The performance profile of synchronization constructs

Name - Rajendra Singh
Roll - 111601017
Topic - Parallel Programming(Assignment 2)

Problem Statement: We discussed various synchronization primitives such as busy-waiting, mutexes, conditional variables, barriers, and read-write locks. You are required to design experiments (or test cases), execute them and measure the performance of each of these synchronization constructs.

Please note that the performance overheads depend on multiple factors such as library version, hardware and system capabilities, computation performed on each thread, number of cores, number of threads and so forth. Therefore, the performance measures are not scalar numbers, but instead is a profile over varying quantities. Use the performance metrics such as speedup, and efficiency discussed in the class if you find them useful. Since there is significant run-to-run variations in multi-threaded programs, ensure you repeat the same experiment at least 5 times while measuring the performance. Report the maximum, minimum, and average time taken for every study you perform.

You have to upload a report, test cases, post-processing scripts (if any), and Makefiles that you used/created for this study. Use graphs in the report to quantify your measurements (with the raw data made available as an appendix), and summarize your findings. Justify any unique observations you made from the data you observed.

Hardware Specification:

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

CPU(s): 4

On-line CPU(s) list: 0-3

Thread(s) per core: 2

Core(s) per socket: 2

Socket(s): 1

NUMA node(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 42

Model name: Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz

Stepping: 7

CPU MHz: 1632.288

CPU max MHz: 2900.0000

CPU min MHz: 800.0000

BogoMIPS: 4589.92

Virtualization: VT-x

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

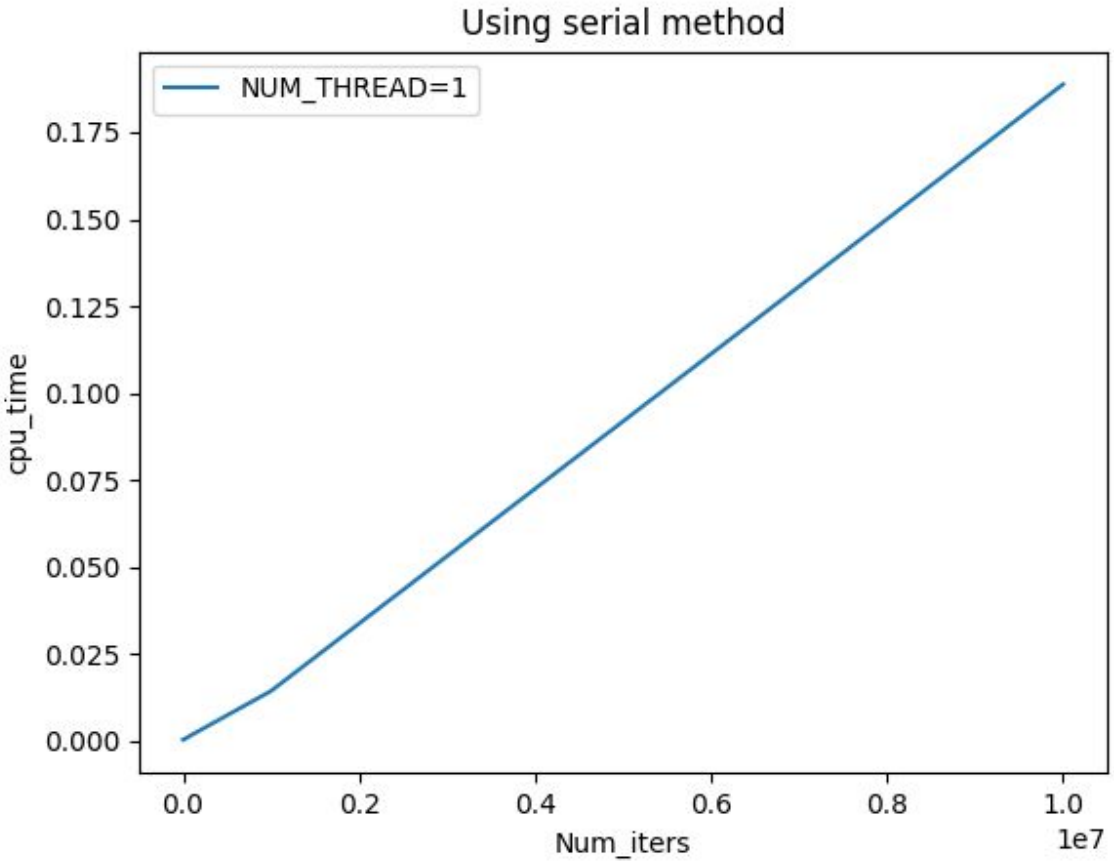
L3 cache: 3072K

NUMA node0 CPU(s): 0-3

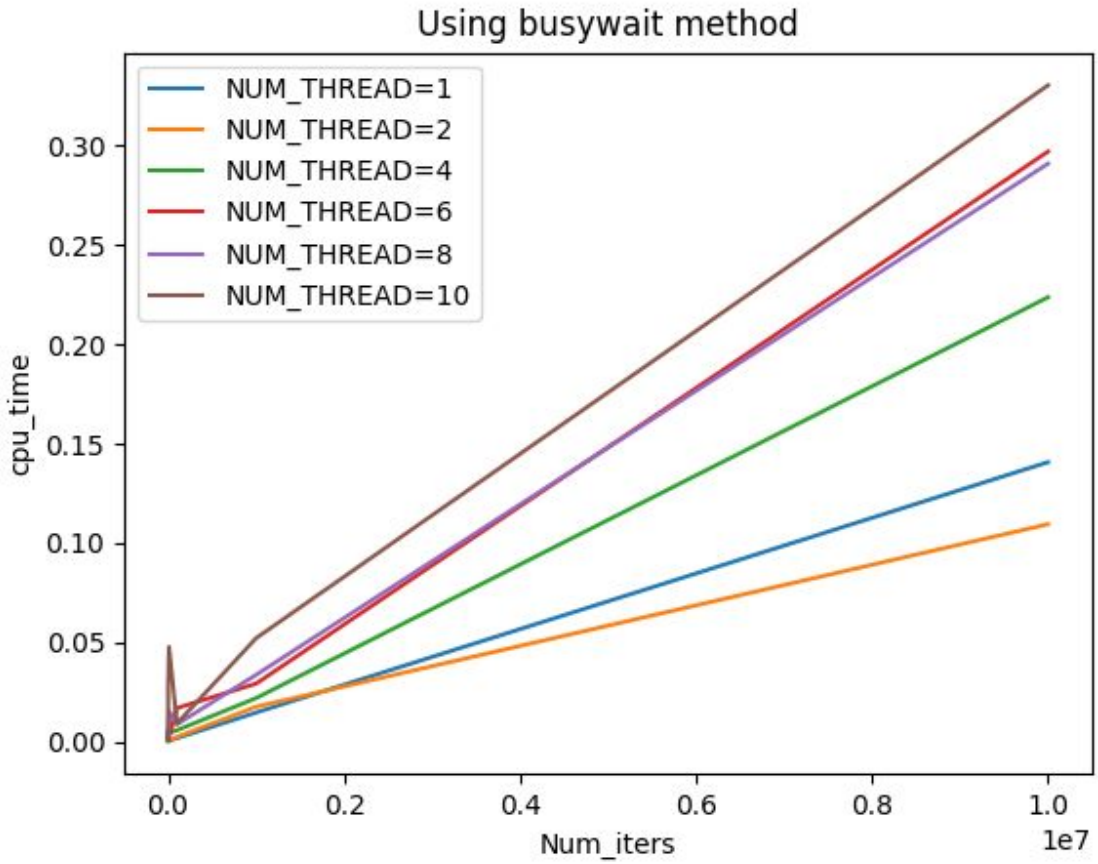
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx lahf_lm epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid xsaveopt dtherm ida arat pln pts flush_l1d

