CS 6301: Special Topics is Computer Science- Introduction to Multicore Programming Section 001
Programming Assignment 2 Report
Submitted By: Rohit Bhattacharjee (rxb151030) and Suraj Poojary (ssp151830)

a) Experiment details:

We used the TACC stampede cluster to conduct our experiments on the different lock implementations. Here is the system configuration of the system:

- OS: Linux
- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 16
- Thread(s) per core: 1
- Core(s) per socket: 8
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 45
- Stepping: 7
- CPU MHz: 2701.000
- BogoMIPS: 5399.22
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 20480K
- NUMA node0 CPU(s): 0-7
- NUMA node1 CPU(s): 8-15

We used Java (SE 8) as the programming language.

Range of values

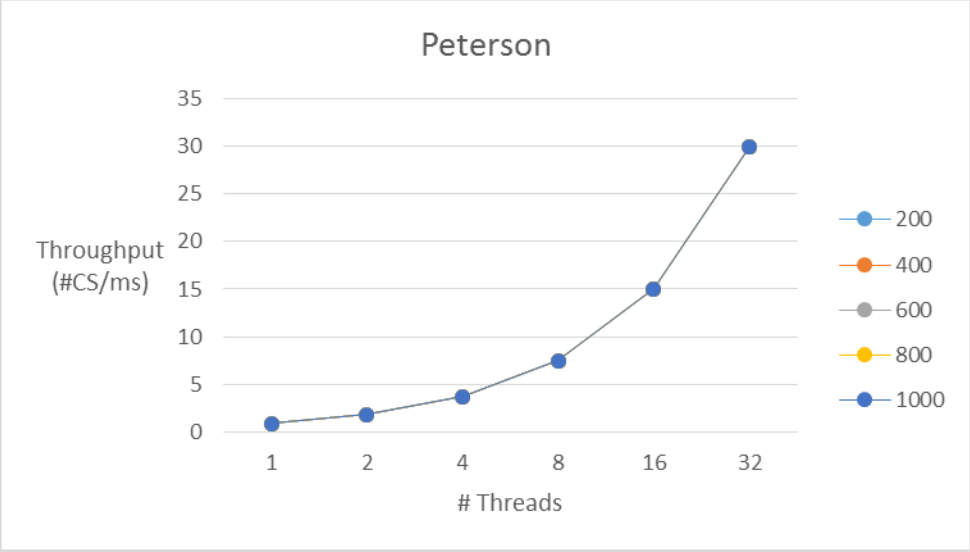
Threads: 1 to 32
Inter Request Delay: 10 to 1000 nanoseconds (10 ns =1 time unit)
Runs for each data point: 15

b) Graphical Data:

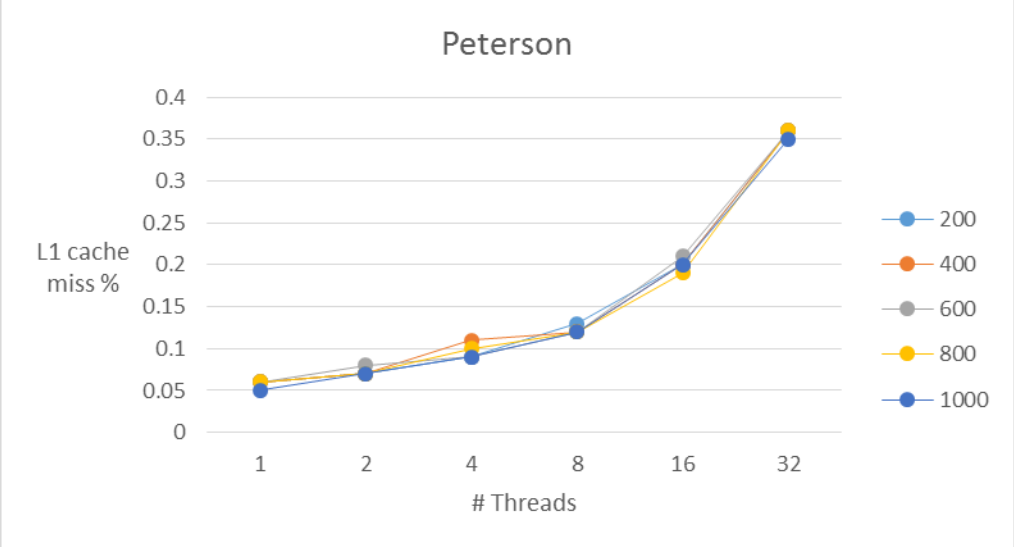
Two graphs have been plotted for each of the five different implementations with System Throughput and L1 cache miss % serving as the Y-axis for the 2 graphs, #threads as the X-axis and the inter-request delay values in the legend.

