An Experimental Comparison of Concurrent Data Structures

Mark Gibson - 10308693

What is the Problem?

- Concurrent Data Structure – Designed for access by multiple threads
- Plenty of work done on how to implement concurrent data structures
- Not much data on comparing the different types of concurrent data structure

Locking Algorithms

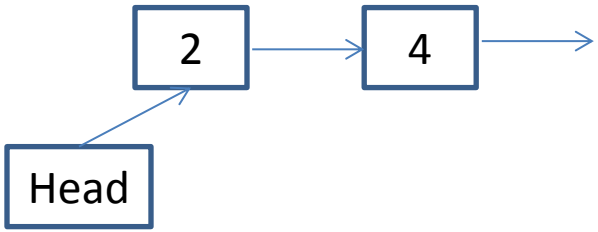
- Locked – Uses mutexes, semaphores to acquire a lock, blocks threads that do not have the lock
- Lock Free – Uses atomic instructions such as compare-and-swap. Guarantees system-wide throughput with the chance of starvation
- Wait Free – Similar to lock free but is also starvation free

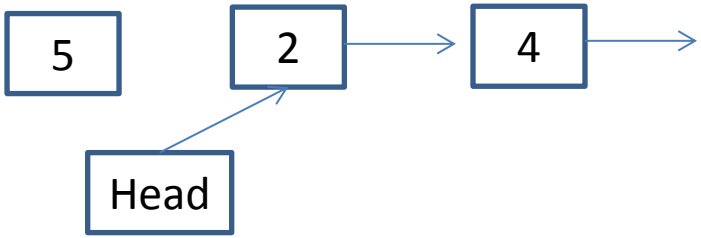
Atomic Instructions

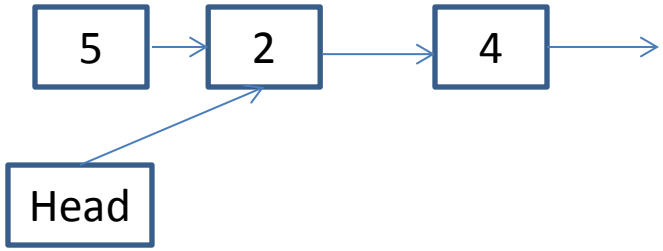
- Either complete fully or not at all
- Are used to implement locked and lockless algorithms
- Example: compare-and-swap instruction
- It compares the contents of a memory location to a given value and, only if they are the same, modifies the contents of that memory location to a given new value

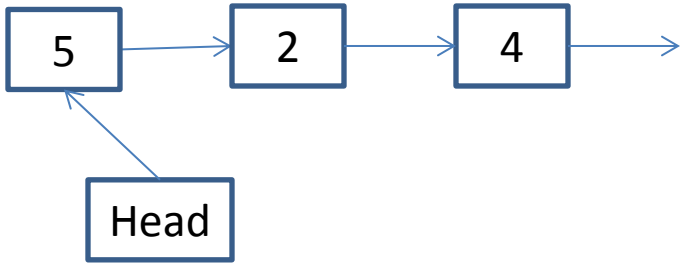
Atomic Instructions

- Can use compare-and-swap to implement a lockless list by atomically changing which node Head points to









Atomic Instructions

- The code for this is as follows:

```
if(std::atomic_compare_exchange_strong(&tmpList->head, &tmpHead, newNode))
{
        //Atomic operation successful
}
```

- Similarly we can use compare-and-swap to implement a lock which will allow us to do the same thing

```
while(true){
        if(lock.compare_exchange_weak(0, 1))break;
        _mm_pause();
}
//Change head pointer to new node
```

Why are Concurrent Data Structures Important?

