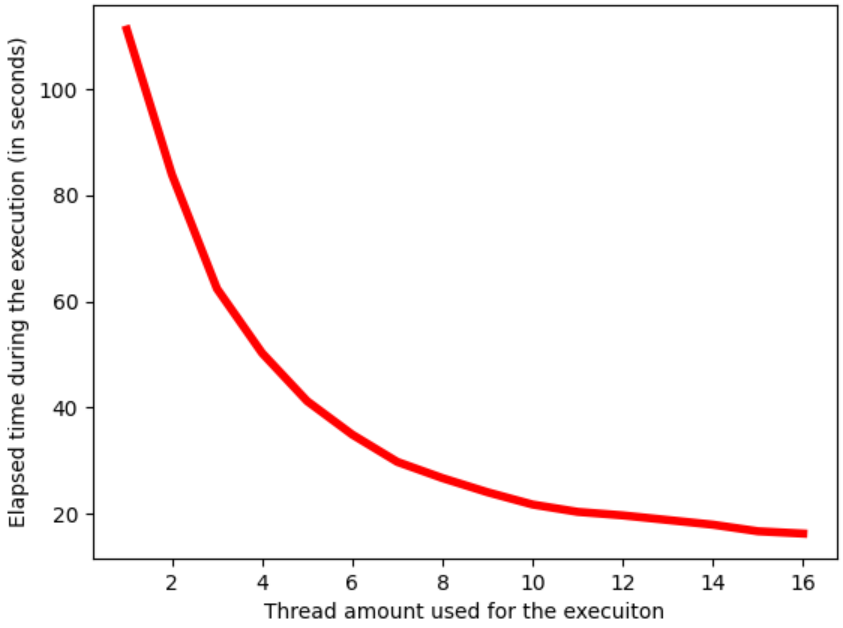

Figure 1: Elapsed time during the execution of the program with varying number of threads (Timings are shared in the detailed form in appendix part)

By looking at the results given in figure 1, the elapsed time during the execution converges to 10 seconds when I increase the number of threads from 1 to 16. It is expected since each core is activated to carry out similar job of for MCS problem when I increase the number of threads. Although execution time is not decreasing linearly for increasing number of threads, 16 threads execute 10 times faster compared to 1 thread. The reason behind nonlinear decrease (kind of exponential decrease) is that each thread has its own overhead such that there has to be some private elements for each thread or synchronization overhead to communication between the threads. These additive elements for each thread such as private program counter, stack pointer and stack region, creates more instruction while context switch cases where change is occurred between activated thread and pending thread. In addition, when 12 threads are activated to find MCS and its corresponding subarray from the given array, each core executes one thread actively. Furthermore, I did not lock up any of the threads for mutual exclusion since my implementation did not require lock mechanism for shared data. Also, I used static scheduling that distributes workload of the iterations to each thread uniformly.

As a result, I can reach 10 times faster execution speed with respect to sequential execution by enabling 16 thread and parallelizing the calculation of each array. Although I could not observe 16 times faster execution, 10 times is also significant speed up rate. The reason why linear speed up cannot be observed is that there are some bottlenecks caused by overheads of the threads and system limitations such as limited cores and memory bandwidth.

APPENDIX

```
burak@burakslaptorp:~/Documents/ceng444/hw2$ gcc hw2.c -fopenmp -o main
burak@burakslaptorp:~/Documents/ceng444/hw2$ ./main
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 1 threads is 111.201 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 2 threads is 83.902 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 3 threads is 62.393 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 4 threads is 50.270 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 5 threads is 41.209 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 6 threads is 34.908 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 7 threads is 29.762 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 8 threads is 26.712 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 9 threads is 24.049 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 10 threads is 21.730 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 11 threads is 20.350 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 12 threads is 19.686 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 13 threads is 18.814 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 14 threads is 17.932 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 15 threads is 16.693 seconds
MCS is 635 and the consecutive sub-array is [165639, 166214]
The elapsed time to run with 16 threads is 16.257 seconds
```

```
burak@burakslaptorp:~/Documents/ceng444/hw2$ python3 plots.py
burak@burakslaptorp:~/Documents/ceng444/hw2$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   48 bits physical, 48 bits virtual
CPU(s):                          12
On-line CPU(s) list:             0-11
Thread(s) per core:              2
Core(s) per socket:              6
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           96
Model name:                      AMD Ryzen 5 4600H with Radeon Graphics
Stepping:                        1
Frequency boost:                 enabled
CPU MHz:                         1507.607
CPU max MHz:                     3000,0000
CPU min MHz:                     1400,0000
BogoMIPS:                        5988.92
Virtualization:                  AMD-V
L1d cache:                       192 KiB
L1i cache:                       192 KiB
L2 cache:                        3 MiB
L3 cache:                        8 MiB
NUMA node0 CPU(s):               0-11
```