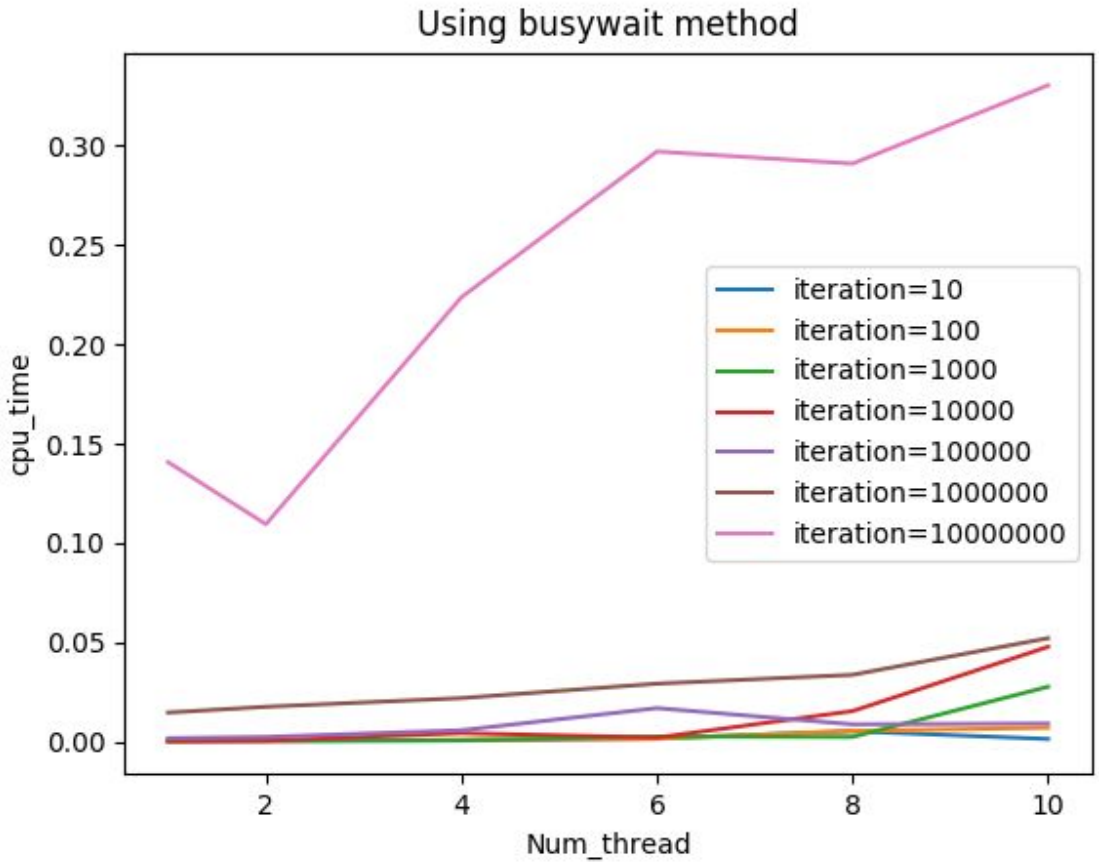
Various Method:

