Parallel Programming Assignment 2

Performance profile of synchronization constructs

Name: Ishan Gadgil

Roll Number: 111601007

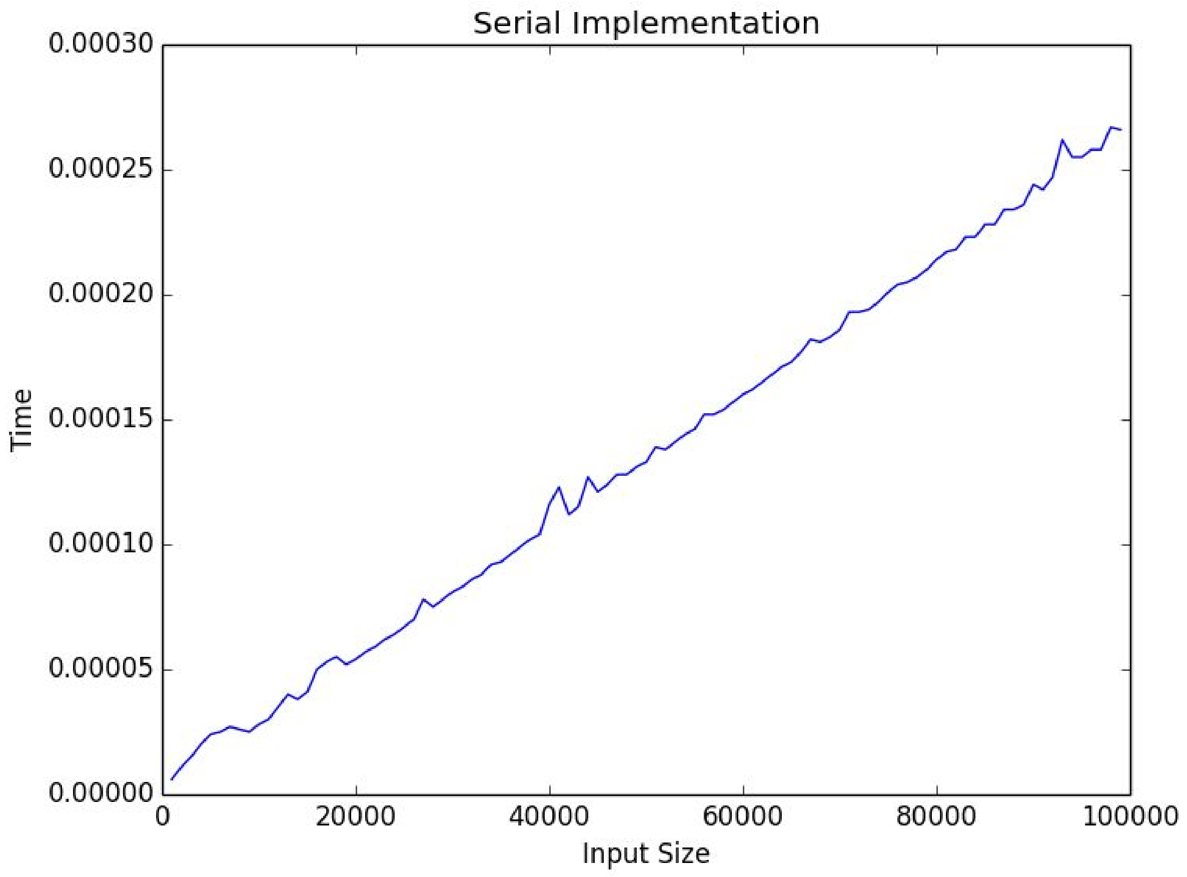
Problem Statement:

Design experiments (or test cases), execute them and measure the performance of various synchronization constructs such as busy-waiting, mutexes, conditional variables, barriers, and read-write locks.

