SeanCooke8892@gmail.com

Parallel and Distributed Systems

README

Purpose

When running parcount, a given number of threads (t) will increment a counter a given number of times (i). After running parcount, the counter should read t * i.

t and i are provided to parcount as command line arguments according to the convention "./parcount –t [t] –i [i]". The argument directly following "–t" (if any) will be t. The argument directly following "–i" (if any) will be i. If multiple "–t" or "–i" flags are found, the last one will be used. If "–t" or "–i" is the last command line argument, it will be ignored.

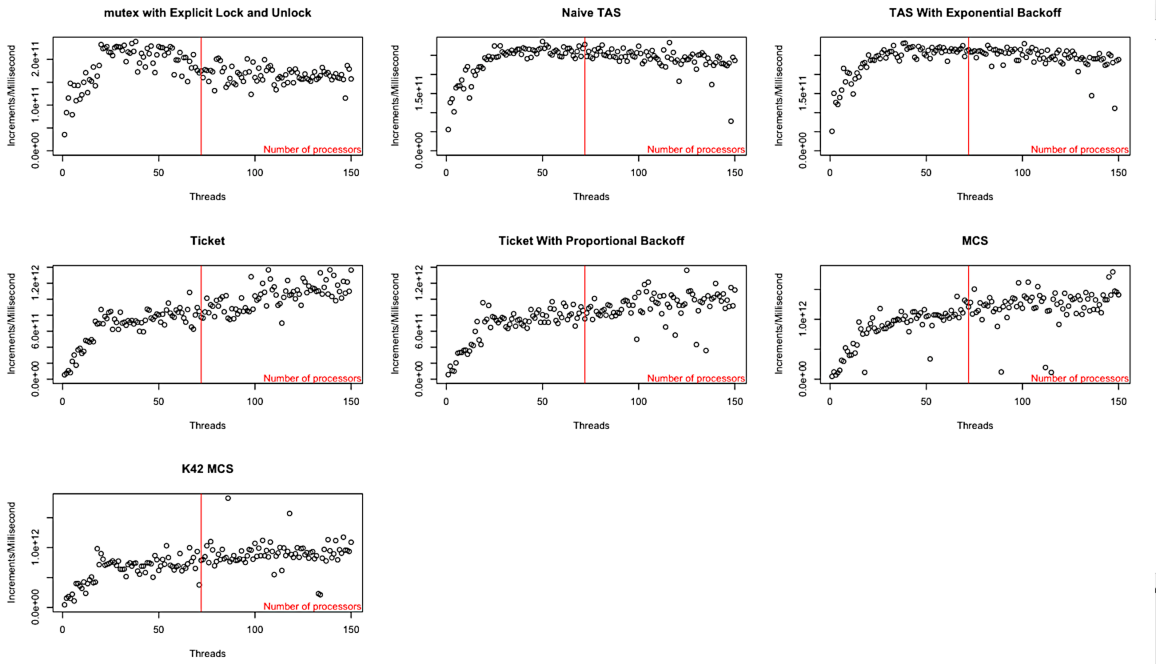
Experiment

The counter is incremented using 7 different spin locks on each run of parcount.