1) Serial

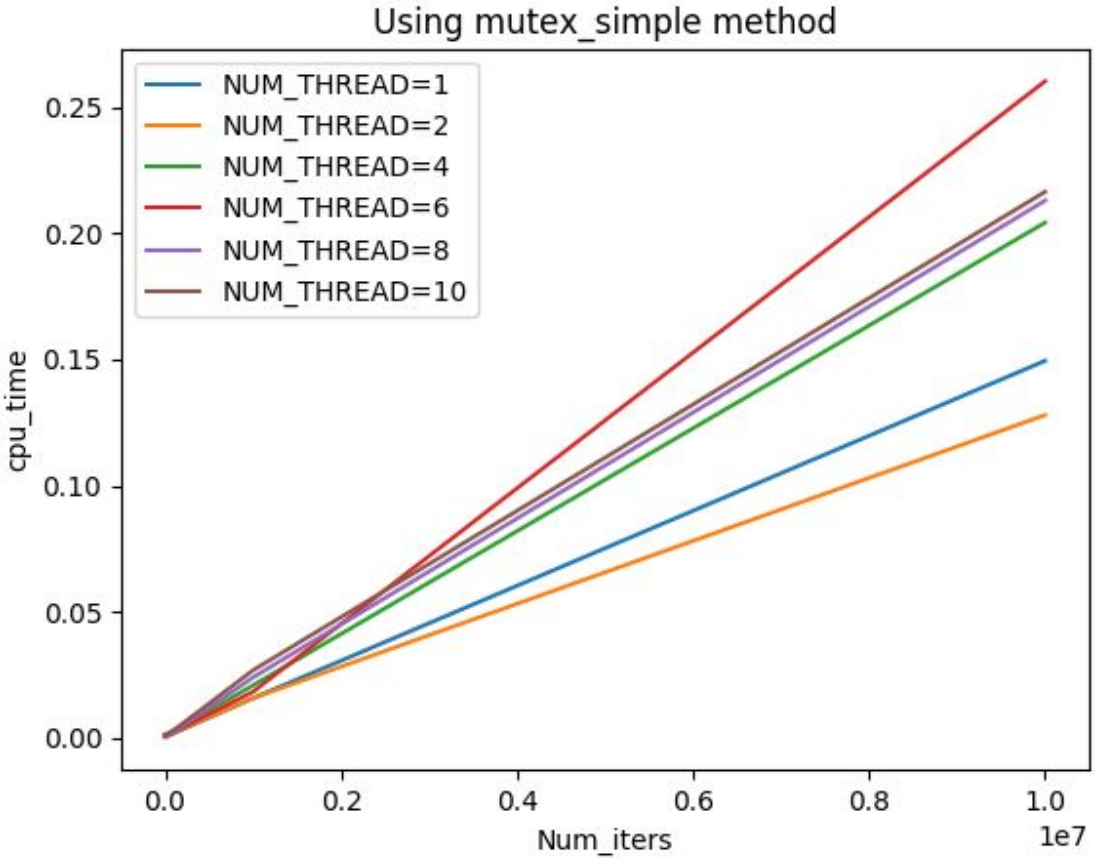


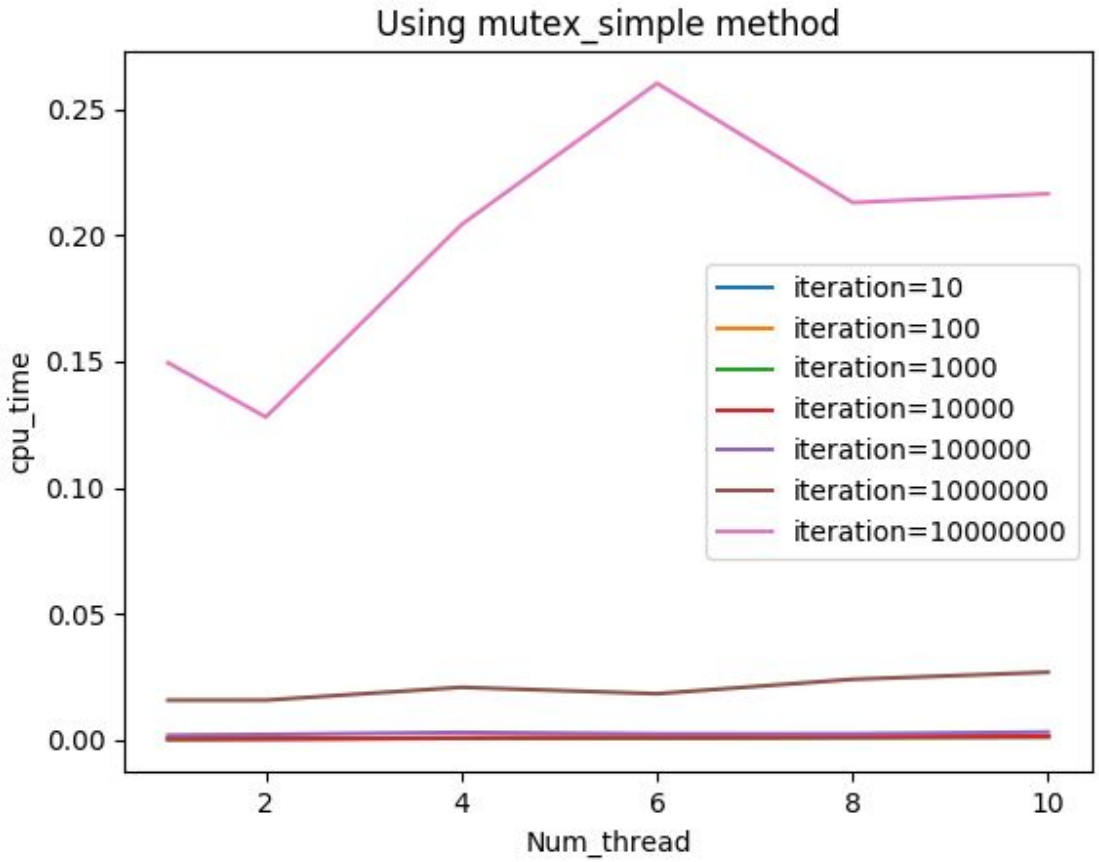
2) Busy Wait


Using busywait method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

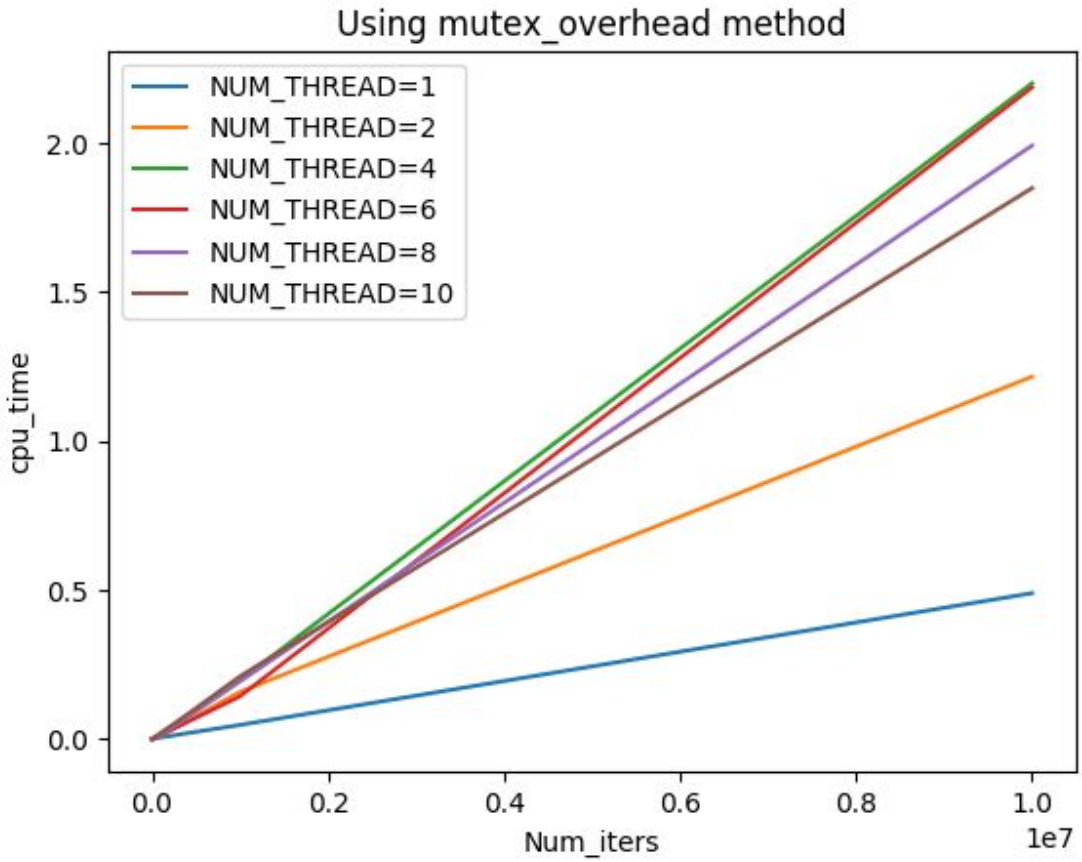


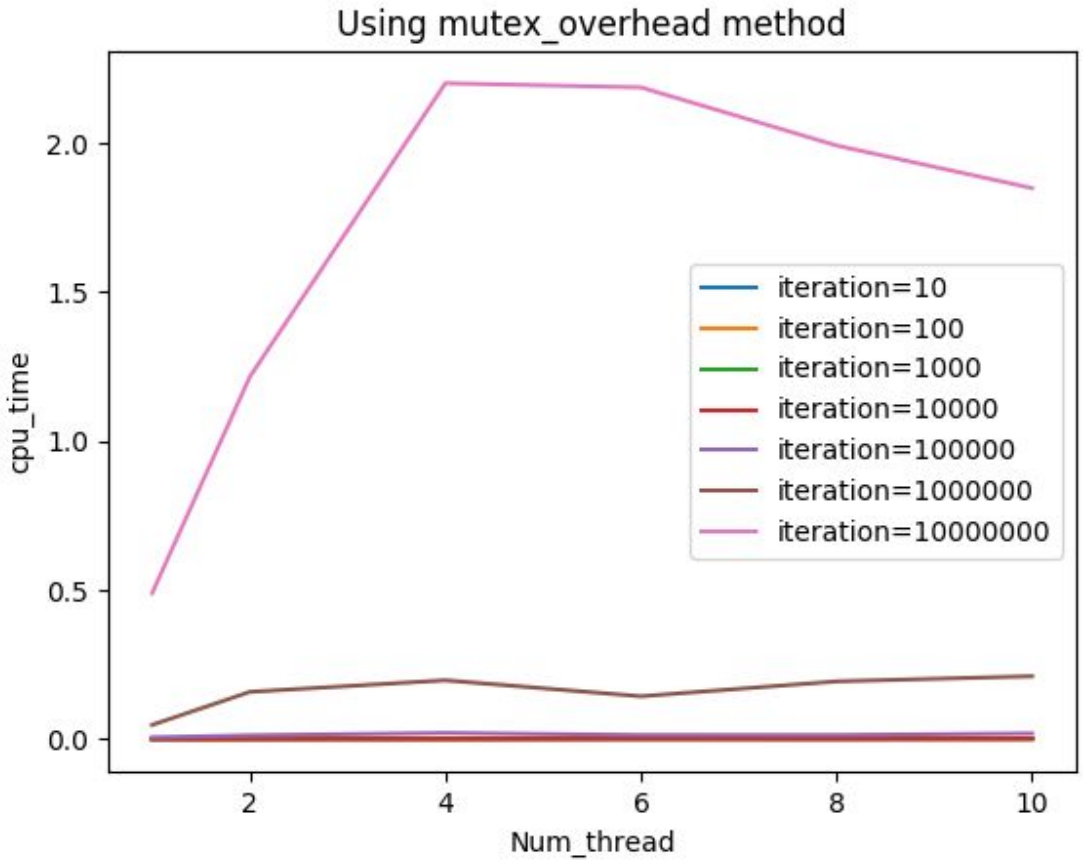
3) Simple Mutex


Using mutex_simple method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