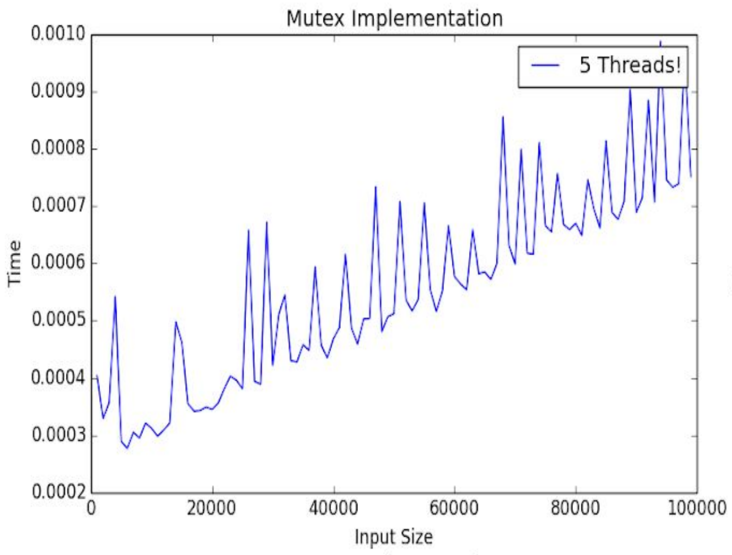
Experiment 1: Sum of n integers

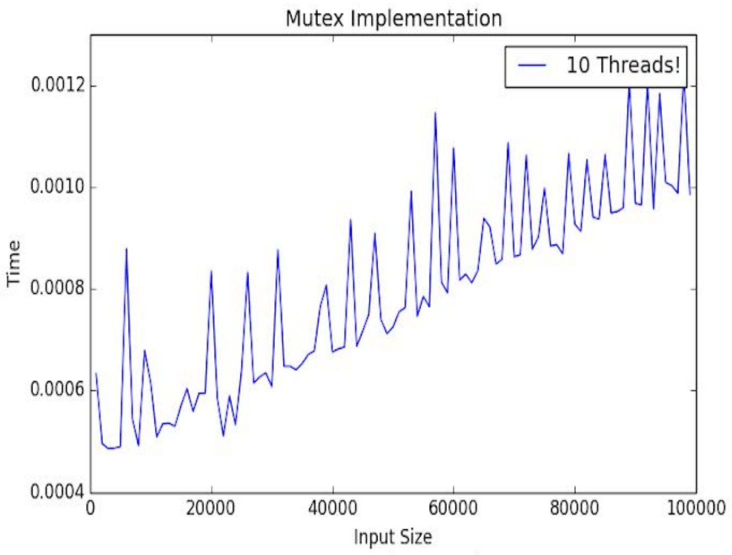
a) Serial Implementation

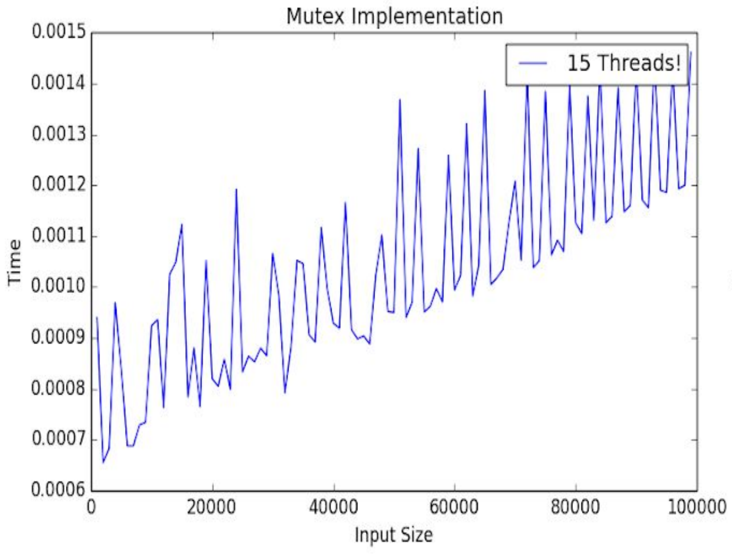
Designed serial implementation of the problem.
Varied the input size from 1000 to 100,000 with increments of 1000. Also, for each input size, taken the average for 10 iterations.

