An Experimental Comparison of Concurrent Data Structures

Mark Gibson - 10308693

What is the Problem?

- Plenty of work done on how to implement concurrent objects
- Not much data on comparing the different types of concurrent data structure
- Locked, Lock-free, Wait-free

Why is it Important?

- Potential for high scalability
- Increased Speeds
- Know when to apply different locking techniques

Others' Work

- "The Art of Multiprocessor Programming"- Herlihy & Shavit - 2008
- "Designing Concurrent Data Structures" – Moir & Shavit - 2001
- "Implementing Concurrent Data Objects" – Herlihy - 1993

What I have Done

- Implemented 3 concurrent data structures
- Implemented both locked and lock-free variations
- Tested & compared them on 3 different systems

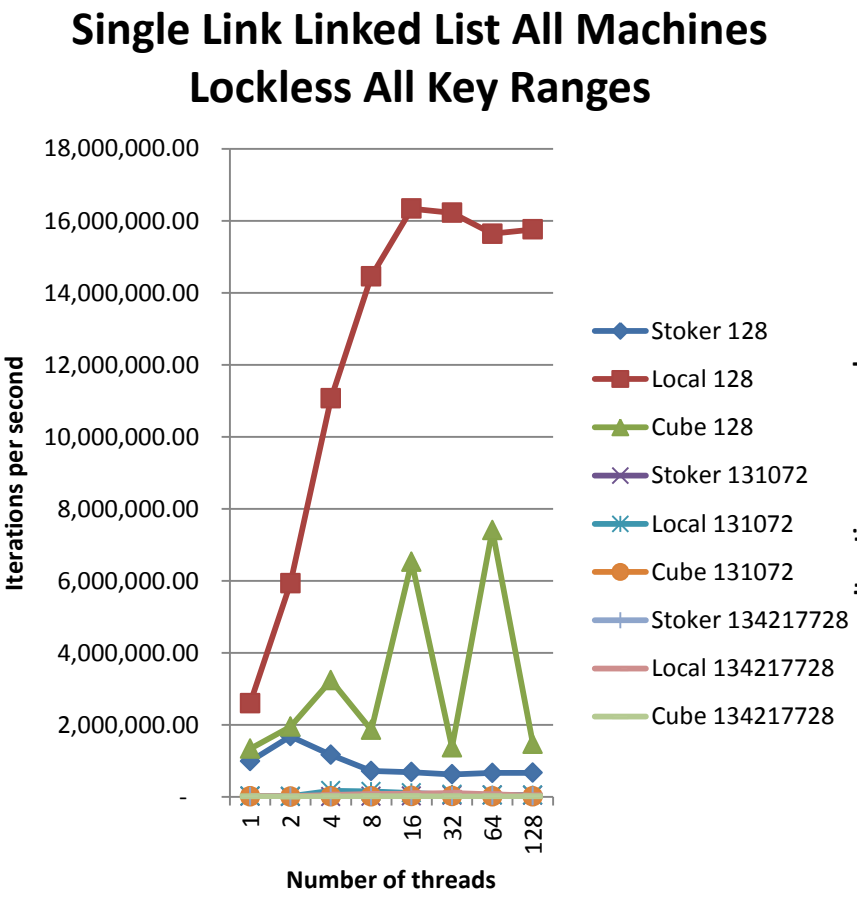
Data Structure Variations

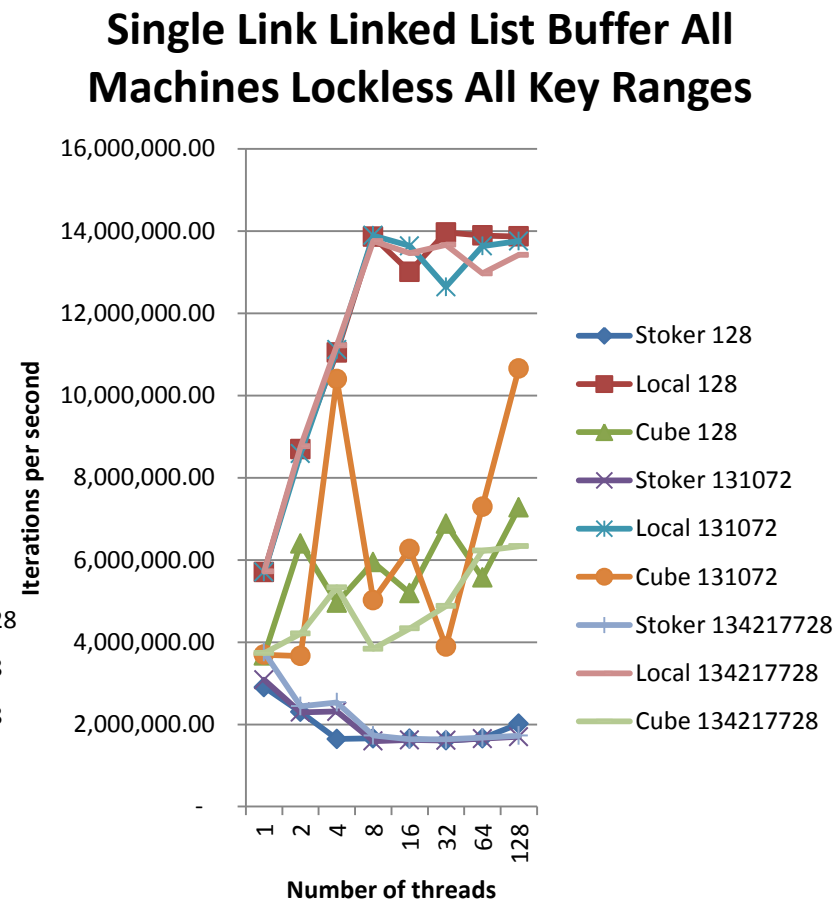
- MPMC Hash Table Closed Addressing
  - Locked Table
  - Lock per Bucket
  - Lockless
- MPMC Linked List
  - Single Link Regular Locked/Lockless
  - Double Link Buffer Locked/Lockless
  - Single Link Buffer Locked/Lockless
- Ring Buffer
  - MPMC Locked
  - SPSC Lockless

Evaluation

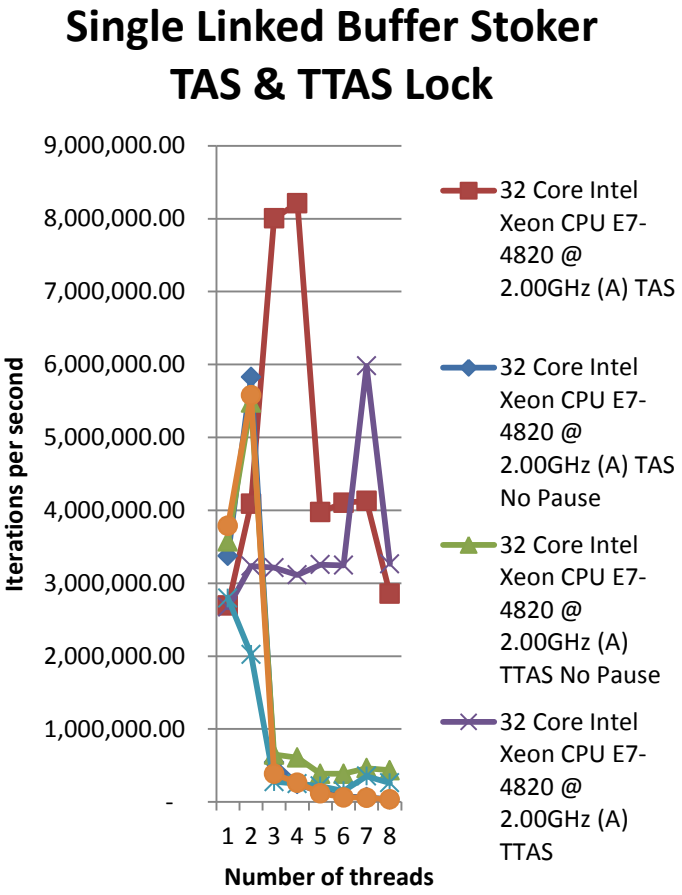
- Ran each variation on each machine for multiple thread counts
- Thread count went from 1-128
- Recorded iterations per second against number of threads
- Used Perf to record cache misses, cpu cycles etc…

