An Experimental Comparison of Concurrent Data Structures

Mark Gibson - 10308693

What is the Problem?

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

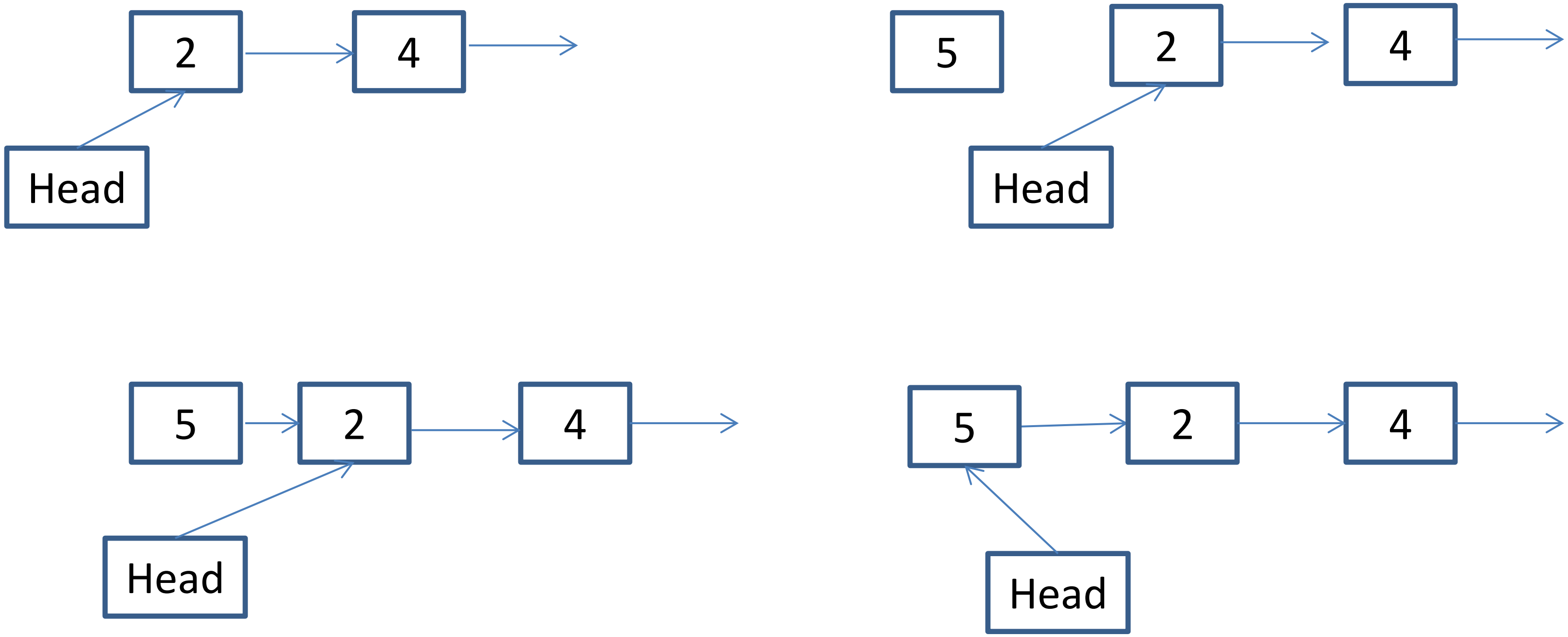
Atomic Instructions

- Either complete fully or not at all
- Are used to implement locked and lockless algorithms
- Example: compare-and-swap instruction

```
    If(*lock == 0)
    {
        *lock = 1;
    }
```

Atomic Instructions

- Can use compare-and-swap to atomically add a node to the head of a linked list:



Atomic Instructions

- Compare-and-swap can be used locklessly as follows:

```
if(std::atomic_compare_exchange_strong(Atomic *Obj, *Expected, Desired))
{
		//Value of Obj was equal to Expected, now equal to Desired
}
```

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

```
while(true){
		if(std::atomic_compare_exchange_weak(Atomic * Obj, 0, 1))break;//Try and acquire lock
		_mm_pause();
}
//Lock acquired, change value of head to new node
```