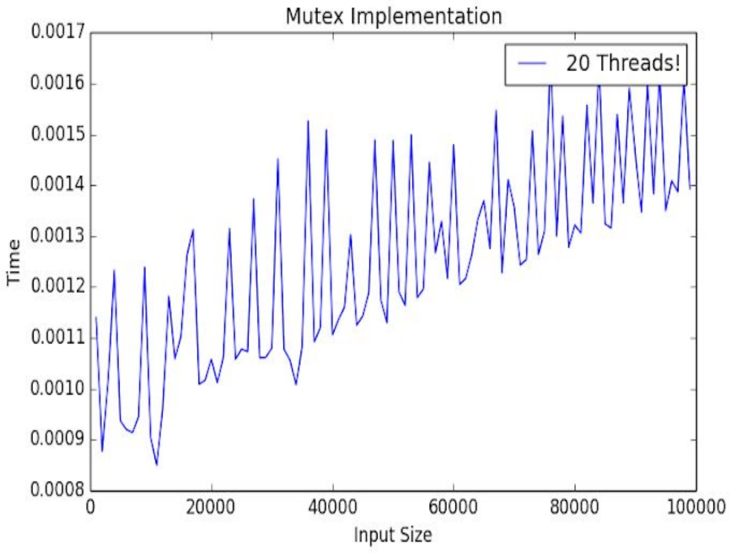


b) Mutex Implementation
Varying the number of threads from 5 to 20 with increments of 5.