- Potential for high scalability
- Better performance than serial implementations of the same data structure

Background Work

- Some of my references:
- “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
  - Ring Buffer
  - Linked List
  - Hash Table
- Implemented both locked and lockless variations
- Tested & compared them on 3 different systems
  - My Local Machine (Intel, 4 Cores @ 3.30GHz)
  - Stoker (Intel, 32 Cores @ 2.00GHz)
  - Cube (Intel, 16 Cores @ 2.27 GHz)

Data Structure Implementation

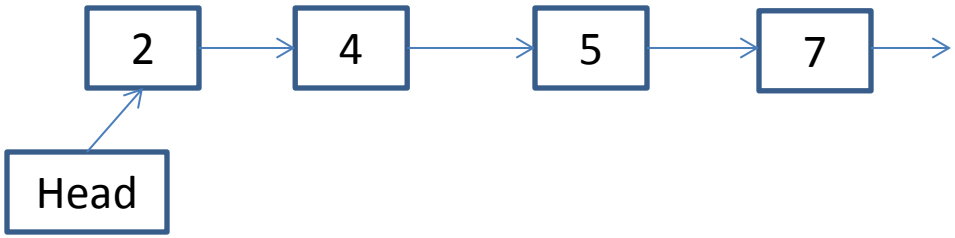
- Locked Variations – mutex, test-and-set, compare-and-swap
- Lockless Variations – C++11 atomic library operations
- Mostly Multi Producer Multi Consumer (MPMC) with one Single Producer – Single Consumer (SPSC)

Ring Buffer

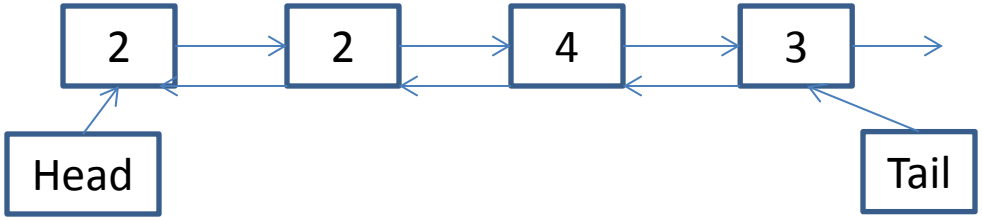
- Used to get to grips with the C++11 library
- MPMC Locked Variation
- SPSC Lockless Variation

Linked List

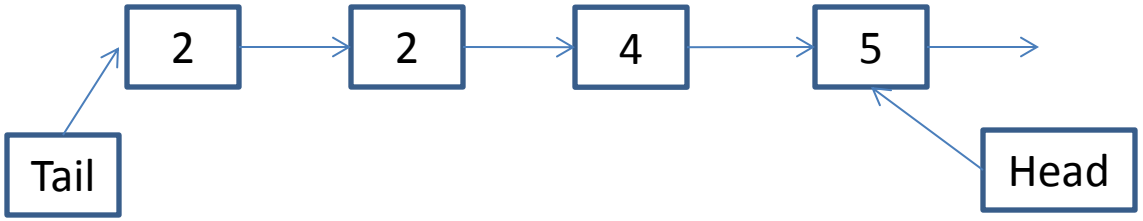
- MPMC Singly Linked List – Locked/Lockless



- MPMC Doubly Linked Buffer – Locked/Lockless



- MPMC Singly Linked Buffer – Locked/Lockless



Hash Table

- Closed Addressing – Collisions are added onto a linked list
- Contains/Add/Remove/Resize
- MPMC Globally Locked Hash Table
- MPMC Lock Per List
- MPMC Lockless

System Details

- My Local Machine (Intel, 4 Cores @ 3.30GHz)
- Stoker (Intel, 32 Cores @ 2.00GHz)
- Cube (Intel, 16 Cores @ 2.27 GHz)

