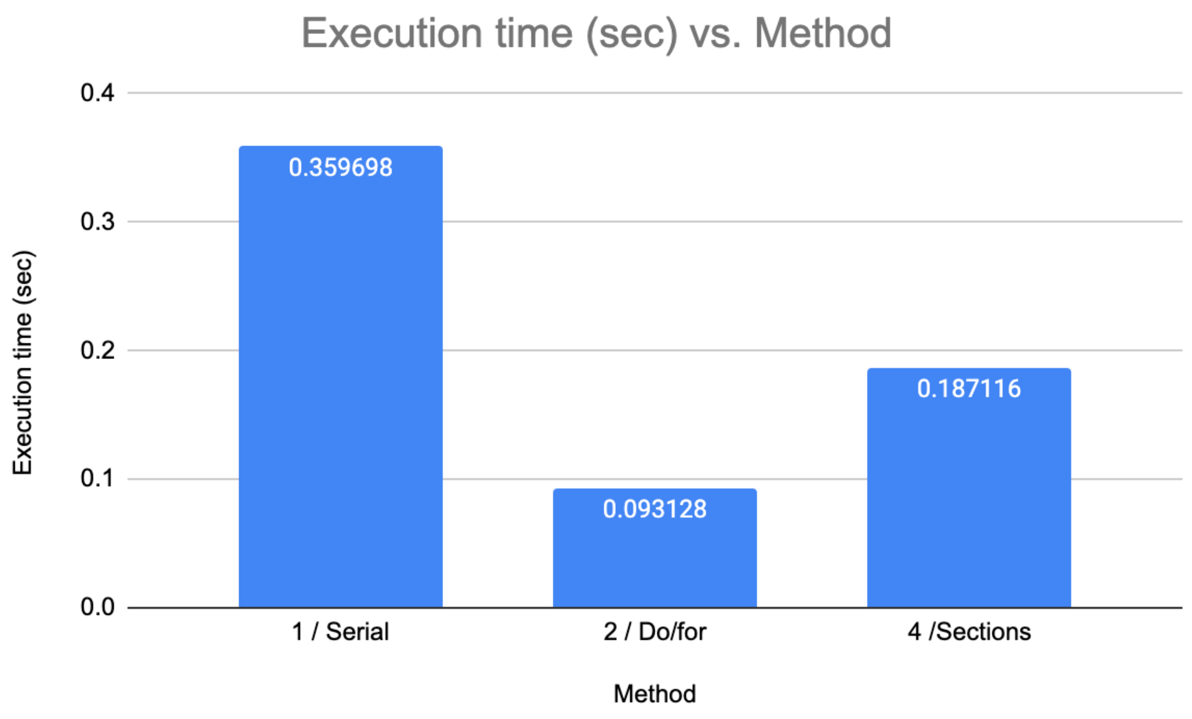
Tingqi (Ting) Wang

## Problem 1: Estimating $\pi$

The following is the results of running a serial, and two OpenMP parallelized programs to estimate the value of pi. The program uses 40000 points to estimate the value. One of the OpenMP parallelized programs uses the DO/for directive (4 threads) and the other uses the Sections directive (2 threads).

The figures below (graph and table) illustrate the corresponding execution times in seconds for serial and two types of OpenMP parallelized execution. The runtime was the highest for the serial execution, and lowest for the Do/for directory parallelized execution. The reason that the Do/for directory execution was lower than the Sections directory execution is because the latter requires additional partitioning of the for loop to allow for parallelism. The Do/for directory also divides the for loop iterations amongst the available threads to allow for better load balancing, thus improving runtime.



| Method | Execution time (sec) |
|---|---|
| 1 / Serial | 0.359698 |
| 2 / Do/for | 0.093128 |
| 4 /Sections | 0.187116 |

Tingqi (Ting) Wang

Terminal log:

```
twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p1a
chunks: 10000000
number of points: 31424016
Estimated pi is 3.142402, execution time = 0.093128 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p1b
number of points: 31424016
Estimated pi is 3.142402, execution time = 0.187116 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p1_serial
Estimated pi is 3.142402, execution time = 0.359698 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$
```

