Performance Measure

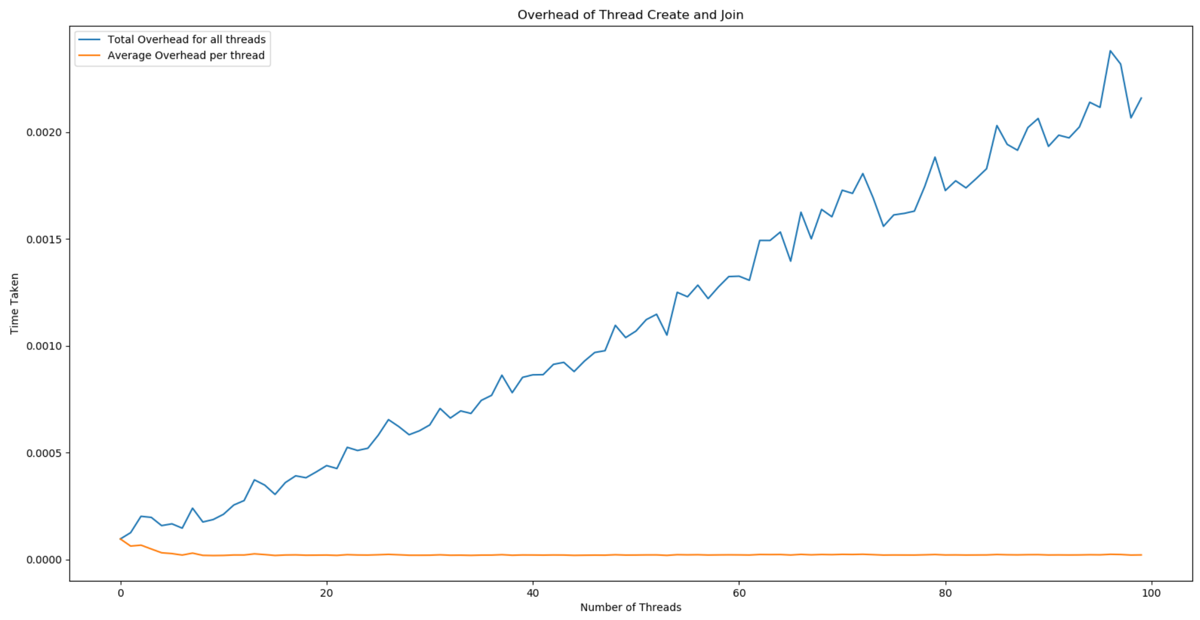
Sparsh Jain
111601026

0. CPU Specifications:

```
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                 : 78
Model name            : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
Stepping              : 3
CPU Mhz               : 500.044
CPU max Mhz           : 3100.0000
CPU min Mhz           : 400.0000
BogoMIPS              : 5184.00
Virtualization        : VT-x
L1d cache             : 32K
L1i cache             : 32K
L2 cache              : 256K
L3 cache              : 4096K
NUMA node0 CPU(s): 0-3
```

1. Overheads

The overhead for creating and joining threads as the number of threads increase, increase almost linearly but average overhead per thread seems to be constant after #threads >= #cores, before that, it decreases as seen in the following graph



The graph is shown after taking an average over 100 runs of the program
Also, on an average (again, taken over 100 runs) overhead for mutex lock/unlock and semaphore lock/unlock is less than 1 microsecond (the precision of timeval struct in time library of c used to measure time)

To Measure actual performance, we check through different programs!

2. Histogram

The program averages over 10 times with random input, and the problem is solved using mutex, busy wait in case of shared variable, and local variables are used as 3rd approach.

Here are the results: (Time in secs)

Size of input	Threads spawned	Time Taken Busy Wait	Time Taken Mutex	Time Taken Local
1000	1	0.000028	0.000045	0.000022
1000	4	0.000824	0.000124	0.000048
1000	8	4.598171 (!!!)	0.000157	0.000169
1000000	1	0.012131	0.027955	0.004446
1000000	4	0.092658	0.145569	0.001848
1000000	8	-	0.149519	0.002632
100000000	1	1.179101	2.776594	0.416924
100000000	4	8.962816	14.203680	0.171422
100000000	8	-	15.422888	0.180475

Clearly busy wait shoots up very high when #threads become higher than #cores
Also, Even mutex doesn’t perform good when input size increases!
Let’s see how Local variable approach performs:

Size of input	Threads Spawned	Time Taken (in seconds)
1000000000	1	4.424654
1000000000	2	1.950792
1000000000	4	1.738082
1000000000	8	1.851233
10000000000	1	5.313981
10000000000	2	2.746883
10000000000	4	2.440457
10000000000	8	2.446395

Even the local variable approach doesn’t improve much when #threads increase #cores
Speedup is max when #threads = #cores