CSCI 6360: Parallel Computing Lecture Summary - 16

Anirban Das (dasa2@rpi.edu)

March 23, 2018

Summary on mutex part of 'Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors' paper:

In this paper the authors shows new methods to improve upon the usual busy wait techniques used for achieving mutual exclusion or barrier synchronization . They show that it is possible to implement busy wait synchronization algorithms which pose no memory or interconnect contention in shared memory parallel system. This can be directly related towards achievement of mutual exclusion in a multi threaded environment.

A busy wait schedule is when processes poll the state of a shared variable to determine when to proceed and can be of two types: spin locks and barriers. The authors study several types of spin locks from existing literature and puts forward their own implementation.

`test_and_Set` lock requires a polling loop to access a boolean flag which indicates whether the lock is held or not, a processor acquires a lock by changing the flag state from false to true. For larger number of processors, this results in contention for the flag at the same time, which may be attenuated by adding a random delay/back-off in polling for every processor. `Ticket lock` can reduce the number of `fetch_and_`ϕ operations where a processor acquires and releases the lock by using `fetch_ and_increment` operation and a request and release counter. However ti still caused memory and network contention because it keeps on polling a common location and can be somewhat fixed by using a random polling delay proportional to the *minimum* time a processor can hold a lock.

To achieve constant bounds on network transactions, array based queuing locks can be used where the idea is that each processor spins on a different location in a different cache line by using atomic operations to get the location of where to spin. And finally the authors put forward their own MCS lock, which guarantees FIFO ordering of lock acquisition. Each processor spins on its locally accessible flag, and each processor hold address of record for process behind that will execute after lock release, almost like a linked list.

Performance analysis on the BBN Butterfly 1 machine with empty critical sections reveals that `text_and_set` and ticket achieves poorest scaling. Using exponential back-off for `test_and_set`, proportional for ticket lock vastly improves the raw performance. However, MCS scaling behavior outperforms all the others.