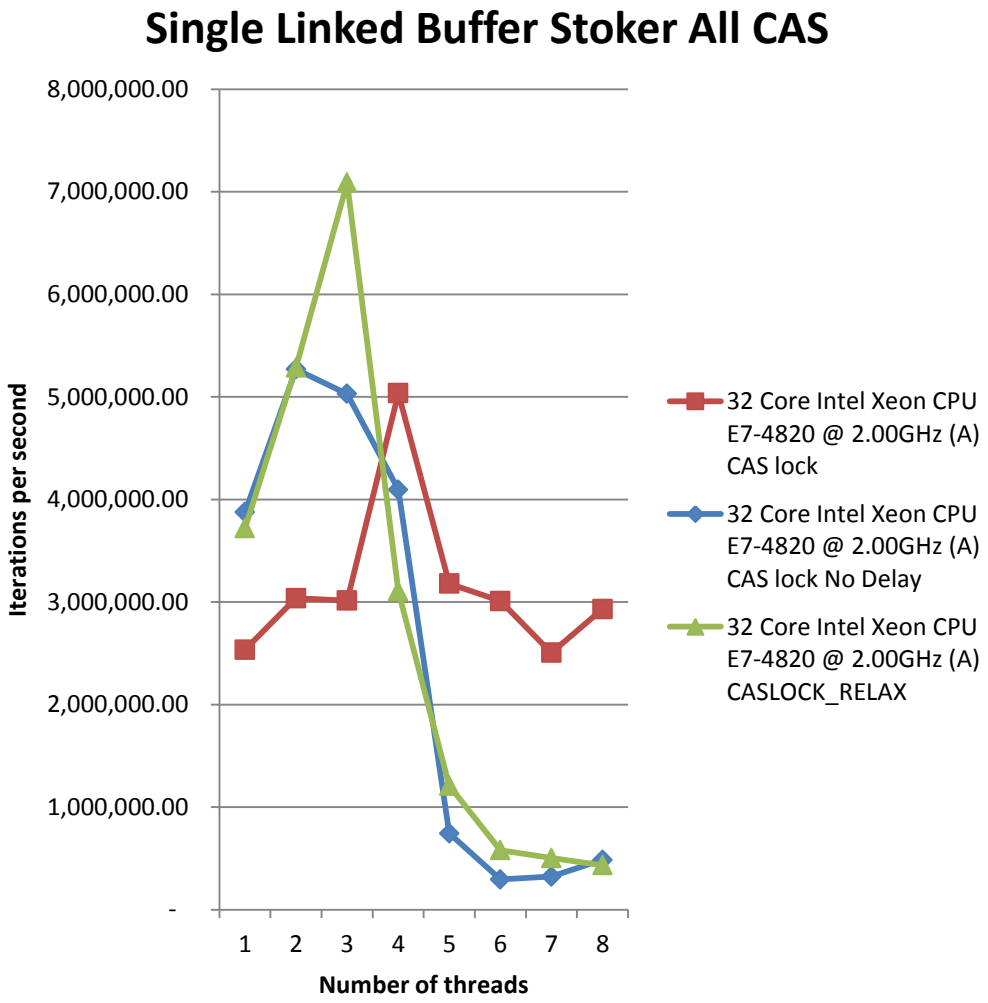
Results & Analysis





Results & Analysis





What have I learned?

- Never assume results
- Lockless algorithms do not guarantee better speeds
- The C++11 atomic library makes it easier to implement lockless algorithms