Bakery lock & Fair Spin lock:
The bakery lock and the fair spin lock had extremely similar results. Both have excellent throughput when there was only one producer and one consumer. This, as with all the spin locks, is because the CADE machines have two cores; the producer is ran on one while the consumer is ran on the other. Since they are ordered locks this means they probably generally alternate access to the queue either each entry or when the queue is full/empty. The ordered nature of the lock is also what accounts for the small standard deviation. With more threads however these are horrible lock methods. This is because it is an ordered spin lock. If you assume that many threads try to obtain the lock at the same time one will succeed and when it loops back around it will always have to wait the remainder of its time slice. Since it doesn't yield the processor many cycles go wasted.

Spin lock:
This lock has the best performance with a small number of threads. It benefits from the same things as the other spin locks but doesn't bog down as fast when adding threads since it is not ordered and has no sense of fairness. Since it is not ordered it doesn't end up wasting as many cycles; once the lock acquisition loop comes around it is free to acquire it again and again for the whole of its time slice. However this causes a wide standard deviation as seen in Figure 2. It is important to note that some amount of the standard deviation is due to one thread getting a head start at producing/consuming since they are created sequentially.

Pthread Mutex:
The Pthread mutex's were by far the most consistent. With only one producer/consumer it doesn't fair as well as the others. This is partly due to the overhead of sleeping/waking processing when acquiring the lock. It is easy to see in Figure 1 how much better this is than a spin lock. Both throughput and the standard deviation begin to accel beyond any version of the spin lock as more threads are added. Since the Kernel is involved the system is able to regulate fairness and can properly wake a thread that is ready, and more importantly able, to take the lock. With more threads this leads to fewer wasted cycles.

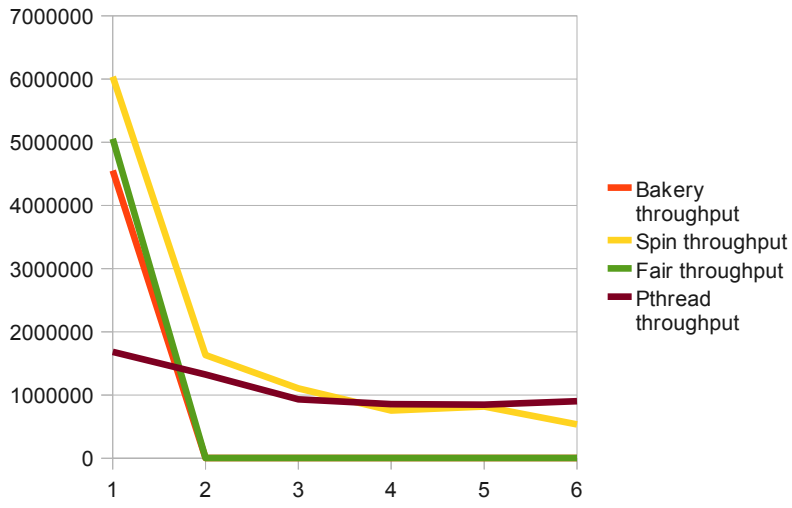

Figure 1: Consumer Threads vs. Throughput

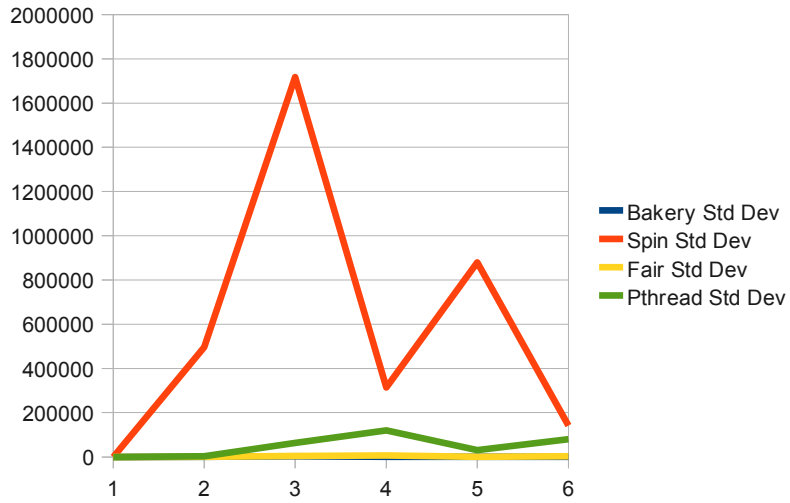

Figure 2: Consumer Threads vs. Standard Deviation