**Paper Review Writeup**

**CS 554: Data Intensive Computing**

Reviewer Name- Nilesh Balu Jorwar (A20405042)

Paper Title- B-Queue: Efficient and Practical Queuing for Fast Core-to-Core Communication

Paper Authors- Junchang Wang, Kai Zhang, Xinan Tang, Bei Hua

Publication Venue- International Journal of Parallel Programming (2013) 41:137–159 DOI 10.1007/s10766-012-0213-x

Year of Publication – February 2013

**Summary of the Paper**

Increase in number of cores in today’s multicore system architecture significantly speed up certain operations which in turn requires the need for faster core to core communication. The authors of this paper presented implementation of faster and efficient data structure i.e. B- Queue.

The authors talked about the efficient and practical queuing for the core to core communication that is an efficient and practical single-producer-single-consumer concurrent lock free queue that solves the deadlock problem gracefully by introducing a self-adaptive backtracking mechanism.

They presented the use of CLF queues in building of network processing system for the parallelism and the evaluation of CLF queues done on dummy and real applications and concluded to focus onto real applications where attention must be paid. Authors introduced faster data structure i.e. CLF queue (B-Queue) to achieve scalability and stability with no parameter tuning and provided the remaining paper with entire details on B-Queue.

Authors presented background of various queues with their limitations. Strong ordering requirement in computation is needed in network processing and stream processing systems where pipeline parallelism is applicable and is achieved using CLF queues. Lock based queues were inefficient which was replaced by the Lamport queue which uses two shared variables where locks are removed from single-producer-single-consumer queues but suffers cache thrashing. The use of FastForward to eliminate the shared variables between consumer and producer also resulted in cache thrashing and needed manual intervention to tune the parameters to achieve peak performance. However, cache thrashing was handled using Multi-Line Cache update and MCRingBuffer which in turn was resulted into deadlock situation. Various deadlock prevention methods were also discussed but none of them were practical to use.

**Core Contributions**

* Evaluation of existing CLF queues on both real and dummy applications.
* Proposed the powerful CLF queue i.e. B- Queue to achieve scalability and stability in distributed environment.

**Problems addressed**

* Overcome the dependency on auxiliary hardware or software timers to handle deadlock problem when batching is used.
* Improved performance in real testbeds.
* Facilitated the fast and efficient core-to-core communication mechanism on existing commodity multi-core platforms.
* Due to lack of efficient core to core communication, parallelization of network applications cannot take effect because a single memory-based core-to-core communication might offset the benefit from parallelization.
* Lock Free FIFO queues are inappropriate for applications featuring fine-grained parallelism. Hence, single-producer-single-consumer concurrent lock-free FIFOs (CLF queues) introduced to support fast core-to-core communication to avoid cache thrashing.
* In real applications, performance of CLF queues decreased drastically as the number of queues increased.

**Strengths and Weaknesses-**

* Batching in B- Queue reduces the number of shared memory accesses.
* Cache thrashing avoidance while facilitating hardware prefetching that improves performance of CLF queues.
* Easily used in real applications.
* A deadlock mechanism, backtracking is used to decrease batching size where no timer and monitoring thread is required thus reducing system complexity. A self-adaptive backtracking mechanism adjusts the batching size and provides trade-off between latency and performance as B- Queue suffers latency when producer is not busy.

**Things to improve-**

* Concurrent lock-free (CLF) queue algorithm does not take full advantage of CPU cache features to improve performance.
* To minimize inter-core communication overheads in pipeline parallelism and for cache-level optimization, we can implement a fast single-producer-single-consumer (SPSC) buffer scheduling queue that is a cache-friendly CLF queue scheduling algorithm (CFCLF). It avoids cache false sharing and cache consistency problem and implements batch processing to improve the throughput and prevents deadlocks.

**Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms**

**Claims of Paper**

In a multithreaded environment, threads communicate through a data structures such as queues, stacks, and so on. To facilitate correct and simultaneous access to data structures over multiple threads, data structures must be protected by a concurrent algorithm. If the algorithm that protects a concurrent data structure is blocking (using a thread suspension), then it is a blocking algorithm. If the algorithm that protects a concurrent data structure is non-blocking, then it is a non-blocking algorithm and the data structure is a non-blocking data structure.

Author proposed the non-blocking concurrent queue and two lock-queue.

Authors claimed that the properties of linked-list are held in implementing these data structures as linked-list is always connected followed by the insertion and deletion operations onto the list as per the queue properties.

They claimed to achieve the linearizability while claiming that lock-free algorithm implemented is non-blocking. They also presented two-lock algorithm which is Livelock-free and suggested this data structure to be used for machines that are not multi-programmed and does lack in universal atomic primitive such as compare-and-swap.

**Problem Addressed**

Authors listed and discussed various earlier proposed solutions for lock free algorithms for concurrent FIFO queues. They started by mentioning that Hwang and Briggs presented lock free algorithms based on compare-and-swap but failed to handle empty and single queues, concurrent enqueues and dequeues. Lamport came up with wait-free algorithm that restricts concurrency while Gottlieb et al. and Mellor-Crummey presented lock-free but non-blocking algorithms. Treiber’s non-blocking algorithm were inefficient while Herlihy and Prakash, Lee proposed non-blocking version of sequential and concurrent lock-based algorithms.

Authors also listed the work of other authors such as Massalin and Pu, Stone, Prakash, Lee and Johnson in implementing non-blocking concurrent queue. However, these algorithms were based on compare-and-swap and should have the ability to handle the ABA problem at the same time that made these algorithms blocking. Hence, authors of this paper introduced concurrent FIFO queue as non-blocking and queue that uses pair of locks.

**Contributions-**

Introduced non-blocking data structure that implemented the queue as singly-linked list and used compare-and-swap and modification pointers to avoid ABA problem. Also achieved reusability of dequeued nodes. Treiber’s non-blocking stack algorithm is used to implement non-blocking free list.

Authors also presented two-lock queue data structure that avoid deadlock problem arising from the processes trying to acquire the locks in different orders.

**Strengths and Weakness-**

The non-blocking concurrent queue data structure can be used in transaction safe versions of lock-free queue.

Non-blocking queue algorithm is a choice for multiprocessors that support universal atomic primitives.

Non-blocking atomic update algorithms outperforms all alternatives, not only on multi-programmed systems, but on dedicated machines as well.

Two-lock queue algorithm allows one enqueue and dequeue to run concurrently. This algorithm should be used for heavily-utilized queues on multiprocessors that has non-universal atomic primitives such as test-and-set.

A queue which is accessed by the single or two processors, single lock would work better than two lock-algorithm.

**Things to improve-**