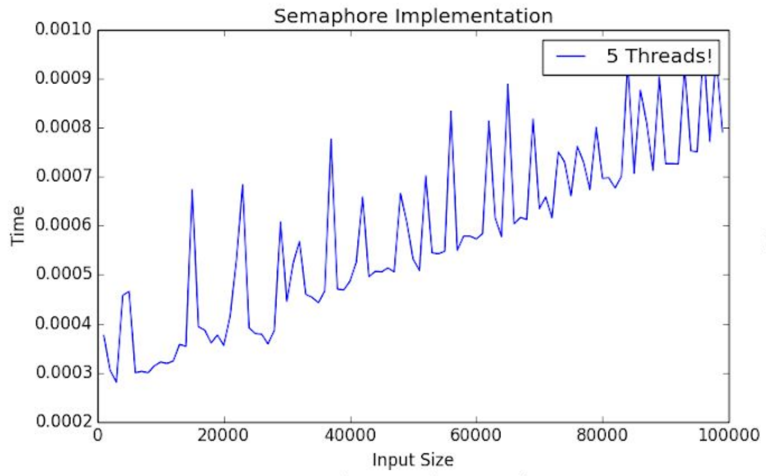


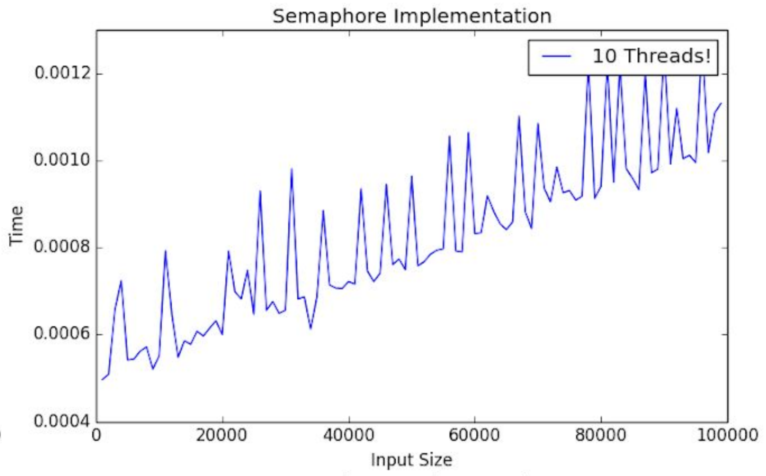


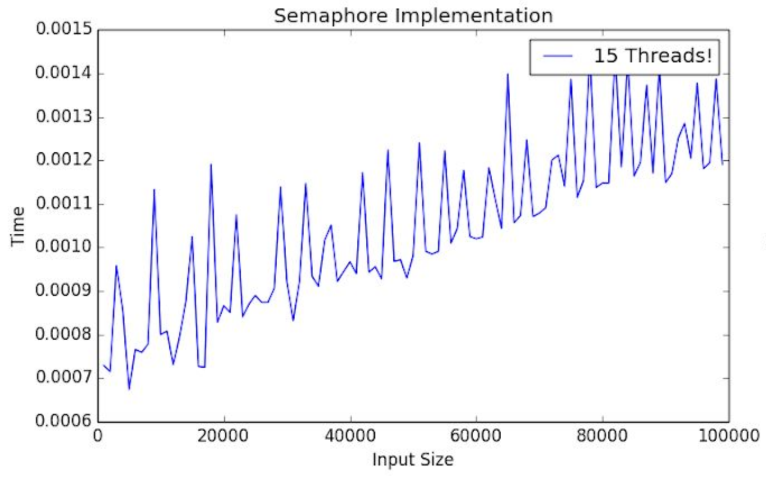


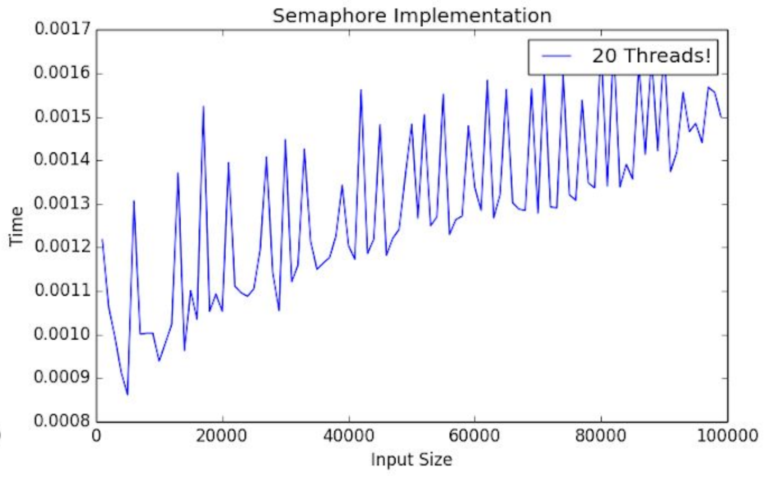
Conclusion: Mutex Implementation with 5 threads gives the best run time.