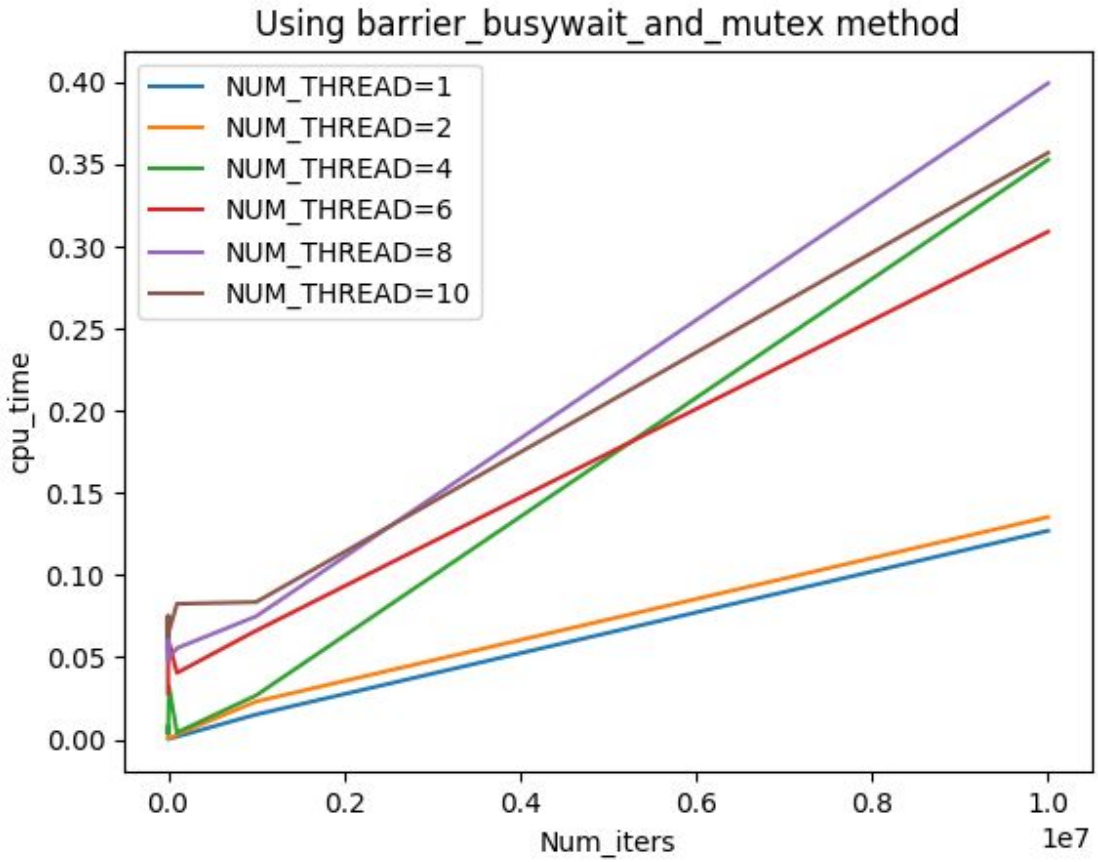


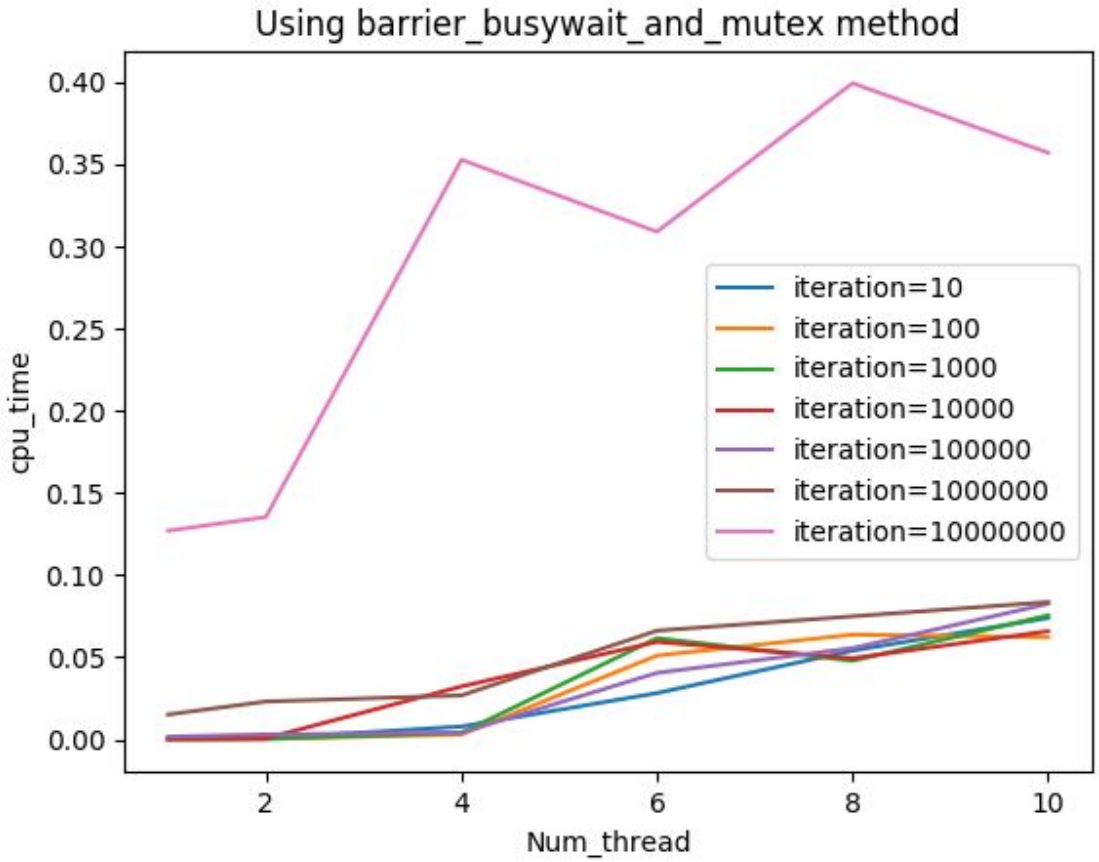
4) Mutex with no overhead


Using mutex_overhead method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

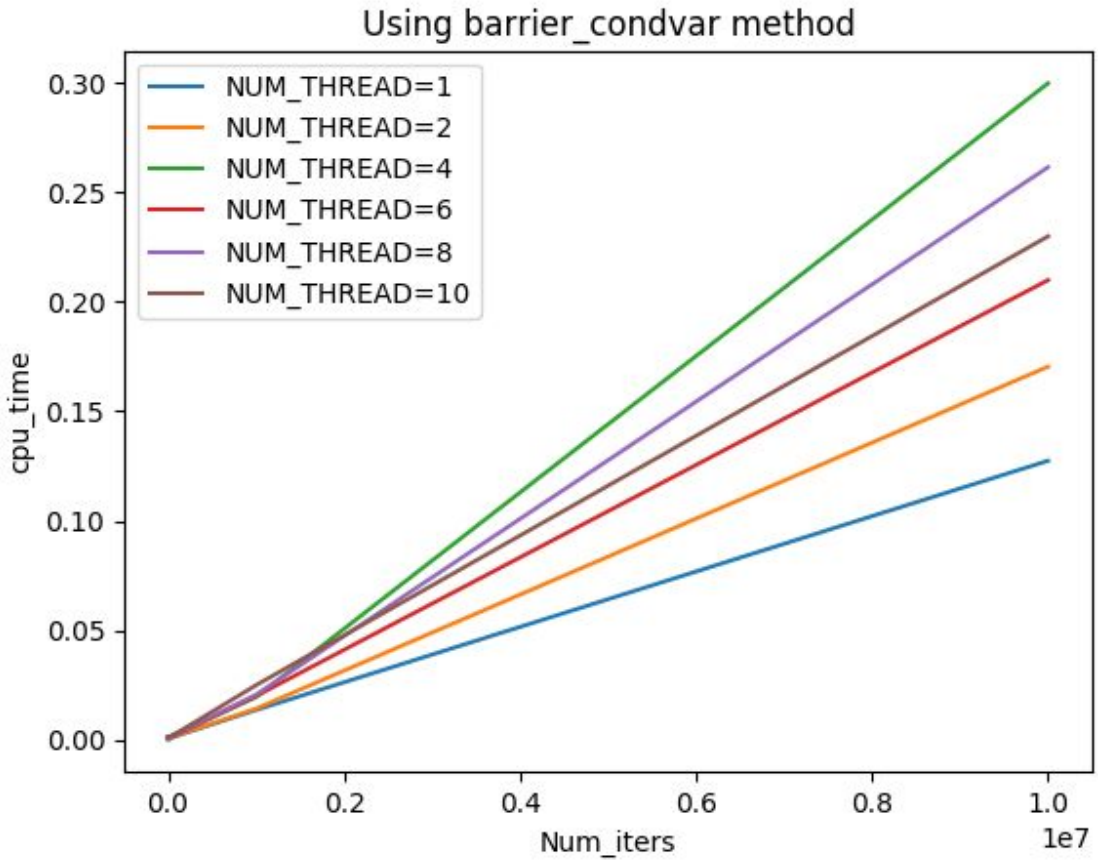


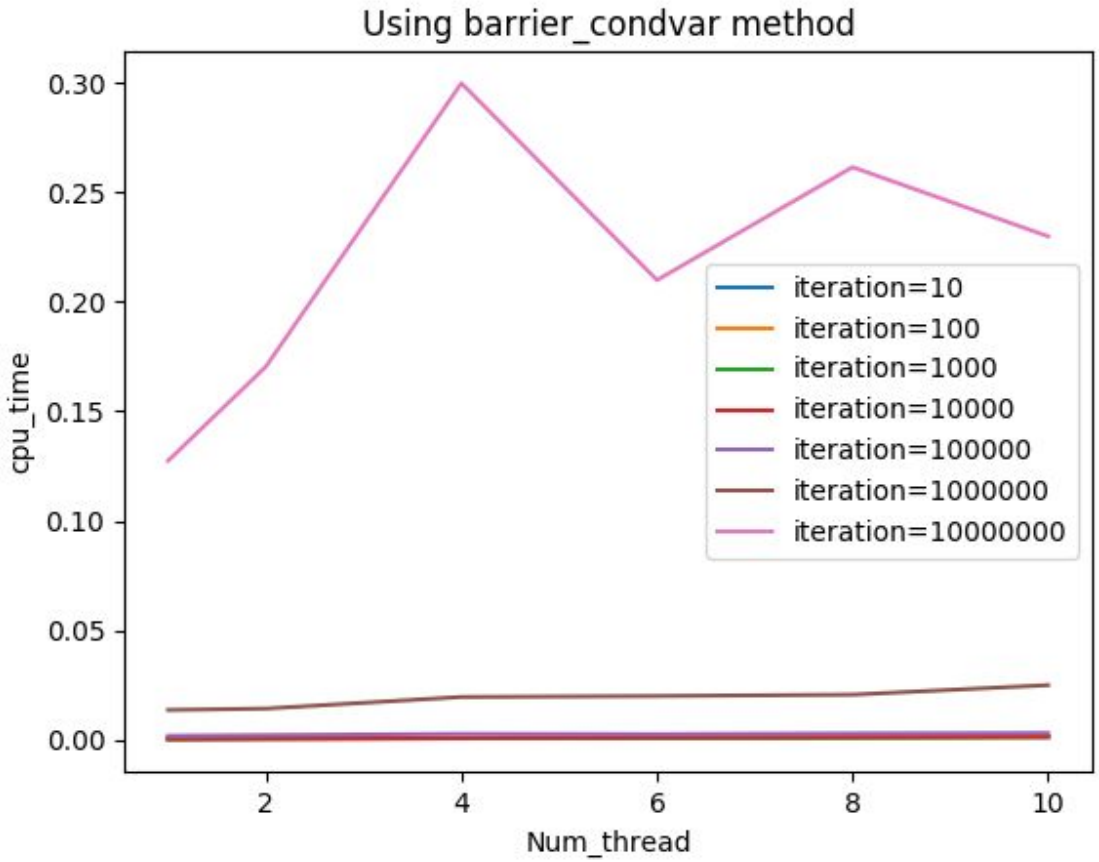
5) Barrier using mutex and busy wait


Using barrier_busywait_and_mutex method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