Locking Algorithms

- Locked
  - Uses mutexes and 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

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
  - Locklessinc.com

What I have Done

- Implemented 3 concurrent data structures
  - Ring Buffer
  - Linked List
  - Hash Table
- Implemented both locked and lockless variations
- Compared them on 3 different systems

Data Structure Implementation

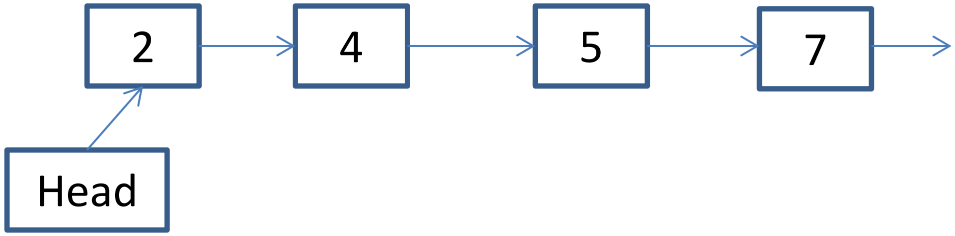
- Locked Variations
  - Mutex
  - Test-and-set
  - Ticket Lock
- Lockless Variations
  - C++11 atomic library operations
    - Atomic Compare-and-swap
    - Atomic Fetch-and-add
- Multi Producer Multi Consumer (MPMC)
- Single Producer Single Consumer (SPSC)

Ring Buffer

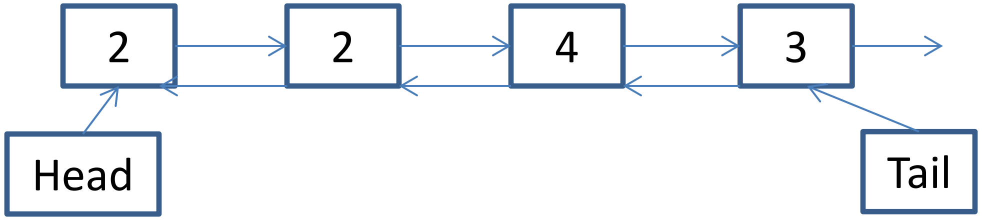
- Used to get to grips with the C++11 library
- Variations
  - MPMC Locked
  - SPSC Lockless

Linked List

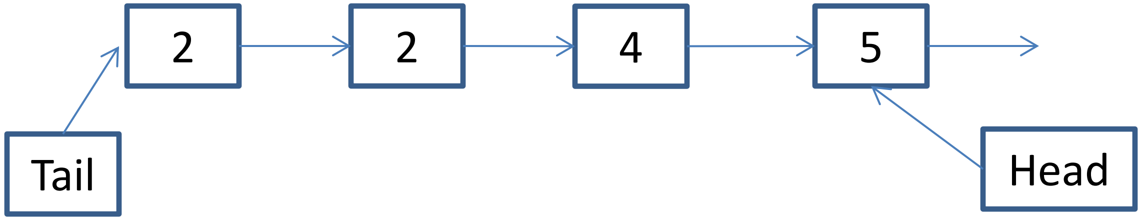
- MPMC Singly Linked List – Locked/Lockless