## Problem 2: Sorting

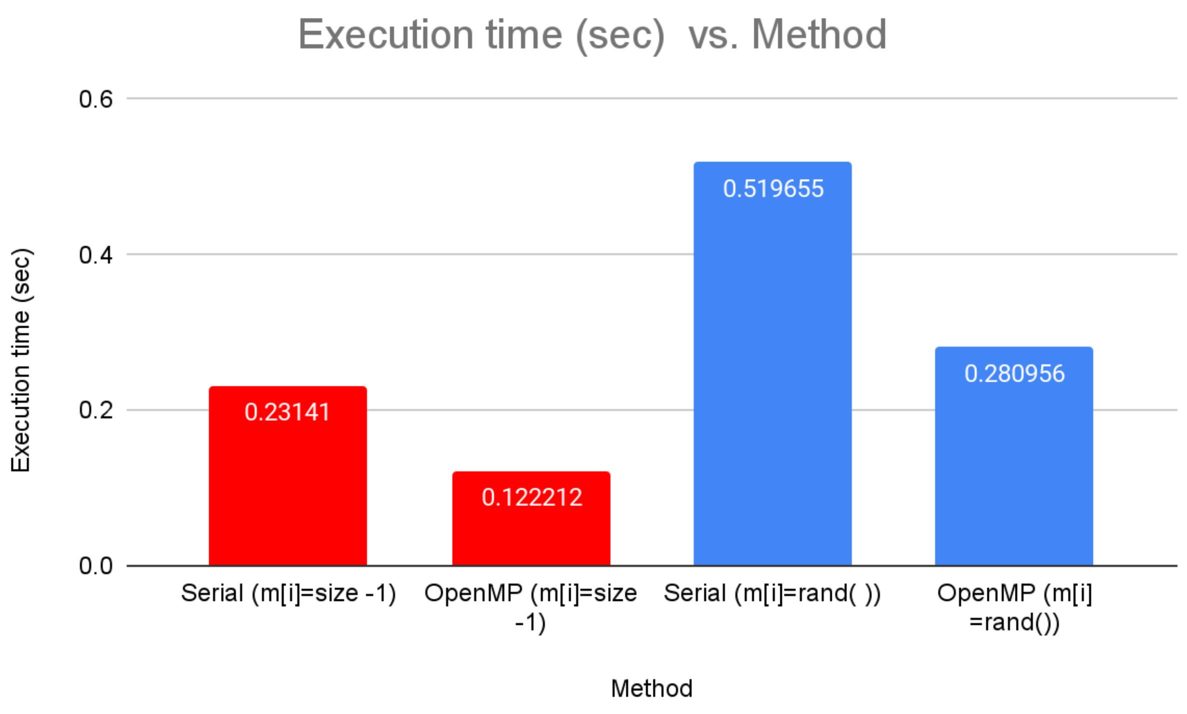
The following is the result of running a program in serial and in parallel (parallelized using OpenMP with 2 threads) on a 2M size array populated using m[i] = size −i or m[i] = rand().

The figures below (graph and table) illustrate the corresponding execution times in seconds for serial and parallel execution. The red bars correspond to the m[i] = size −i instances, whereas the blue corresponds to the m[i] = rand() instances. Looking at the graph, it can be seen that the runtime of the non random instances (red) were lower than the random instances (blue) for both methods. For both instances, the runtime for the OpenMP parallelized method was lower than that of the serial method.

Explanation

The reason for the overall higher runtime for the m[i] = rand() instances is potentially due to the lack of predictability in the data. m[i] = size −i means that all data is in descending order, which could be beneficial in boosting the cache hit rate, which would largely reduce the runtime through decreasing memory access time. On the other hand, for the random instance, the randomness is likely to cause high cache misses, which means that there is a higher runtime.