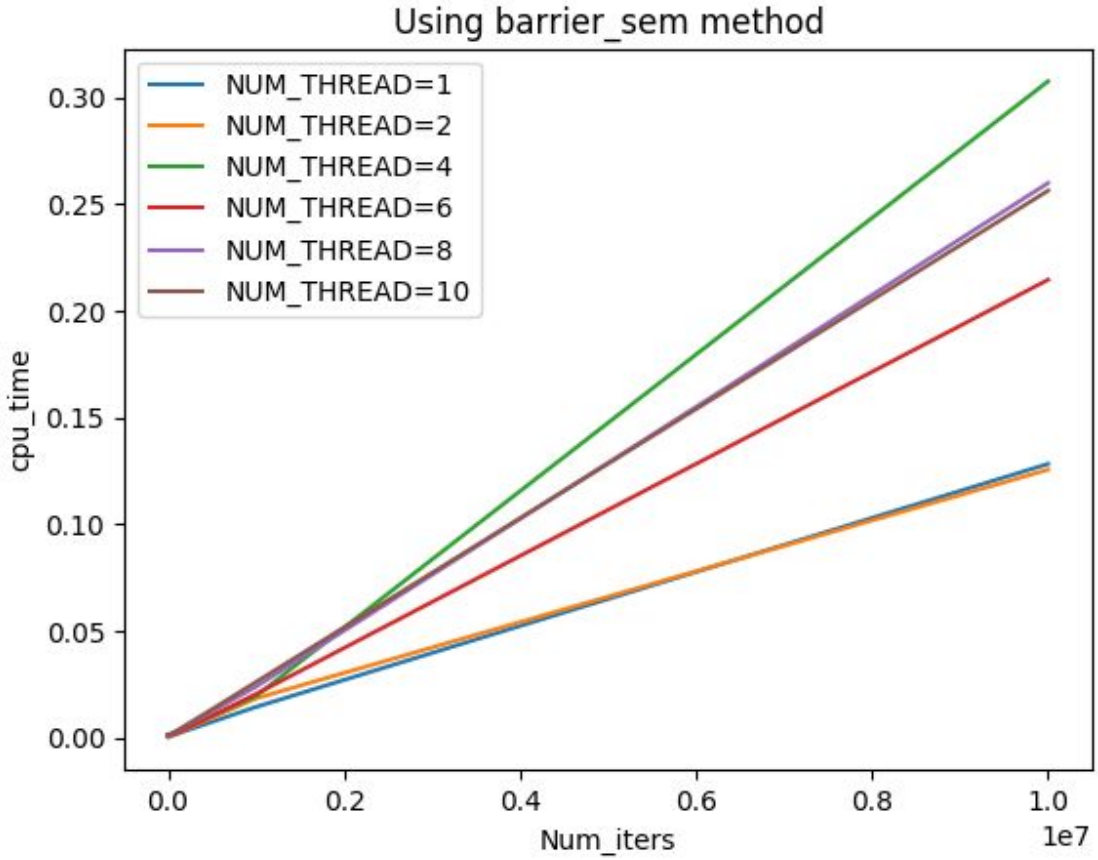


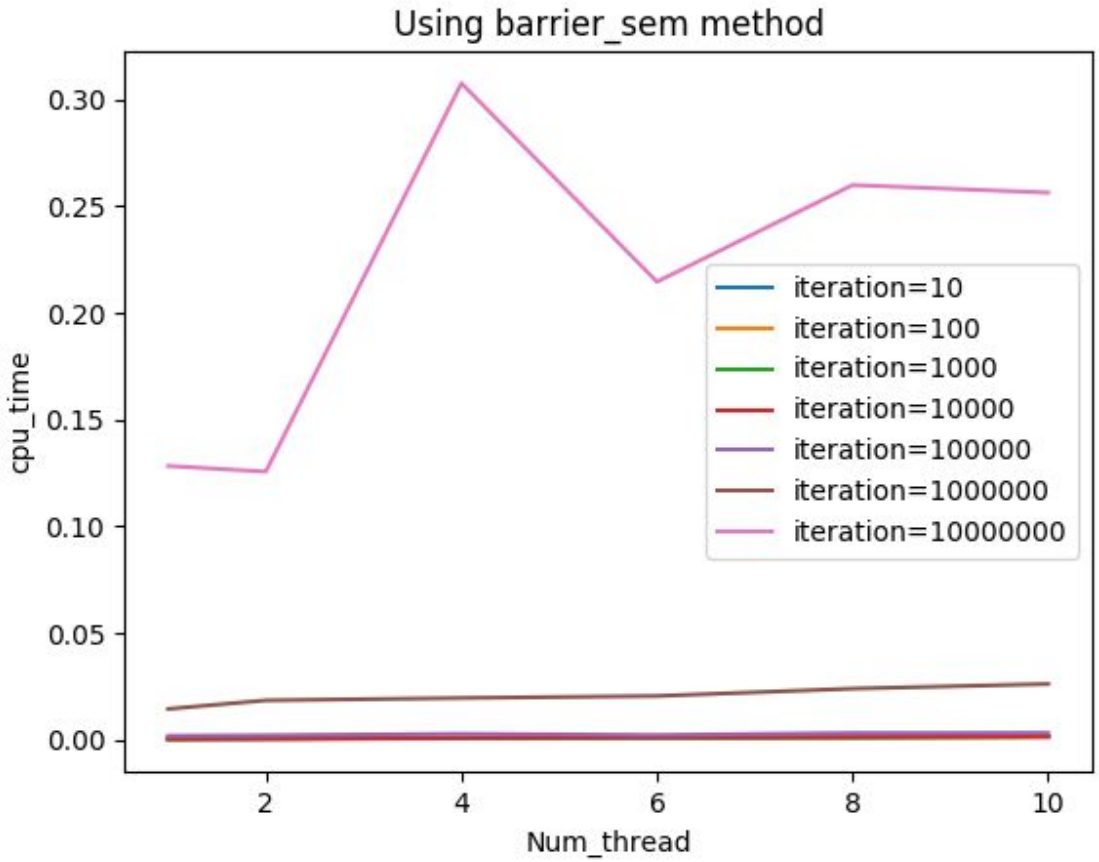
6) Barrier using Conditional variable


Using barrier_condvar method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