- **Peterson's Algorithm implemented for n-threads using Binary Tree locks:**

Throughput:

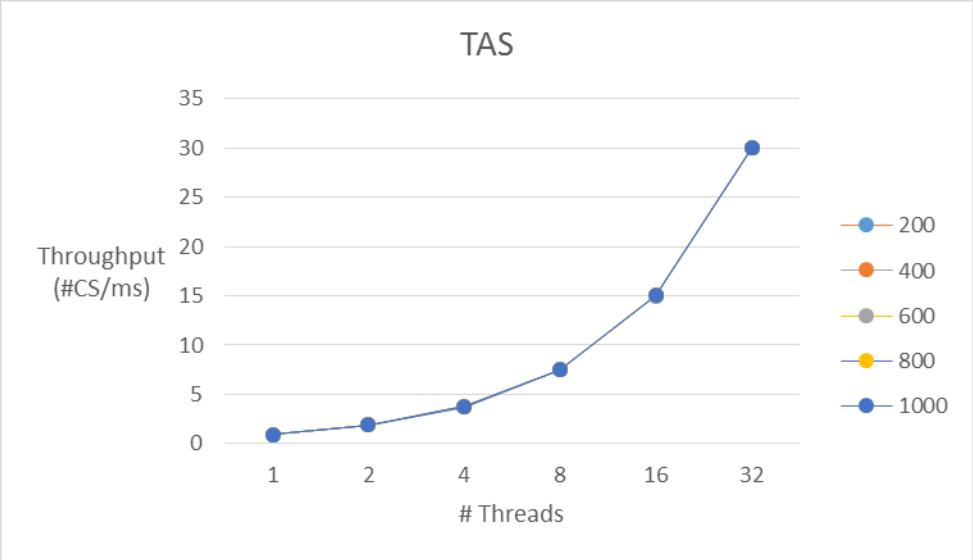


Cache Performance:

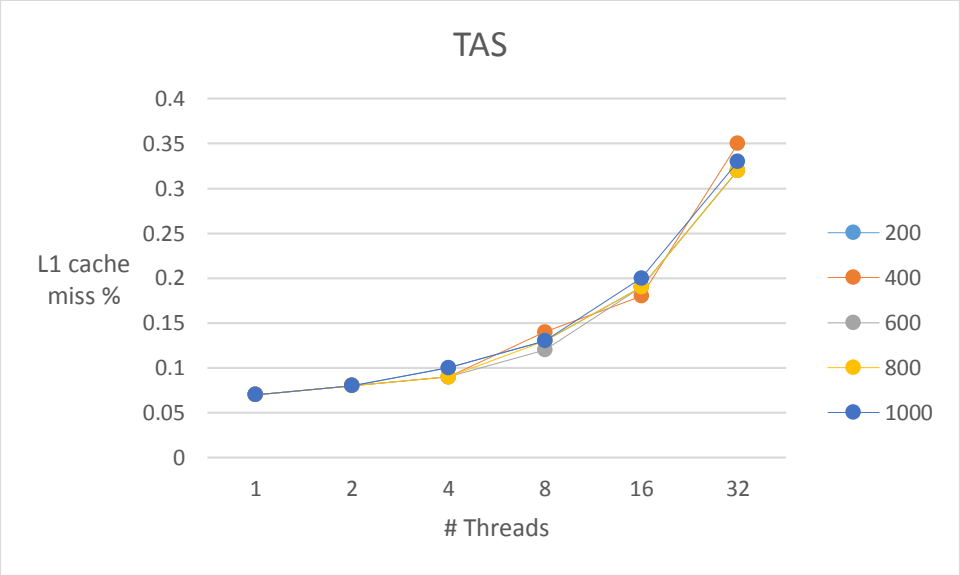


- **TAS Lock:**

System Throughput:

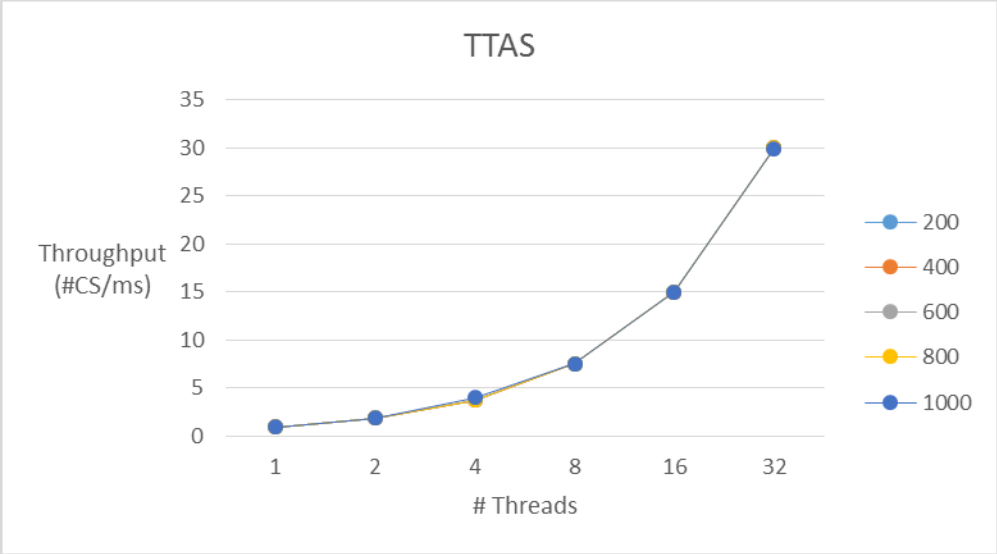


Cache Performance:

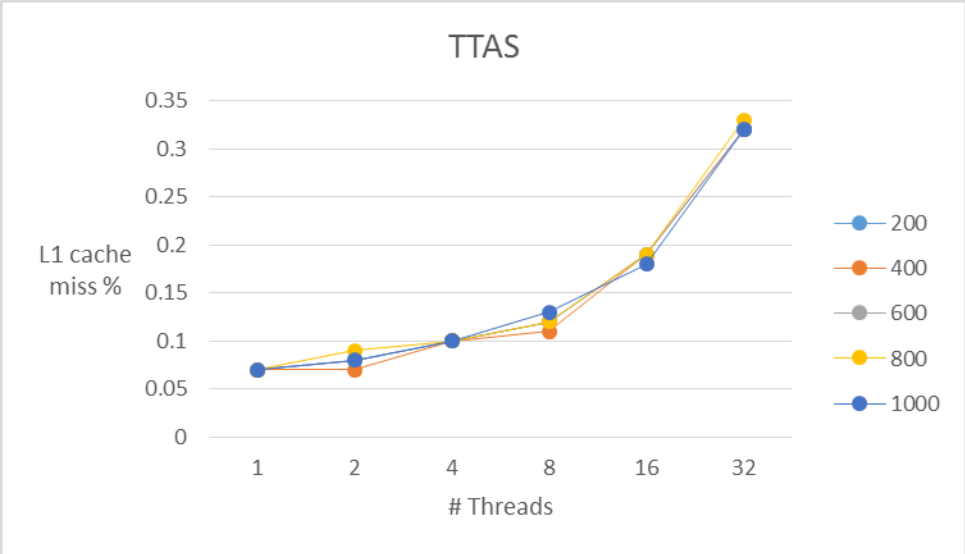


- **TTAS Lock:**

  System Throughput:

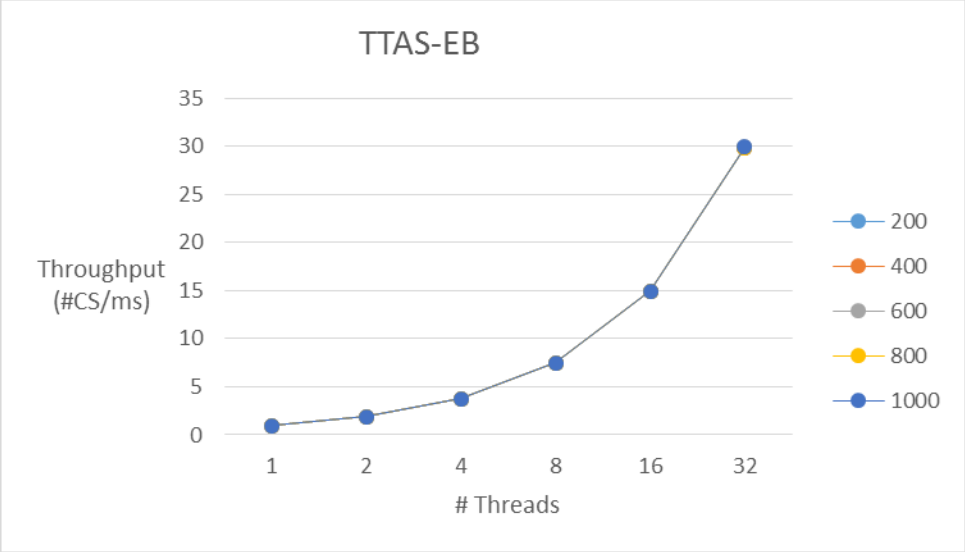


  Cache Performance:

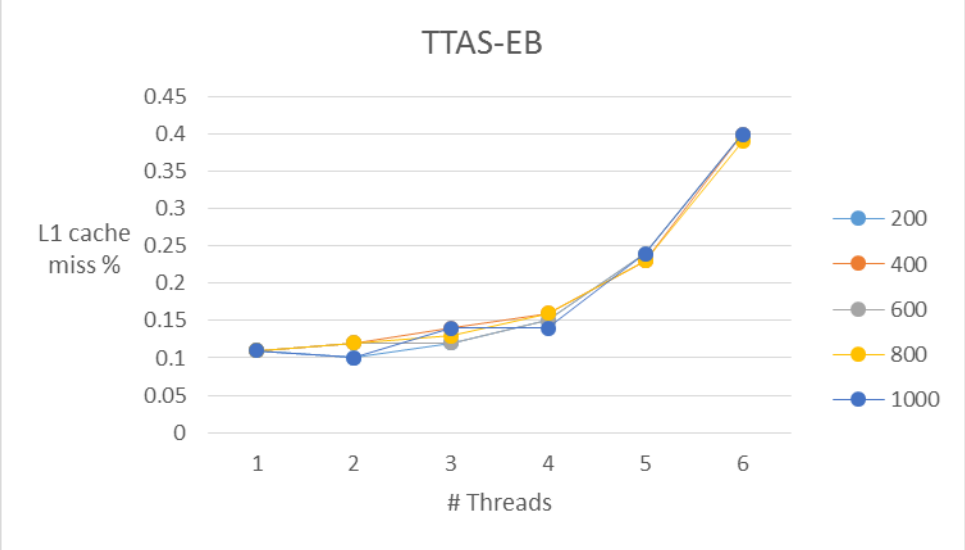


- **TTAS-EB Lock:**
  The maximum and minimum delays for exponential Backoff were experimentally found by running 64 threads with inter-request delay 500ns and 1000 Critical Section runs for each thread.
  The maximum throughput was found for the minimum delay of 1 ns and maximum delay was set to 4ns as the throughput decreased when the max delay was greater or less than 4ns. Experimental data has been provided in the excel sheet "TTAS-EB Experiment"