Tingqi (Ting) Wang



| Method | Execution time (sec) |
|---|---|
| Serial (m[i]=size -1) | 0.23141 |
| OpenMP (m[i]=size -1) | 0.122212 |
| Serial (m[i]=rand( )) | 0.519655 |
| OpenMP (m[i]=rand()) | 0.280956 |

Terminal Log:

```
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p2
1803 2582 2807 2837 4493 4537 4643 5006 7405 8217 8814 10616 10717
11640 12560 14422
Execution time = 0.280956 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p2_serial
270 481 4568 8856 9857 10835 11774 12745 13089 14425 14882 17518
18466 18743 20198 21427
Execution time = 0.519655 sec

[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p2
```

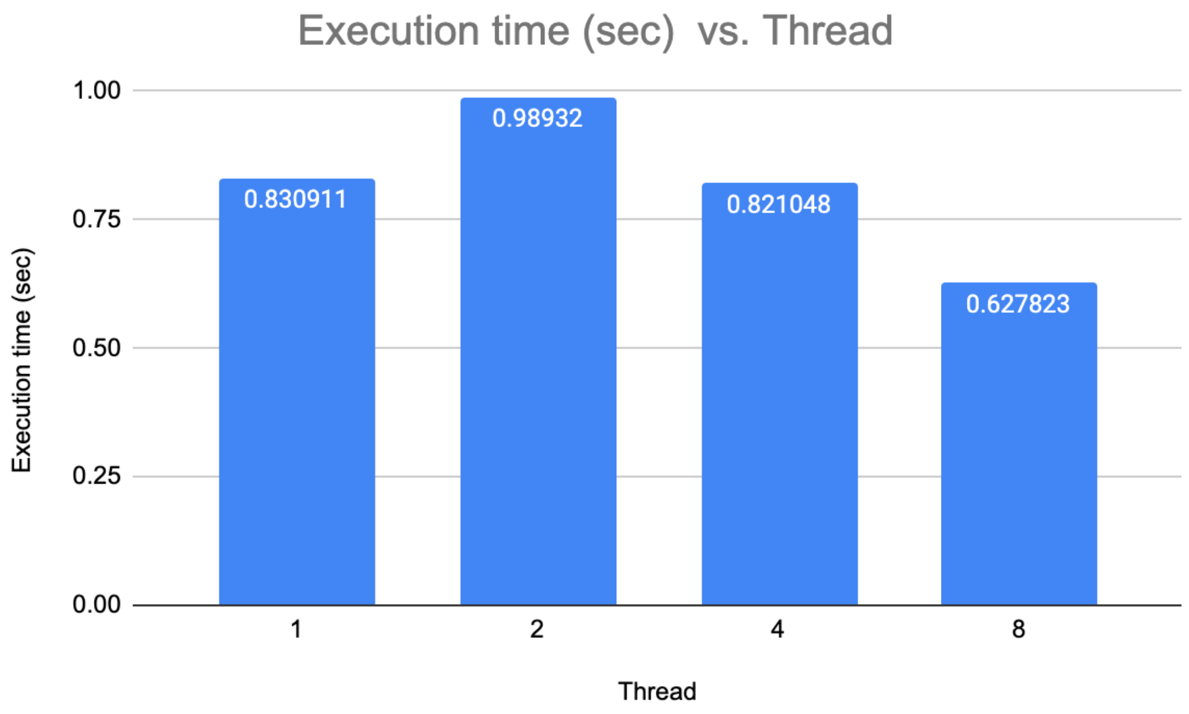
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Execution time = 0.122212 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p2_serial
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Execution time = 0.231410 sec
```

## Problem 3: Parallel K-Means

The following is the result of running a parallel K-means program that uses mutexes and conditions to maintain synchronization to reduce iteratively creating and joining threads. The program takes in 800*800 input.raw and produces a 800*800 output.raw. The program does not test for convergence, instead iterates 50 times then finishes. The program also runs on the following number of threads: 1, 2, 4, 8.

The figures below (graph and table) illustrate the corresponding execution times in seconds for each number of threads. Looking at the graph, it can be seen that the runtime of the program first increases with the increase of threads from 1 to 2. After the increase however, the runtime decreases linearly with respect to the increase of the number of threads. The expansion for the initial increase may come from the increased overhead of adding an additional thread, such that the acceleration from the additional thread is less than the increase in overhead to manage 2 threads.