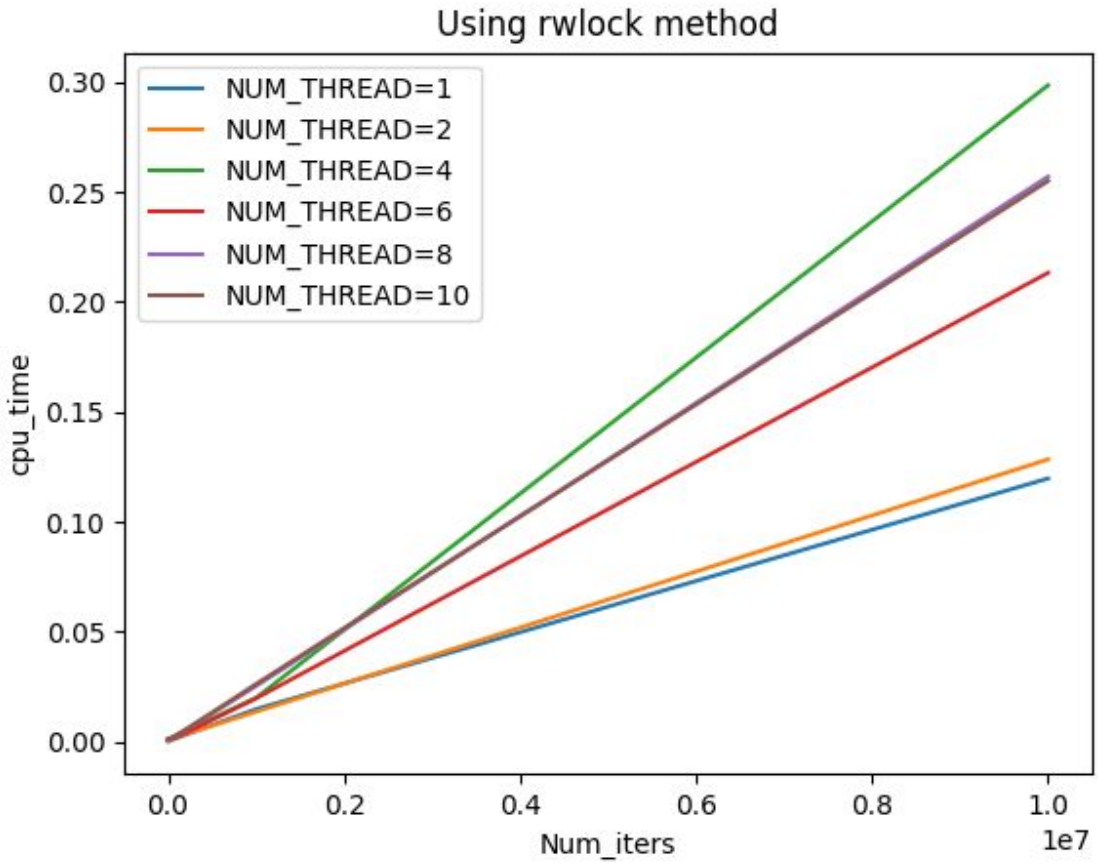


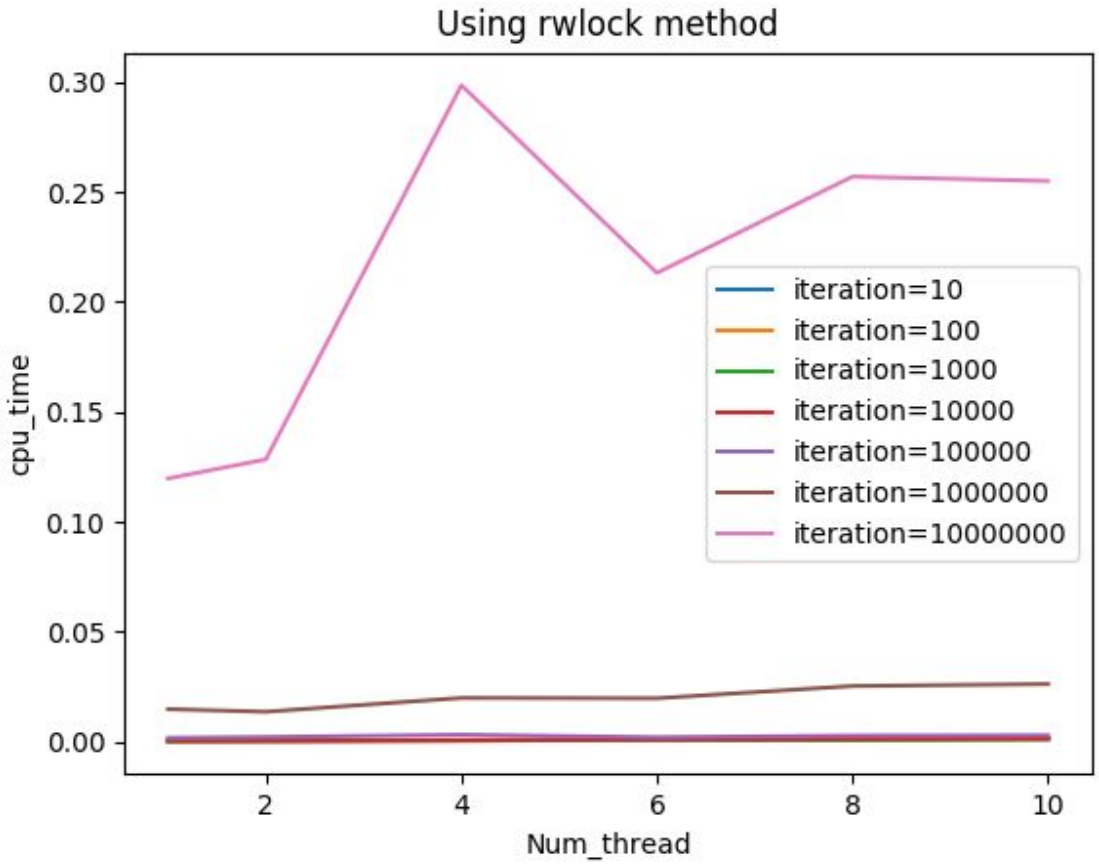
7) Barrier using Semaphore


Using barrier_sem method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

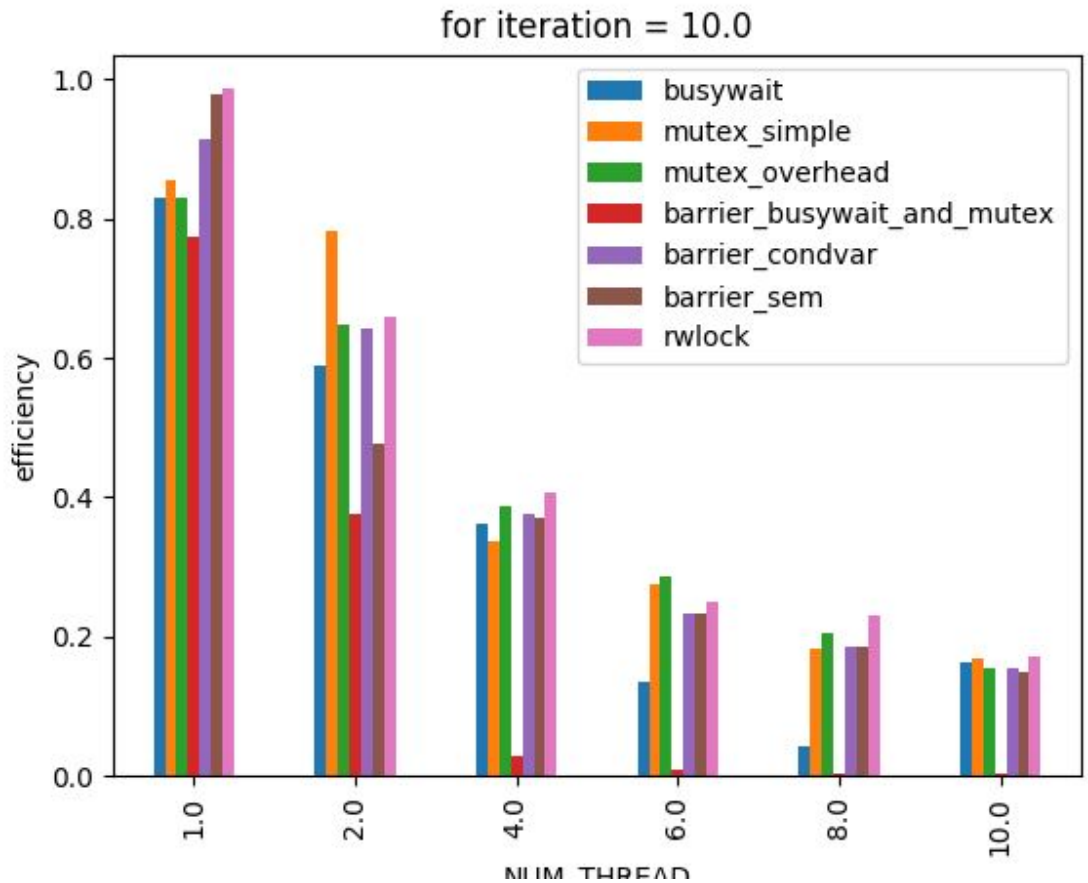


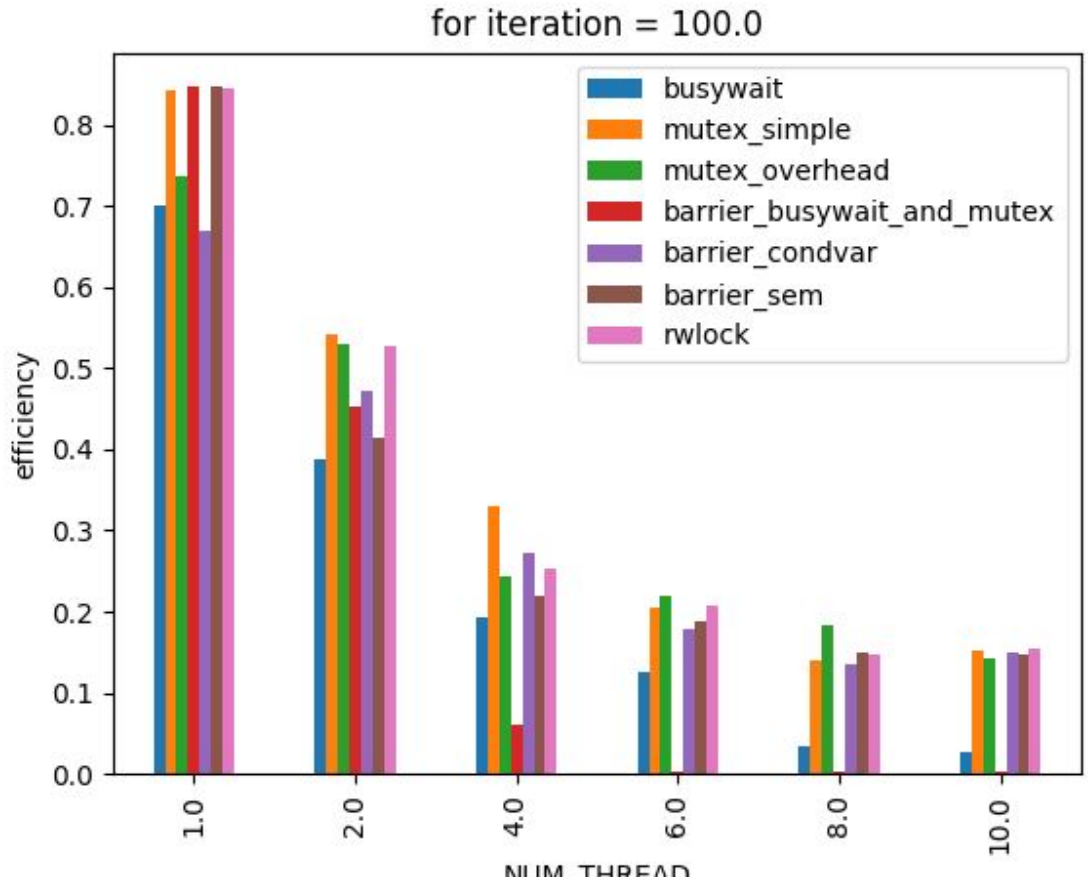
8) Read write lock

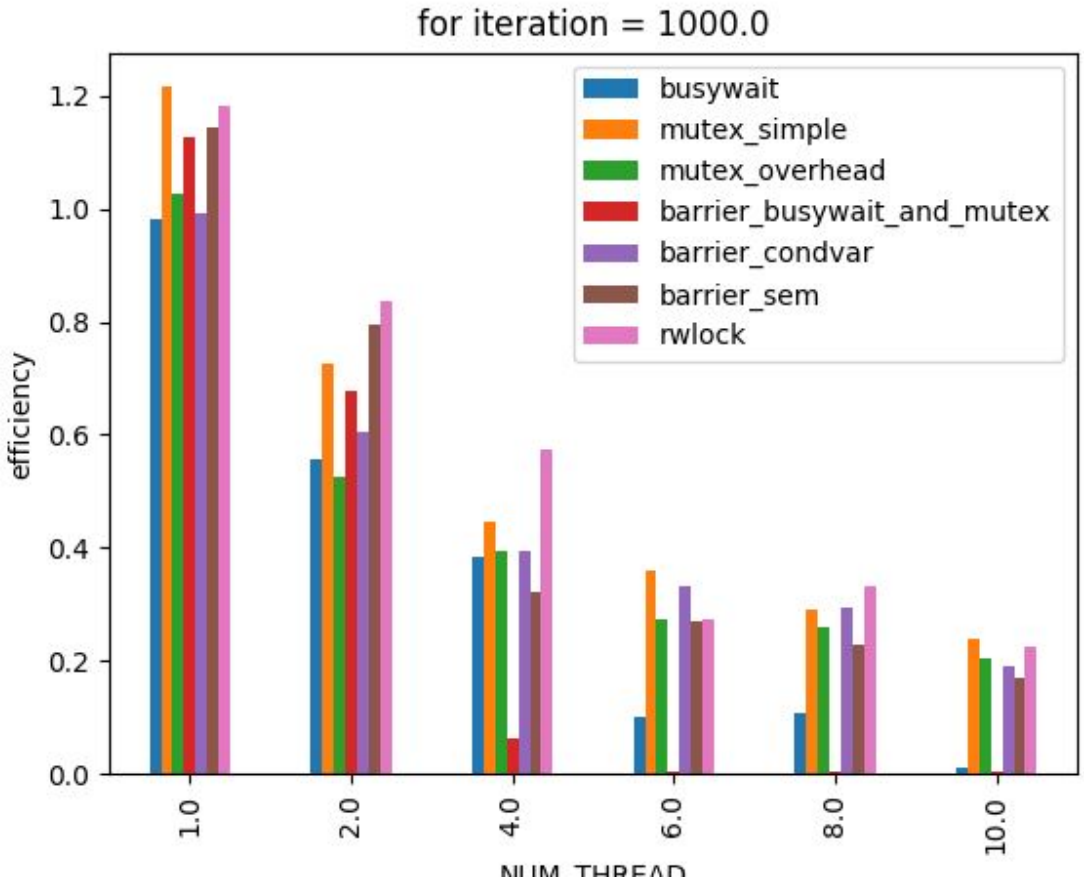

