Paper Review Writeup: Lock-Free Queues

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: 2013

Summary

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.
- 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.

Difference with related work

- Authors of the paper used the batching in B-Queue to reduce the number of shared memory accesses and to detect the batch of available slots at time. Two variables head and tail are used to state the current positions of the producer and consumer. Batch slots for both the producer and consumer are probed by batch_head and batch_tail variables.
- Additionally, batching provides cache thrashing avoidance while facilitating hardware prefetching that improves performance of CLF queues. B-queues are easily used in real applications while they are insensitive to batch-size when considering performance.
- However, if compared with MCRingBuffer presented, used local variable which is accessed to avoid cache thrashing. Though batching improved the performance but is prone to deadlock.
- Deadlock prevention mechanisms presented in related work failed and led to the performance degradation. In one approach, they used timer and auxiliary monitoring threads to periodically check for deadlock and turned SPSC queue into MPSC queue. They also used software and hardware timers that could not solve deadlock problems.
- B-queue used deadlock prevention mechanism, a backtracking to decrease batching size where no timer and monitoring thread is required thus reducing system complexity. Backtracking prevents deadlock by removing condition of circular wait and is efficient as no garbage data is generated. As batching increases the latency, self-adaptive backtracking mechanism presented adjusts the batching size and provides trade-off between latency and performance as B- Queue suffers latency when producer is not busy.

Pros

- Batching in B- Queue reduces the number of shared memory accesses.

- Cache thrashing problem avoided using batching while facilitating hardware prefetching that improved performance of CLF queues.
- Can be easily used in real applications.
- A deadlock prevention 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.

Cons

- 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.
- The efficiency of the B-queue is not tested on the real applications to compare the results with those from real testbeds.

Improvisation for Author

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)

Future Work

B-Queue could be implemented as multiple-producer-multiple-consumer queue in faster core to core communication. Applications of B-Queue are needed to be tested on real applications to facilitate its use in multi-core architecture.

Reviewer Name: Nilesh Balu Jorwar (A20405042)

Paper Title: Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms

Paper Authors: Maged M. Michael, Michael L. Scott

Publication Venue: Proceeding PODC '96 Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing

Year of Publication: 1996

Summary

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.

Authors 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.

The performance of these algorithms is tested onto the 12 processor Silicon Graphics Challenge multi-processor and compared with Single Lock algorithm presented in related work. Results of the evaluation has shown the effective use of two lock algorithm in multi-processor system not multi-programmed ones.

Contributions

- Introduced non-blocking data structure that implemented the queue as singly-linked list with head and tail pointers where head points to dummy node (first node in list) and tail points to last or second to last node in list. The algorithm used compare-and-swap and modification pointers to avoid ABA problem.

- Reusability of dequeued nodes is achieved through dequeuing process. Treiber’s non-blocking stack algorithm is used to implement non-blocking free list.
- Authors also presented two-lock queue data structure that has separate head and tail locks that employ concurrency between enqueues and dequeues. In non-blocking queue, we have dummy node at beginning of the list because of which enqueue never have to access head and dequeues never have to access tail, thus avoiding potential deadlock.

Difference with related work

- 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.
- Similar to Valoi’s algorithm, non-blocking queue is implemented as singly-linked-list with Head and Tail pointers where head points to dummy node and tail points to last or second to last node in list. This algorithm used compare-and-swap with modifications counter to avoid ABA problem. Consistent values of pointers are obtained using sequence of reads which are similar to Prakash et al. algorithm. Treiber’s non-blocking stack algorithm is used to implement non-blocking free list.

Pros

- Linearizability is achieved at algorithm assures single set of operations on single object.
- ABA problem is handled in non-blocking queue.
- 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.

Cons

- For a queue accessed by the single or two processors, single lock would work better than two lock-algorithm.
- Two-lock algorithm cannot complete with non-blocking alternatives on multi-programmed systems.
- Safely properties of non-blocking concurrent queue are not provided with detailed information and hence not sufficient enough to ensure the linearizability.
- It allows only one enqueue and one dequeue to proceed at the same time.

Improvisation for Author

Authors could propose the implementation of the two-lock algorithm on multi-programmed system. Additionally, safety properties of non-blocking concurrent queue data structure could be better tested in more descriptive manner to ensure the linearizability. The performance of these two data structures could be better evaluated on real applications.

Future Work

A system with high thread contention on non-blocking data structures, CPUs may end up burning a lot of cycles busy waiting. Hence designing a system with minimal thread contention using non-blocking concurrent data structures should put into place as these algorithms are published in 1996. These algorithms could be fine-tuned to work on MPMC problems.