- MPMC Doubly Linked Buffer – Locked/Lockless



- MPMC Singly Linked Buffer – Locked/Lockless



Hash Table

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

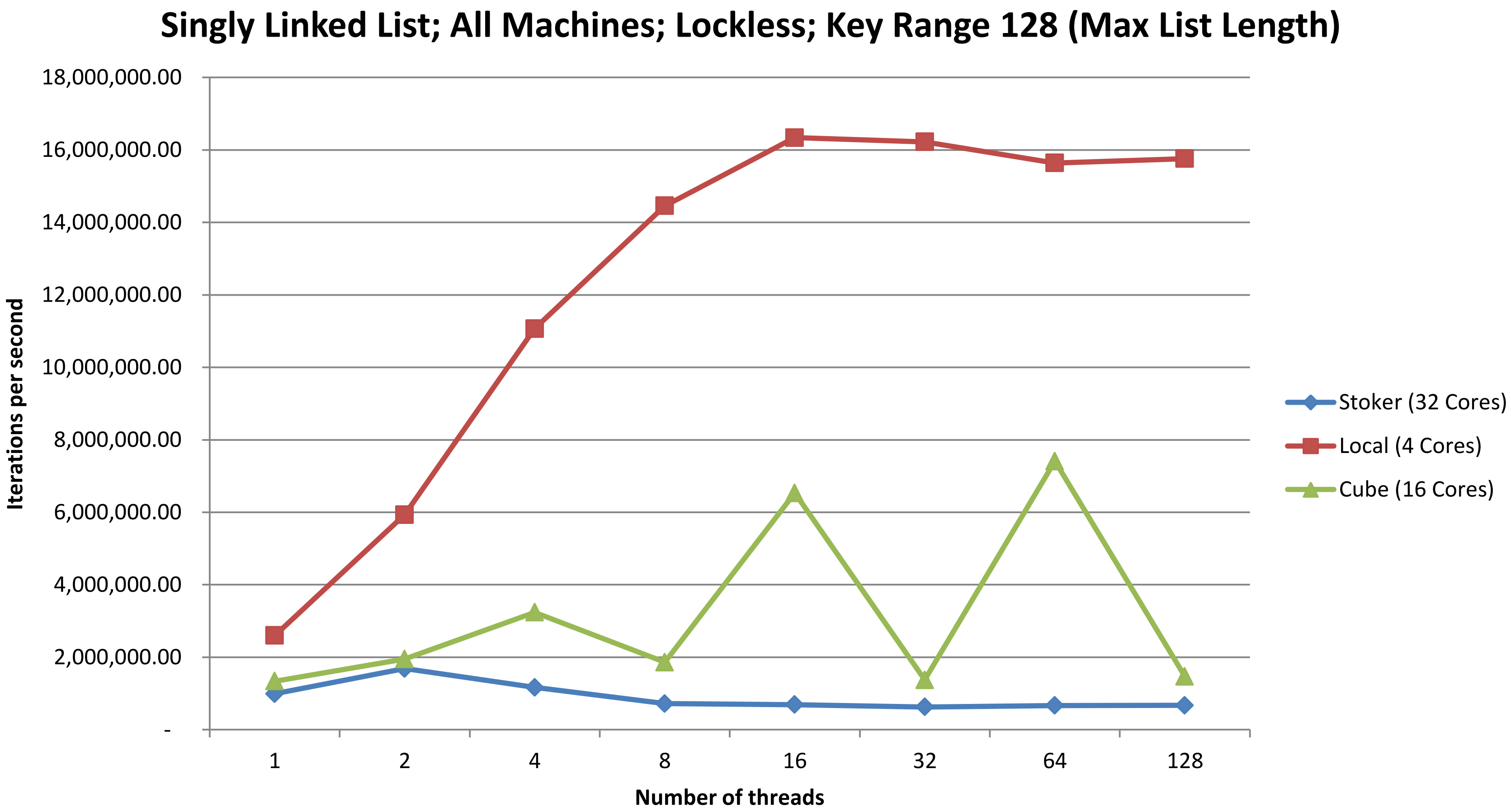
System Details

- My Local Machine (Sandy Bridge 32nm, 4 Cores @ 3.30GHz)
- Stoker (Ivy Bridge EX 22nm, 32 Cores @ 2.00GHz)
- Cube (Gainestown 45nm, 16 Cores @ 2.27 GHz)