c) Semaphore Implementation



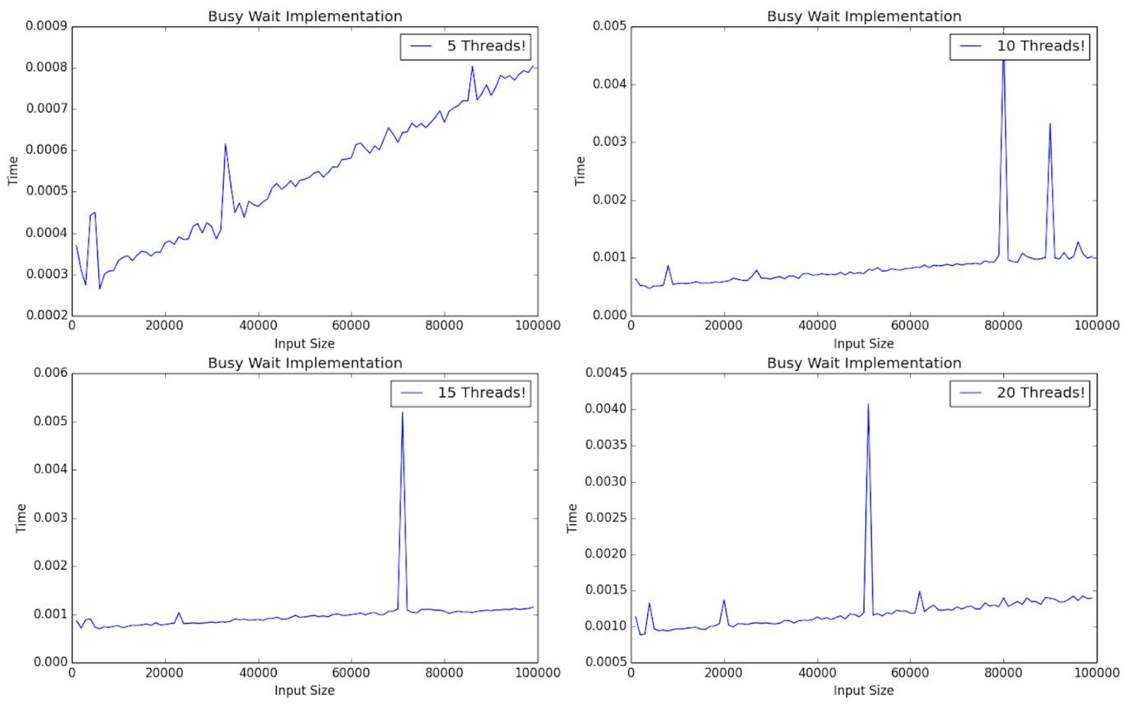






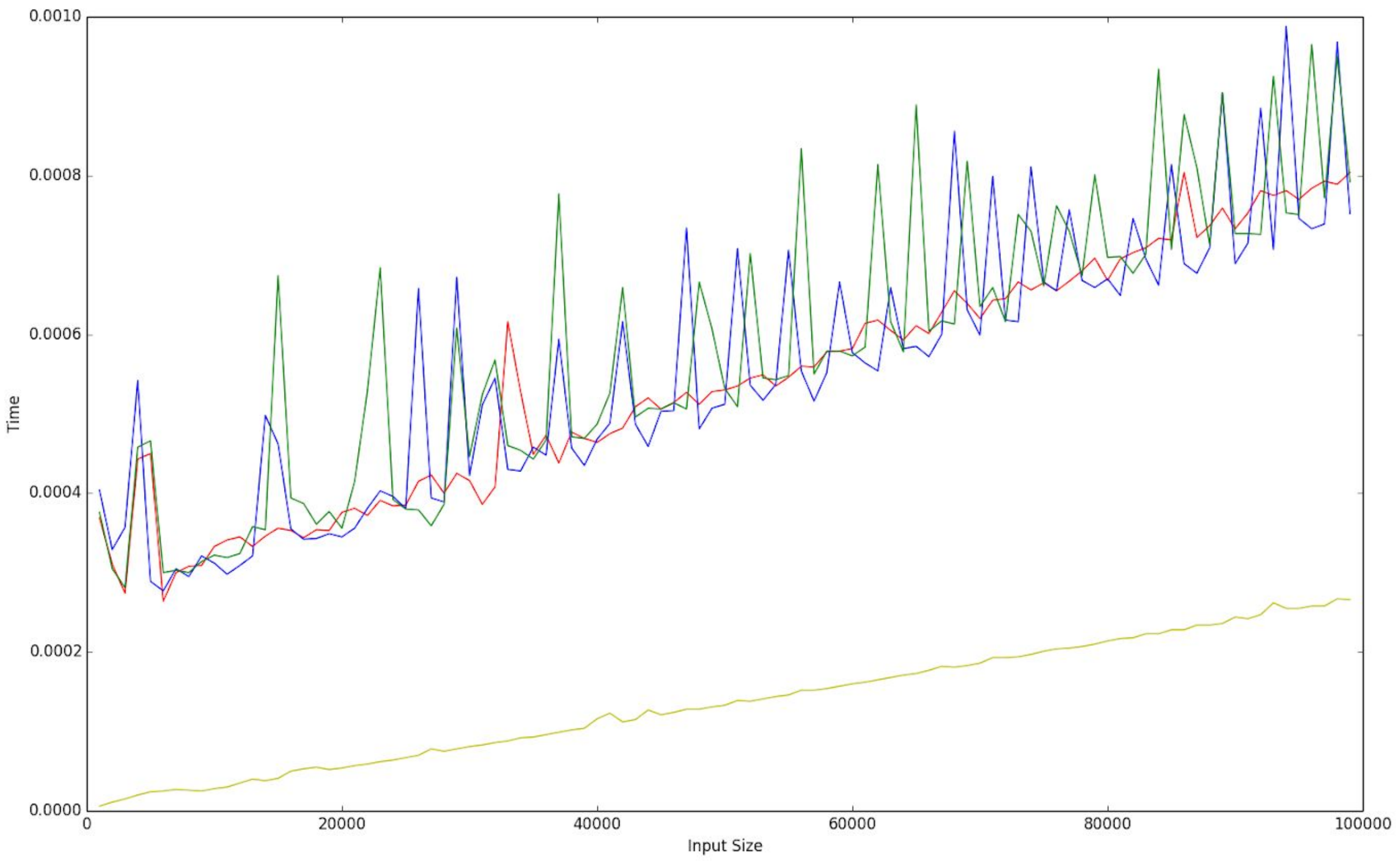
Conclusion: Semaphore Implementation with 5 threads gives the best run time amongst the above.

d) Busy Wait Implementation



Conclusion: Busy wait implementation with 5 threads is best amongst the above.

Comparison between all the implementations:



Red: Busy Wait
Blue: Mutex
Green: Semaphore
Yellow: Serial

Conclusion: For the above experiment, we have concluded the serial implementation has the least running time.