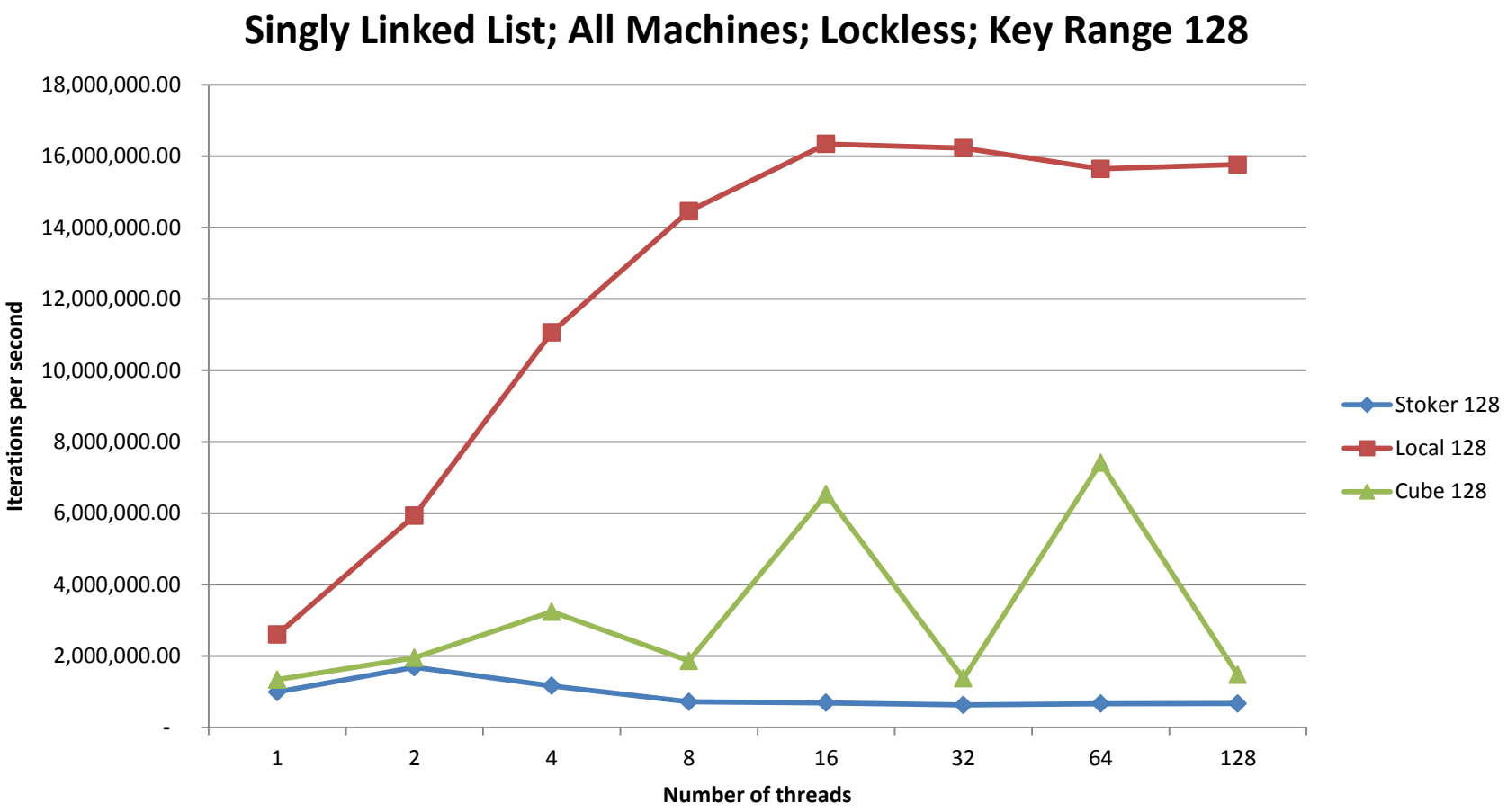
Evaluation

- Ran each variation on each machine for multiple thread counts, varying maximum list size for example
- Thread count went from 1-128
- Recorded iterations per second against number of threads
- Used Hardware Performance Counters, special registers used for performance analysis, to record cache misses, cpu cycles etc…