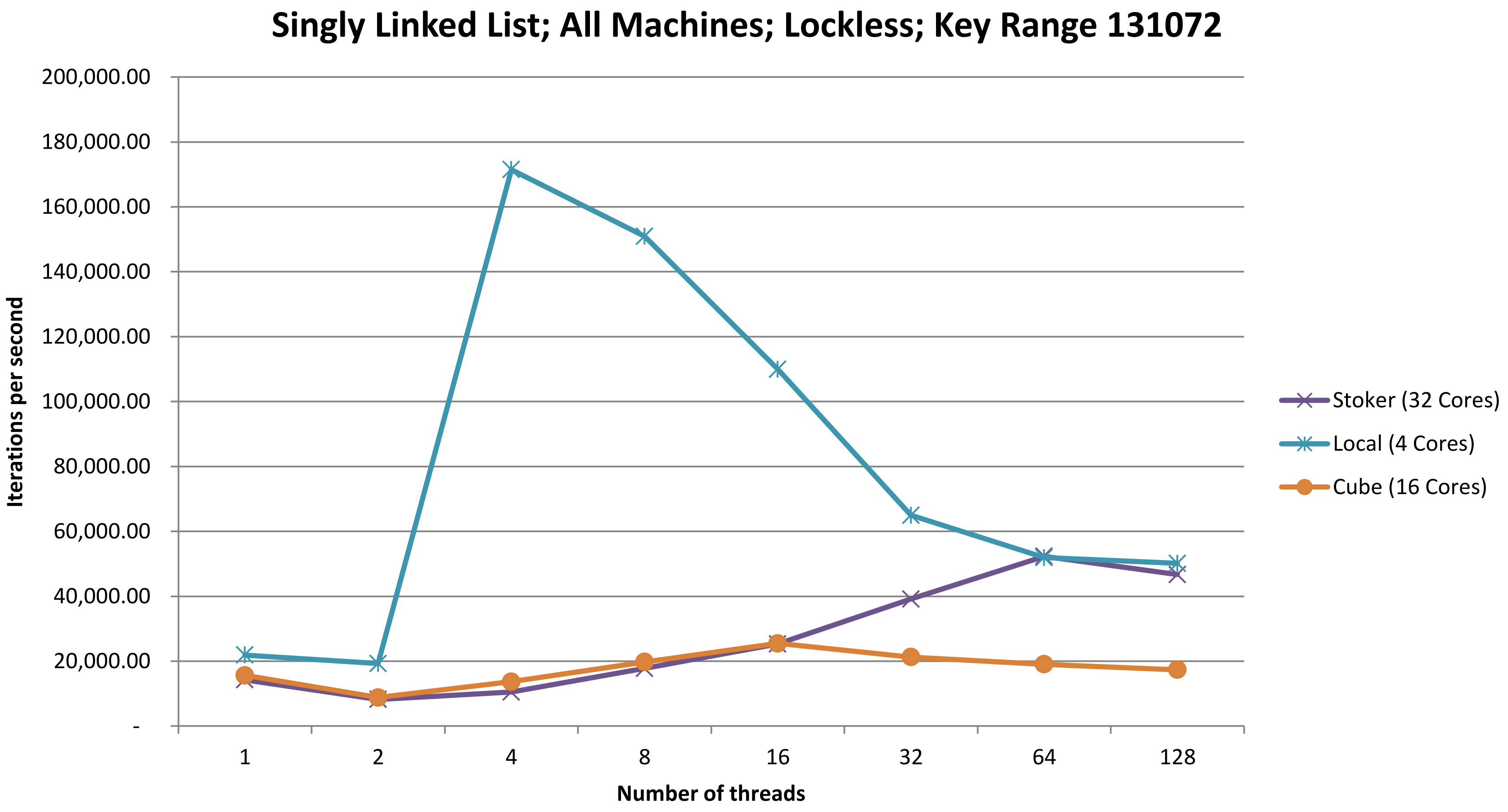
Evaluation

- Compared Data Structure Variations
  - Thread Count 1-128
  - Varied list, buffer and table sizes
  - Locked vs Lockless
- Used Hardware Performance Counters
  - Special registers
  - Performance analysis
  - Record cache misses, cpu cycles etc…