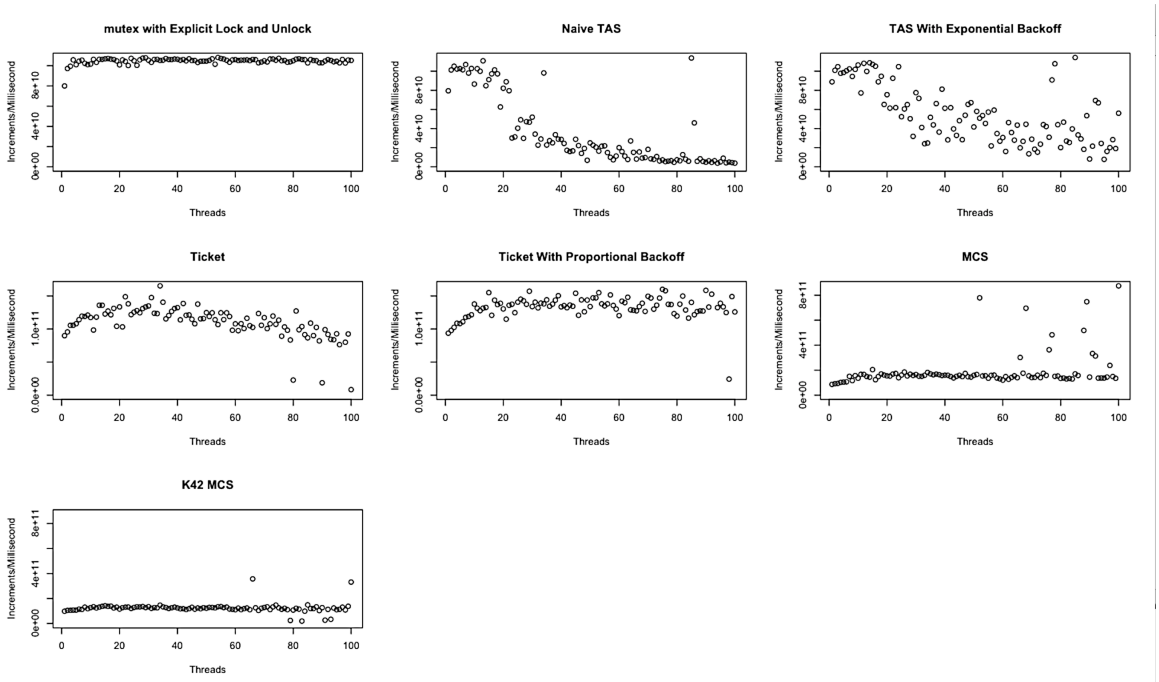
1. C++ mutex
2. Naïve test and set (TAS) lock
3. TAS lock with exponential backoff (base = 1, cap = 4, multiplier = 1.45)
4. Naïve ticket lock
5. Ticket lock will proportional backoff (base = 1.25)
6. MCS lock
7. K42 MCS lock

Results

The results after running parcount on node2x18a.csug.rochester.edu (72 processors), varying t between 1 and 150, and fixing i at 10,000 are shown below.



The results after running parcount on node-ibm-822.csug.rochester.edu (160 processors), varying t between 1 and 100, and fixing i at 10,000 are shown below.



We will refer to Increments/Millisecond as throughput.

Average throughput rankings (from greatest to least) for depicted data on node2x18a.csug.rochester.edu are shown below:

1. MCS lock
2. Naïve ticket lock
3. Ticket lock with proportional backoff
4. K42 MCS lock
5. TAS lock with exponential backoff
6. Naïve TAS lock
7. C++ mutex

Average throughput rankings (from greatest to least) for depicted data on node-ibm-822.csug.rochester.edu are shown below:

1. MCS lock
2. Ticket lock with proportional backoff
3. K42 MCS lock
4. Naïve ticket lock
5. C++ mutex
6. TAS lock with exponential backoff
7. Naïve TAS lock

Conclusions

Of the spin locks implemented, the MCS locks performed best in terms of throughput on both machines. On both machines its K42 counterpart performed less efficiently. The MCS lock performs best when each thread provides its own qnode.

The ticket lock algorithms performed second most efficiently. The ticket lock with proportional backoff saw more relative throughput that the ticket lock on node-ibm-822.csug.rochester.edu than on node2x18a.csug.rochester.edu. This can be attributed to the more relaxed memory model of node-ibm-822.csug.rochester.edu.

The C++ mutex performed better than the class of TAS locks on node-ibm-822.csug.rochester.edu than on node2x18a.csug.rochester.edu. This is because TAS lock threads request exclusive bus use each time they spin on TAS. This scales performance down linearly on node-ibm-822.csug.rochester.edu as seen on the above graphs.