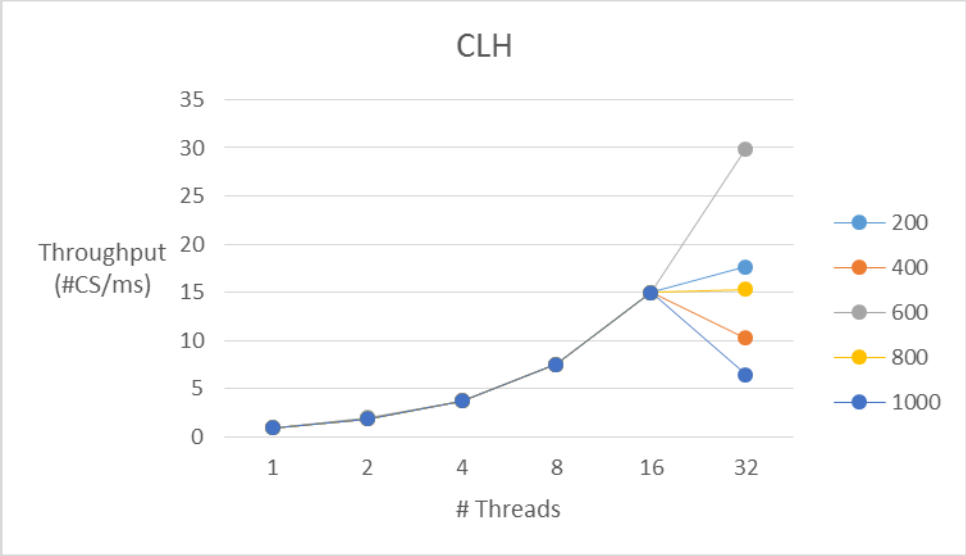
  System Throughput:



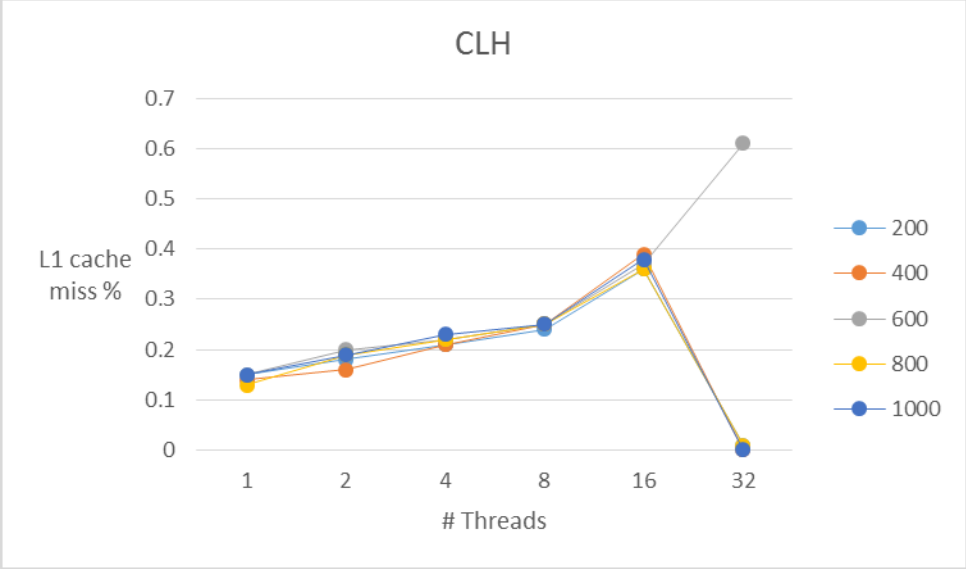
  Cache Performance:



- **CLH Lock:**
  System Throughput:



Cache Performance:



c) Discussion of Results:

- For all cases, the throughput increases with the number of threads, but does not fluctuate much with the change of inter request delays.
- L1 cache miss % similarly increases with the increase in number of threads in the systems, and does not fluctuate much when inter request delay is changed (more variation than the throughput graph though).
- CLH lock performed **abnormally** in case of 32 threads as the throughput started decreasing. In CLH, as the threads spin on the predecessor to get into critical section, if there is a large contention, the spinning locks will form a queue, and one thread may have to wait for 31 threads to complete critical section in order to gain access. This may have reduced the performance to a large extent.
- From our observations, the throughputs for each implementation were pretty much similar for each value of inter-request delay. In terms of L1 dcache miss %, Peterson, TAS and TTAS performed well for less number of threads. TTAS performed well overall whereas L1 cache miss % for CLH dropped significantly for 32 threads. [Refer Table below]

Average L1 dcache miss % with respect to number of threads.

#threads	1	2	4	8	16	32
Peterson	0.058	0.072	0.096	0.122	0.2	0.358
TAS	0.07	0.08	0.094	0.13	0.19	0.328
TTAS	0.07	0.08	0.1	0.12	0.188	0.322
TTAS-EB	0.11	0.112	0.13	0.152	0.236	0.398
CLH	0.144	0.184	0.218	0.248	0.372	*0.126