Using rwlock method
NUM_THREAD=1
NUM_THREAD=2
NUM_THREAD=4
NUM_THREAD=6
NUM_THREAD=8
NUM_THREAD=10
cpu_time
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Num_iters

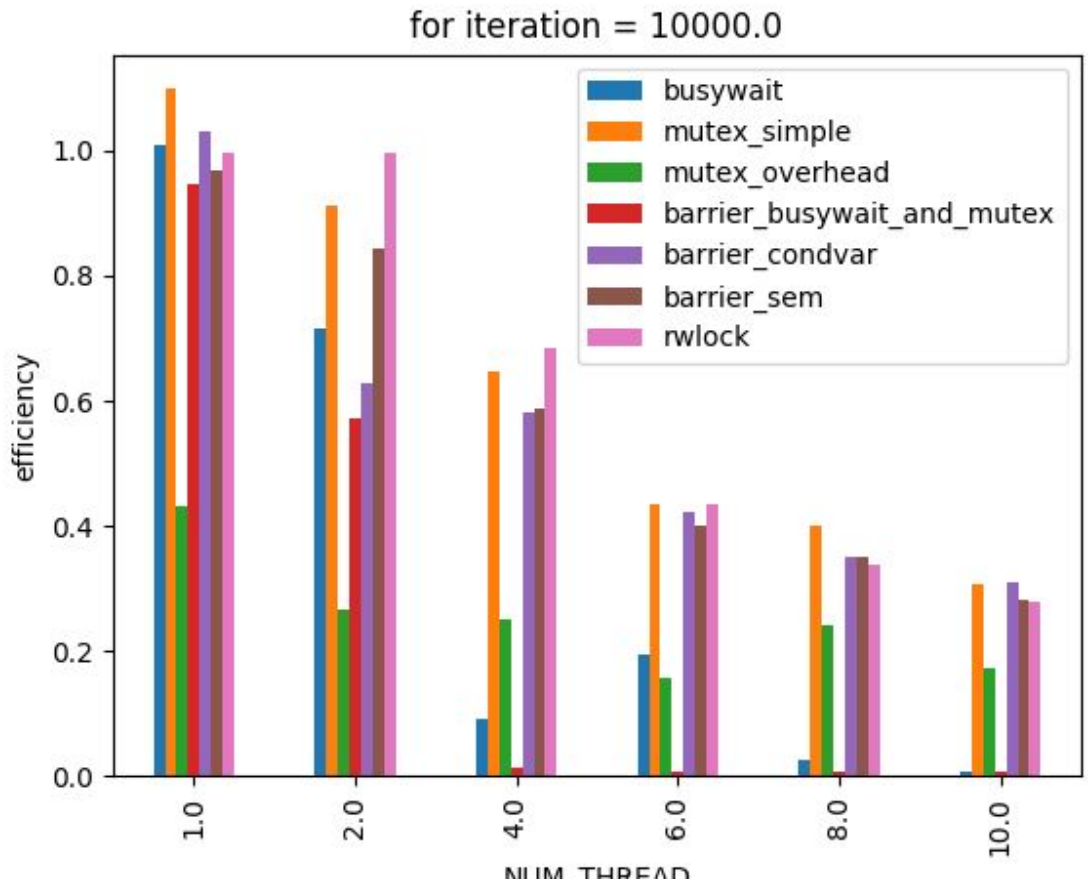


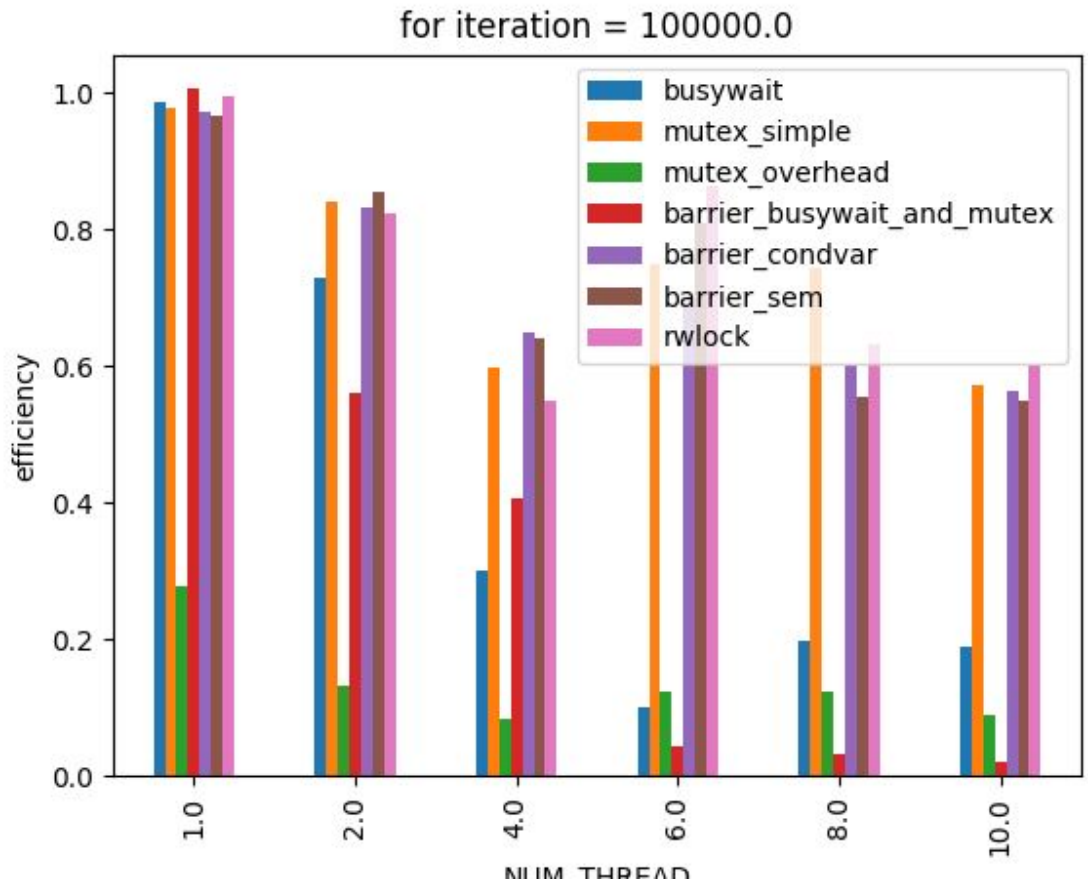
Efficiency Comparation between all methods:

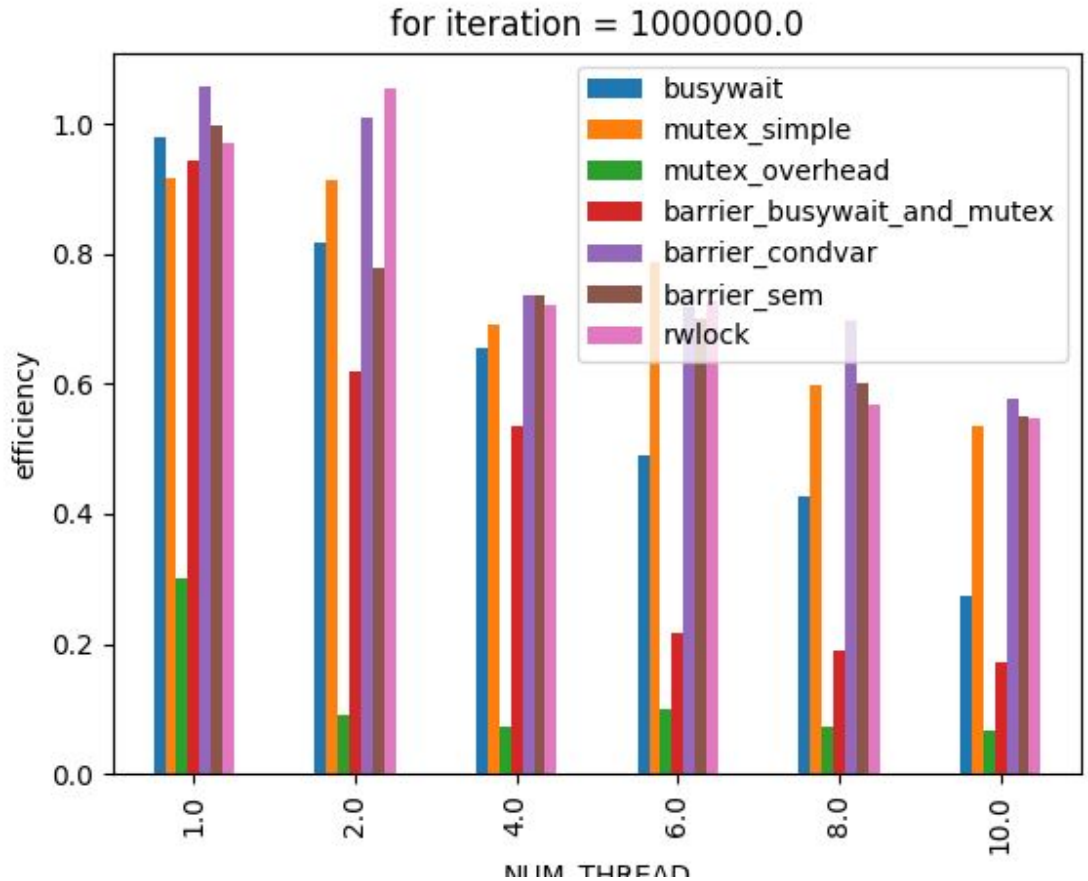

for iteration = 10.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
1.0
0.8
0.6
0.4
0.2
0.0
1.0
2.0
4.0
6.0
8.0
10.0

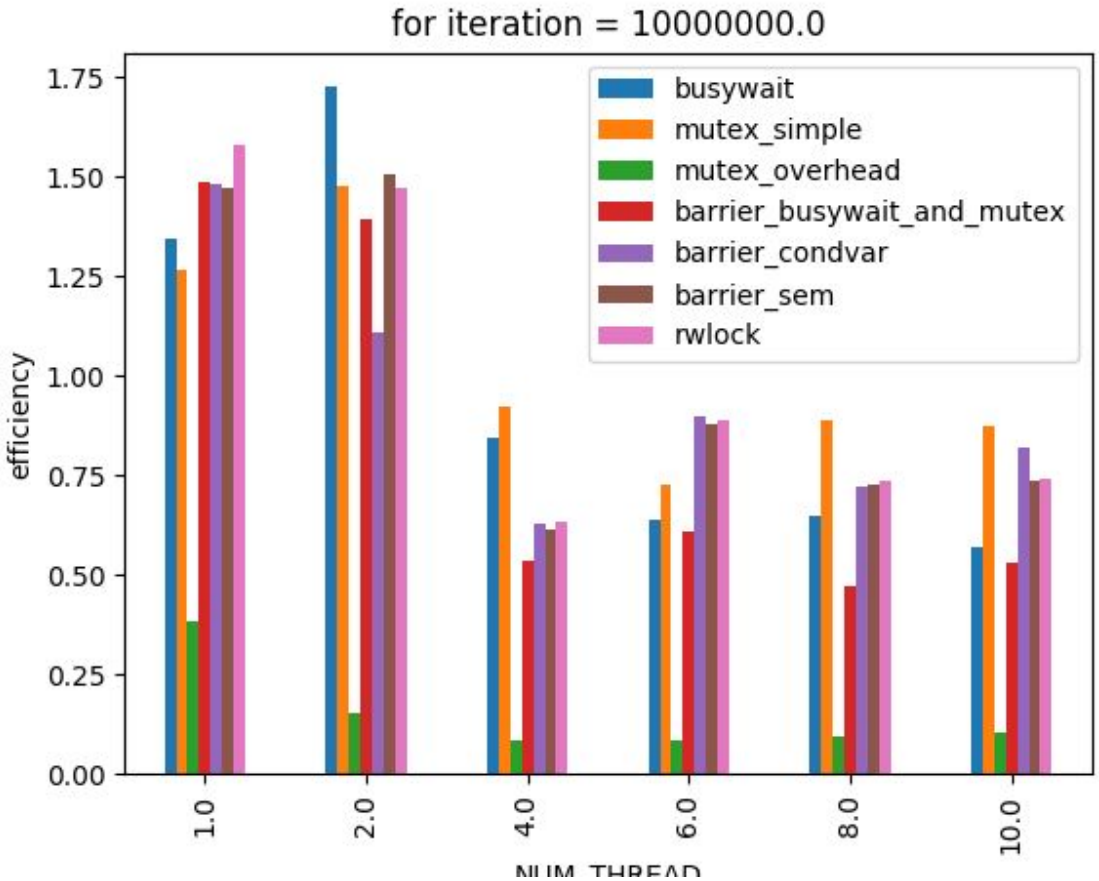

for iteration = 100.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1.0
2.0
4.0
6.0
8.0
10.0


for iteration = 1000.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1.0
2.0
4.0
6.0
8.0
10.0


for iteration = 10000.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
1.0
0.8
0.6
0.4
0.2
0.0
1.0
2.0
4.0
6.0
8.0
10.0


for iteration = 100000.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
1.0
0.8
0.6
0.4
0.2
0.0
1.0
2.0
4.0
6.0
8.0
10.0


for iteration = 1000000.0
busywait
mutex_simple
mutex_overhead
barrier_busywait_and_mutex
barrier_condvar
barrier_sem
rwlock
efficiency
1.0
0.8
0.6
0.4
0.2
0.0
1.0
2.0
4.0
6.0
8.0
10.0



Observation

1) In general, cpu_time **increases** uniformly with an increase in no. of iteration.
2) In the above case, Cpu_time increase with **least gradient** for NUM_THREAD = **2**. And Cpu_time increase with **maximum gradient** for NUM_THREAD = **4**(and more for some cases.)
3) cpu_time increases as NUM_thread **increases smaller no. of iteration**(<4). But for very **large iteration** (>10^6) where cpu_time **increases** for NUM_thread(**<4**) and then eventually start **decreasing** for NUM_thread(**> 4**) and then again start **increasing** at NUM_thread = **8**(6 in some cases). Which mean my **optimal no. of the thread is 8**.
4) Also looking at the specification, it clear that no. of thread(per core) X No. of core = **8**. Which **match with a practical result**.

5) For different iteration, the different method worked well. But in general, **RWLOCK** and **Barrier_semaphore** worked best. While mutex with no overhead performed worst. For **low iteration**(=100), **Simple_mutex** performed well.