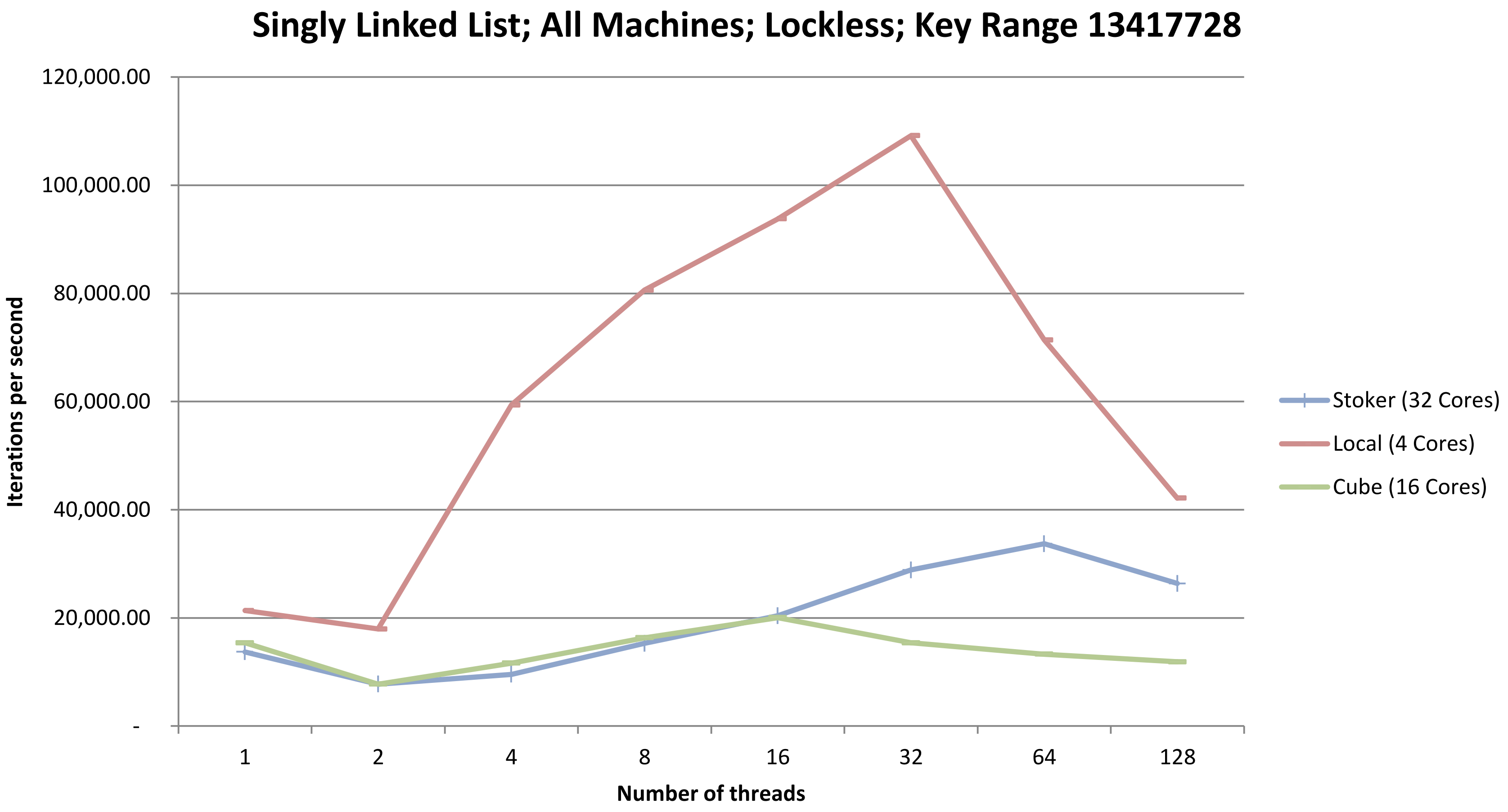
Results & Analysis



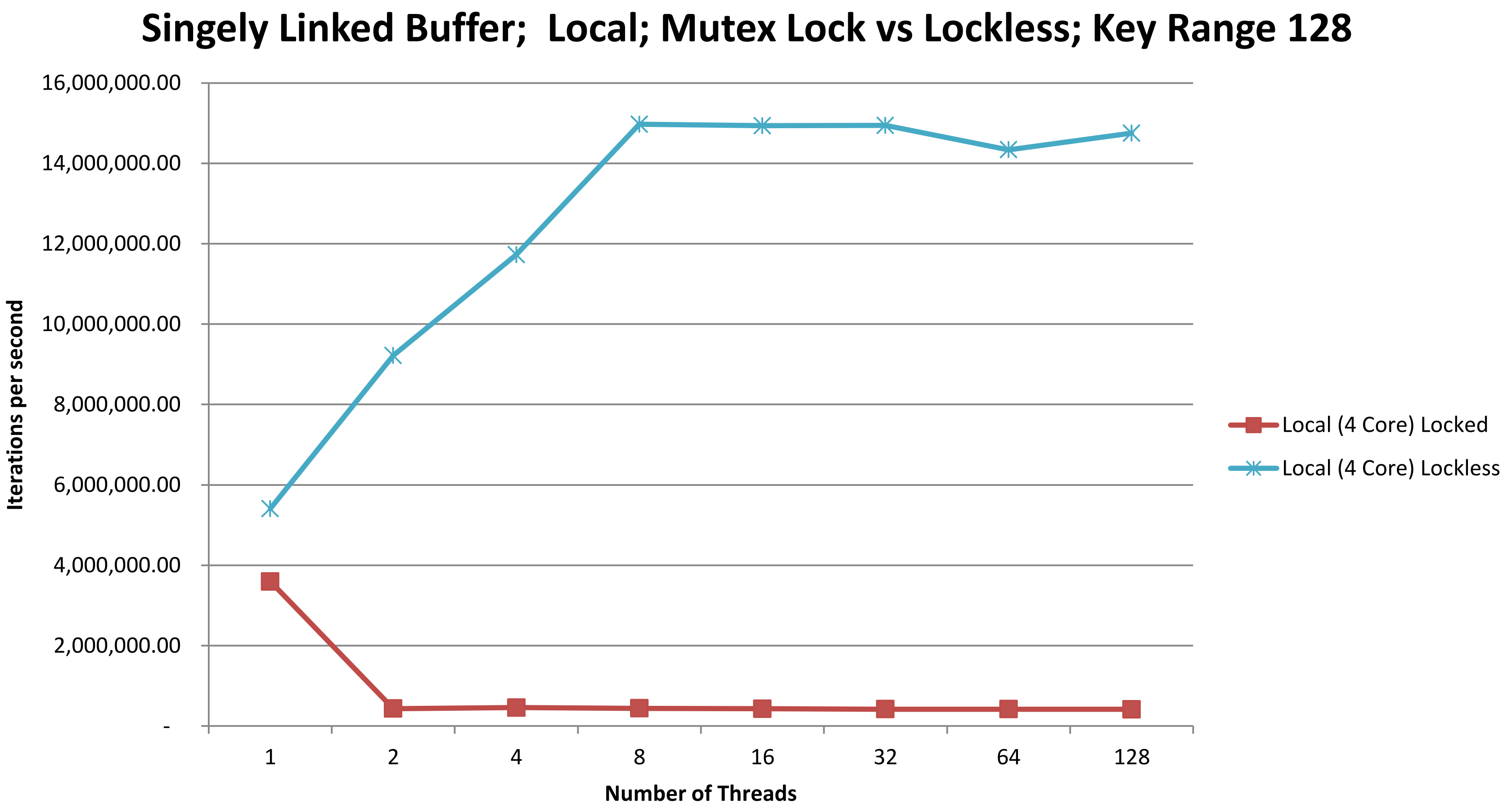
Results & Analysis



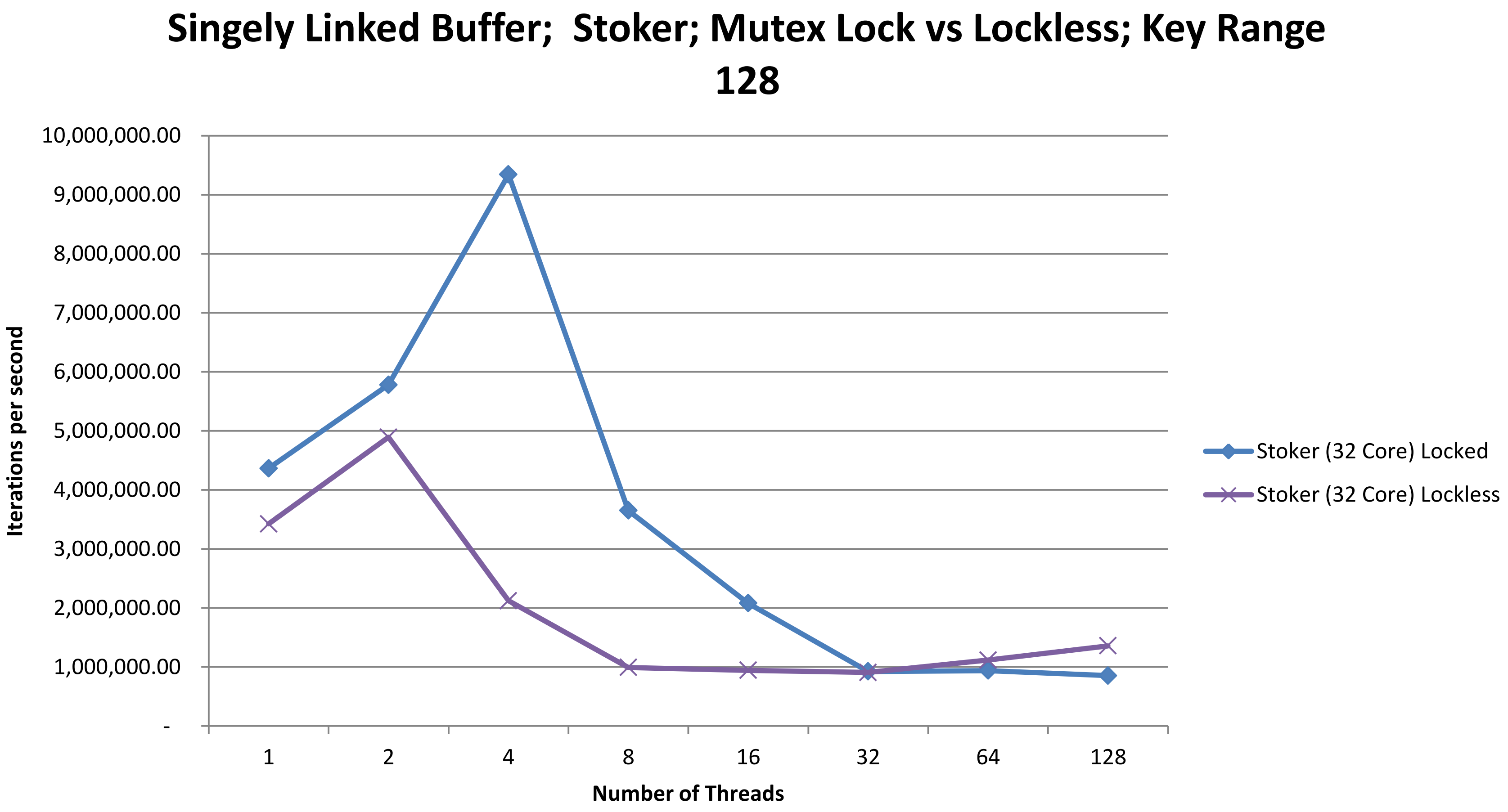
Results & Analysis



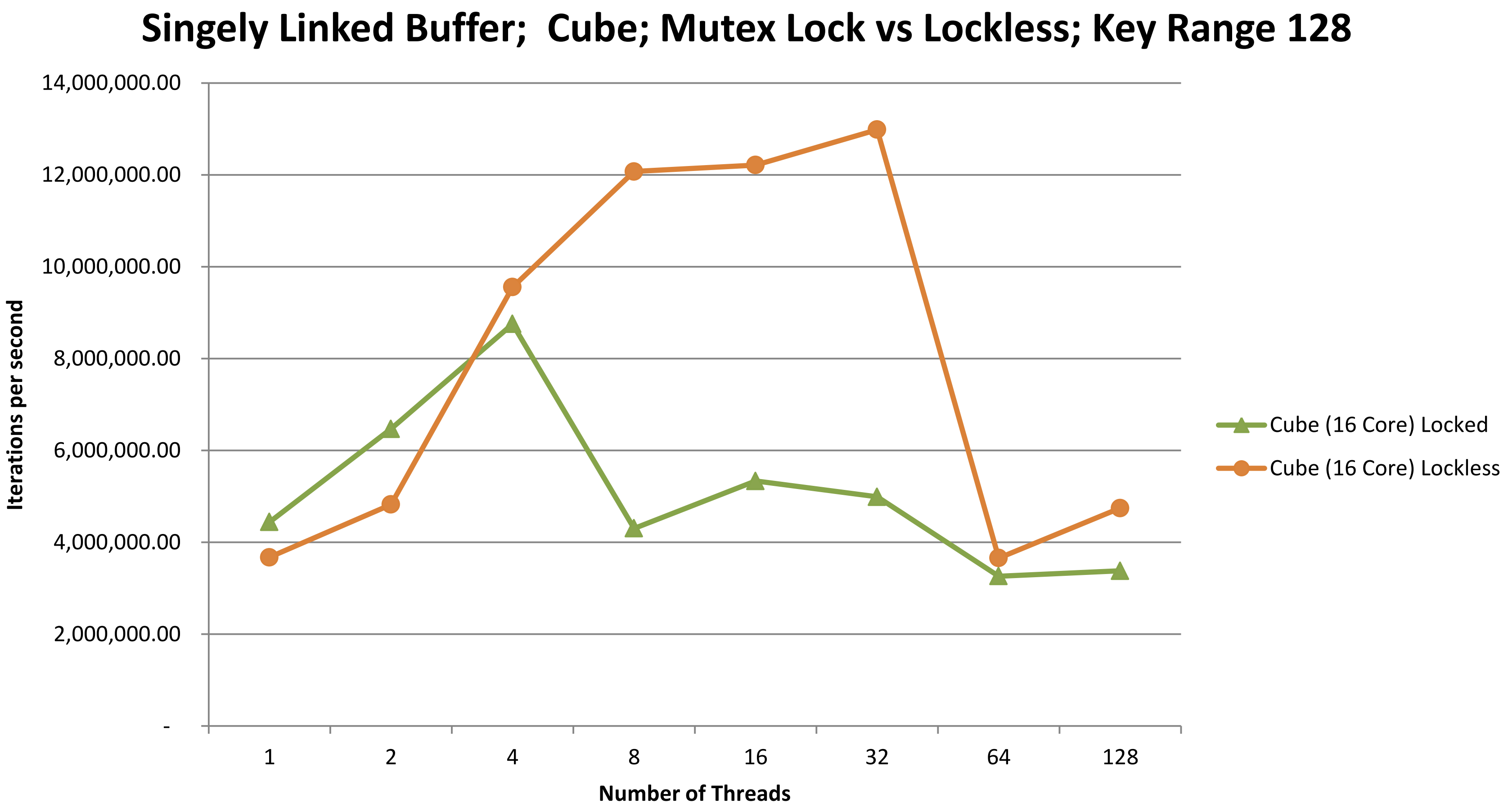
Results & Analysis



Results & Analysis



Results & Analysis



Results & Analysis

- Sample of Hardware Performance Counter Data from previous slide (Cube):

	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
CPU Cycles	16,644,060,219	37,007,949,115

What have I learned?

- Lockless algorithms
  - More Difficult to Design & Implement
  - Generally provide a performance boost when compared to locked algorithms
  - There are always exceptions

What have I learned?

- Ring Buffer
  - Locked variations proved to be mostly slower than lockless varieties
  - Exceptions came in the form of the TAS and TTAS locks running on the Local Machine
  - Stoker proved to be the fastest of the three machines with the lockless algorithm

What have I learned?

- Linked List
  - Local Machine performed well locklessly, outperforming the other two machines
  - Local Machine outperformed every lock on all 3 variations with the lockless variation
  - Gained the largest performance boost by using a lockless algorithm out of the three data structures

What have I learned?

- Hash Table
  - Globally Locked and Lock per Bucket performed relatively equally
  - TestAndSet lock seems to be well suited to my implementation with consistently good performance across variations
  - Surprisingly, both locked variations outperformed the lockless variant by a sizeable margin