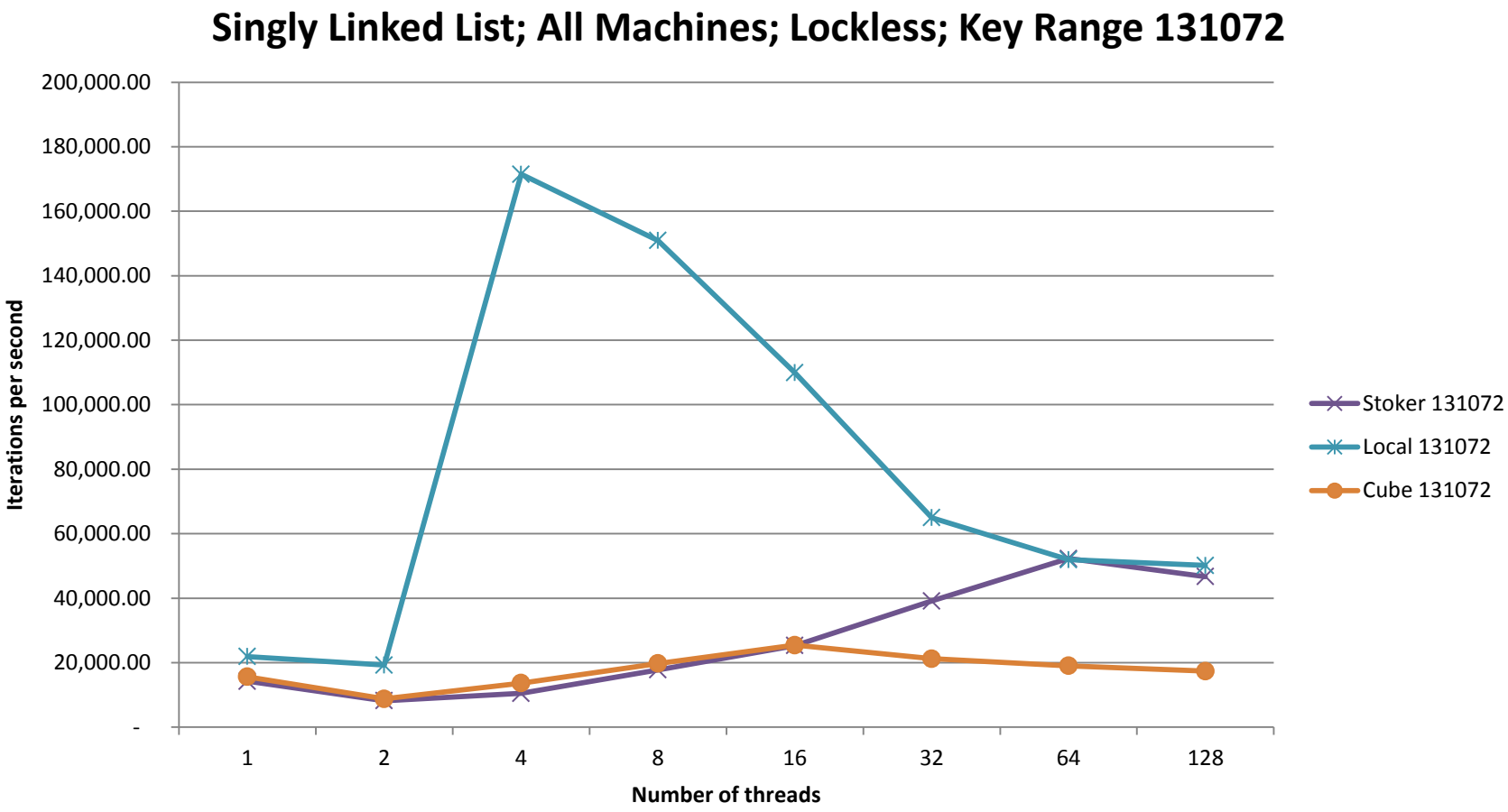
Results & Analysis

- For each graph:
- X-axis: Number of threads
- Y-axis: Iterations per second
- Each line of data, such as Stoker 128 represents the machine that the data was collected from and in this case, for the linked list, the maximum number of nodes allowed in the list