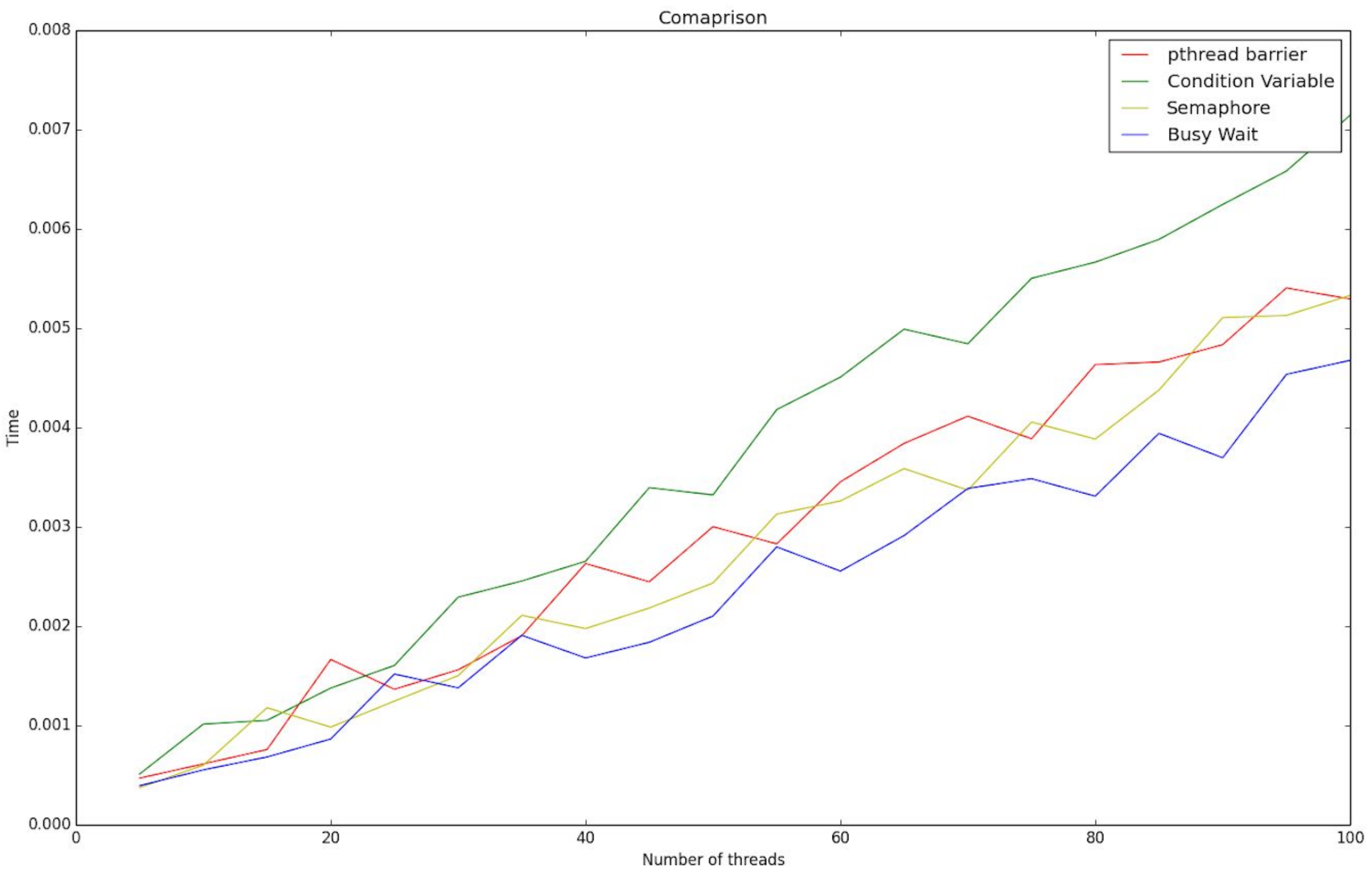
Experiment 2: Thread Barrier

In parallel computing, a barrier is a type of synchronization method. A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.

Thread Barriers have been implemented using the following:
1) Busy Wait and one mutex
2) Binary Semaphore
3) Condition Variable
4) pthread Barriers

In this experiment, for each of the above implementation, we will start from 5 threads and increment upto 100. For each instance of number of threads we run the program 10 times and then take the average to find the time required to implement the barrier.