Tingqi (Ting) Wang



| Thread | Execution time (sec) |
|---:|---:|
| 1 | 0.830911 |
| 2 | 0.98932 |
| 4 | 0.821048 |
| 8 | 0.627823 |

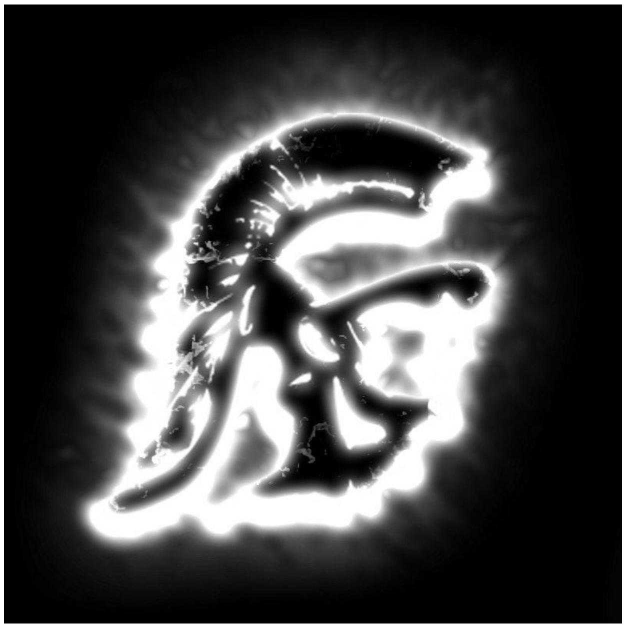
<u>Results from PHW2</u>
The runtime of the original serial program was 0.286369 sec
The runtime of the parallelized program with 4 threads was 0.237486 sec
The runtime of the parallelized program with 8 threads was 0.165518 sec

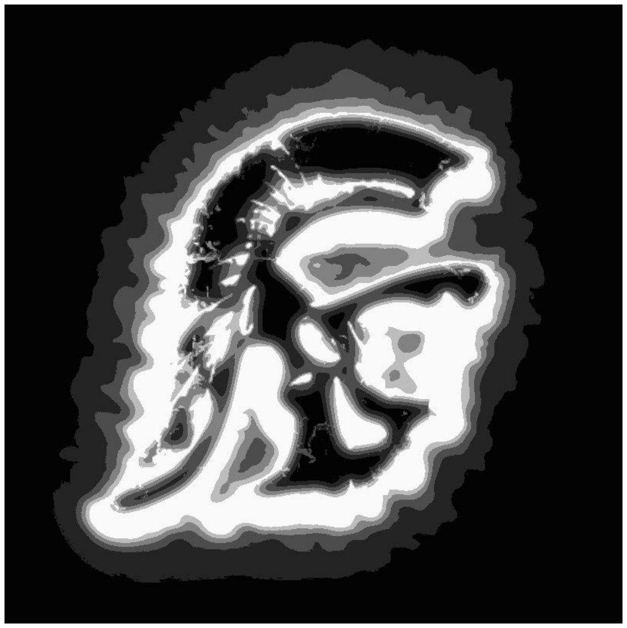
<u>Comparison</u>

The runtime for the PHW2 program was less than the runtime for the PHW3 program on the whole. However, as threads increased to 4 and 8 for both programs, the runtime decreased linearly. One explanation for the difference in runtime is that PHW2 and PHW3 were run on different test platforms (local windows vs. CARC). In addition, the method adopted in PHW3 also meant that there were increases in overhead time in using mutexes and conditions that may cause increased runtime.

Tingqi (Ting) Wang

The images below demonstrate the images generated using imageJ from input.raw (left) and output.raw (right) files.





Terminal Log:

```
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p3 8
Time taken to run: 0.627823 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p3 4
Time taken to run: 0.821048 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p3 2
Time taken to run: 0.989320 sec
[twang356@d05-33 EE_451_S_2023_PHW_3]$ ./p3 1
Time taken to run: 0.830911 sec
```