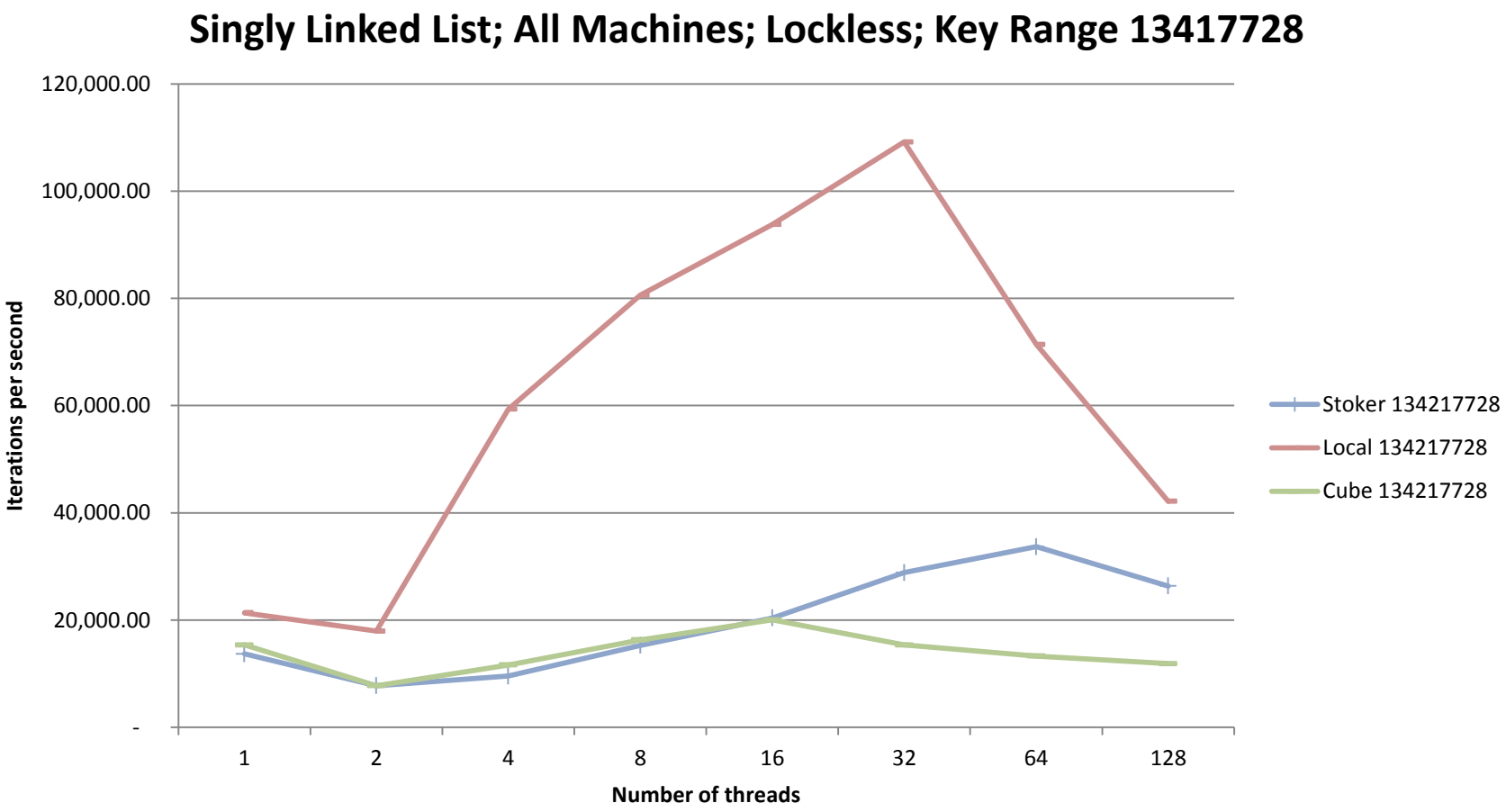
Results & Analysis



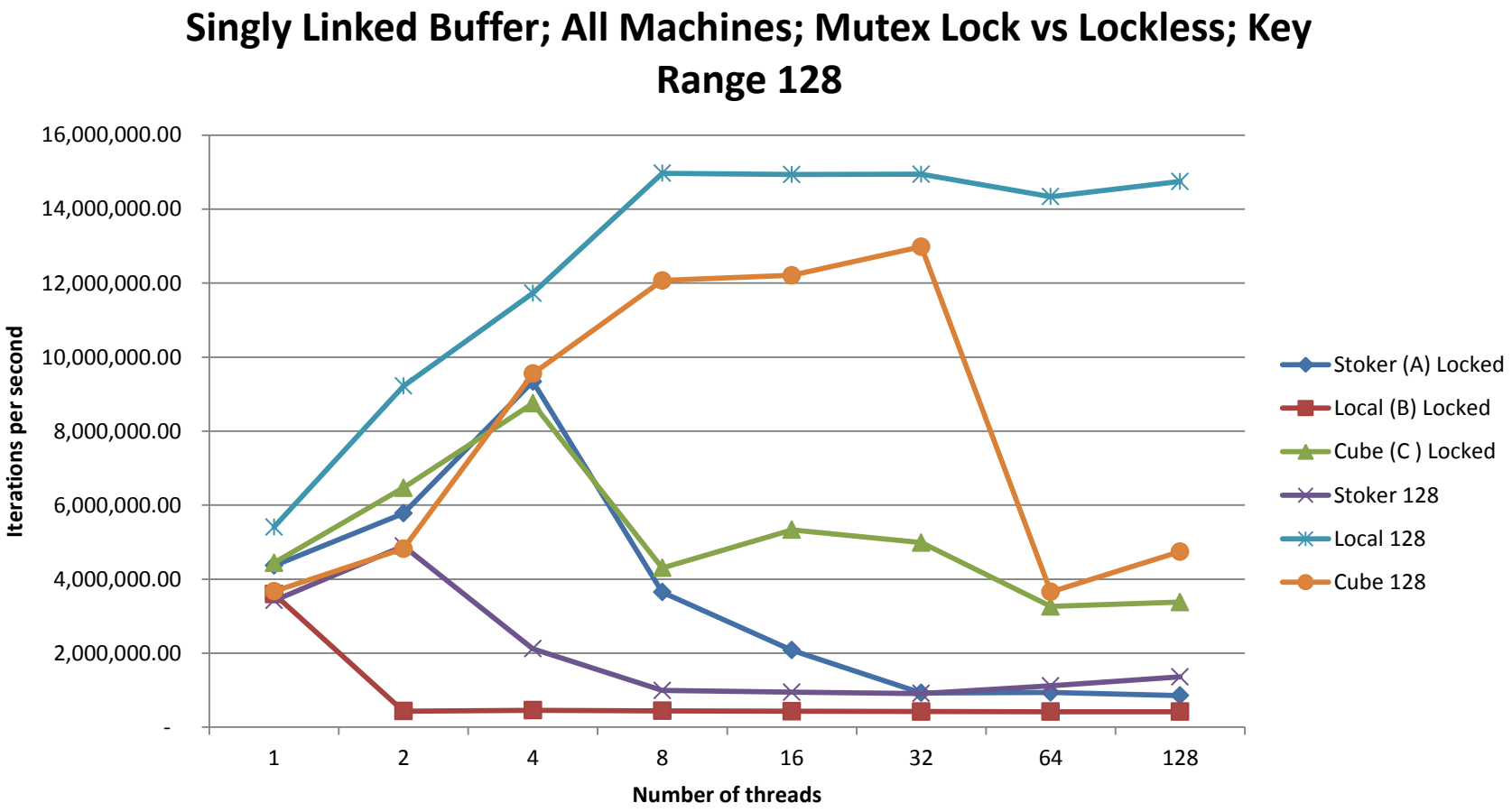
Results & Analysis



Results & Analysis



Results & Analysis



Results & Analysis

- Sample of Hardware Performance Counter Data from previous slide:

	Cube 128	Cube Locked
Stalled Front End Cycles	8,603,339,371	28,157,586,014
Stalled Back End Cycles	4,290,118,281	19,235,301,044
Faults	98,644	177,612
CPU Cycles	16,644,060,219	37,007,949,115

What have I learned?

- Never assume results, I expected that Stoker (32 Cores) would always beat my local machine (4 Cores) but this was not always the case, as seen from the graphs in the previous slides
- Lockless algorithms do not guarantee better speeds