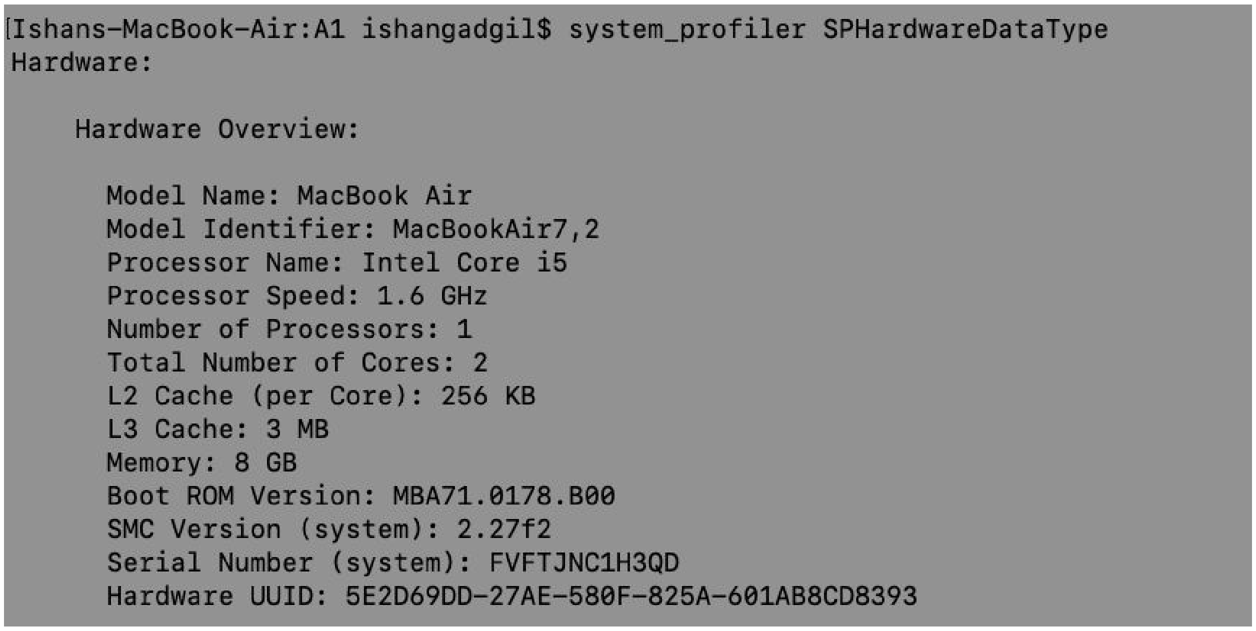
Comparison:



Observation:

Least time taken to implement barrier: Busy Wait Implementation
Semaphore and pthread barrier implementation have similar running time.
Max time taken to implement barrier: Condition Variable Implementation

Hardware



```
[Ishans-MacBook-Air:A1 ishangadgil$ system_profiler SPHardwareDataType
Hardware:

    Hardware Overview:

      Model Name: MacBook Air
      Model Identifier: MacBookAir7,2
      Processor Name: Intel Core i5
      Processor Speed: 1.6 GHz
      Number of Processors: 1
      Total Number of Cores: 2
      L2 Cache (per Core): 256 KB
      L3 Cache: 3 MB
      Memory: 8 GB
      Boot ROM Version: MBA71.0178.B00
      SMC Version (system): 2.27f2
      Serial Number (system): FVFTJNC1H3QD
      Hardware UUID: 5E2D69DD-27AE-580F-825A-601AB8CD8393
```

Raw Data

Since for each implementation, we have iterated from 5 threads to 20 threads in the increments of 5,

1) For serial implementation: serial.txt

2) For mutex implementation: a) mutex5.txt for 5 threads
b) mutex10.txt for 10 threads
c) mutex15.txt for 15 threads
d) mutex20.txt for 20 threads

3) For semaphore implementation: a) sem5.txt for 5 threads
b) sem10.txt for 10 threads
c) sem15.txt for 15 threads

d) sem20.txt for 20 threads

4) For busy wait implementation: a) bw5.txt for 5 threads
b) bw10.txt for 10 threads
c) bw15.txt for 15 threads
d) bw20.txt for 20 threads

<u>Conclusion:</u>

The hardware specs of this laptop shows that it has only one processors and two cores. This would explain why when we increase the number of threads, the running time increases.