Agile Queue: A Fast and Scalable Concurrent Queue on GPU

Md. Sabbir Hossain Polak
The University of Mississippi
University, Mississippi, USA
mhpolak@go.olemiss.edu

David Troendle
The University of Mississippi
University, Mississippi, USA
david@cs.olemiss.edu

Byunghyun Jang
The University of Mississippi
University, Mississippi, USA
bjang@cs.olemiss.edu

ABSTRACT

This work presents *Agile Queue*, a queue specifically designed to support high concurrency on modern GPUs. At its core is to replace conflicting accesses to shared objects with independent accesses to private data. The proposed Agile queue operates on two different granularity - thread block and warp. While the thread block granularity exploits better parallelism among threads, it requires a synchronization primitive to designate a master thread. The warp granularity, on the other hand, leverages work sharing strategy among threads in a warp without any synchronization, which reduces the inherent branch divergence. Both variants support the wrap-around of the head and tail across the ring buffer. Each request to the ring buffer generates a ticket for strict ordering without fully blocking at queue boundary conditions. While the thread block variant utilizes shared memory to reduce global memory accesses, the warp variant broadcasts the offset to all other lanes in the warp by the leader (first active) thread within the warp. Our experiments demonstrate the superior performance and scalability of the Agile queue over existing solutions. Specifically it outperforms the BWD (Broker Queue Work Distributor), the fastest GPU queue to our knowledge, by more than 2× without compromising FIFO semantics.

KEYWORDS

Concurrent Queue, Data Structure, Parallel Computing

ACM Reference Format:
Md. Sabbir Hossain Polak, David Troendle, and Byunghyun Jang. 2024. Agile Queue: A Fast and Scalable Concurrent Queue on GPU. In *The 53rd International Conference on Parallel Processing Workshops (ICPP Workshops '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3677333.3678269

1 INTRODUCTION

The queue serves as an important data structure for building numerous applications yet designing and implementing scalable concurrent queues for GPUs is a challenging task due to elevated concurrency. Traditionally concurrent queue design strategies often prioritize lock-free approaches for performance but the frequent failure of Compare and Swap (CAS) operations outweighs the benefits of lock-free designs [2, 3]. Blocking approaches [4–6] mitigate such bottlenecks but suffer from poor scalability.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICPP Workshops '24, August 12–15, 2024, Gotland, Sweden
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1802-1/24/08
https://doi.org/10.1145/3677333.3678269

We presents a novel scalable and linearizable concurrent queue named *Agile Queue* that is specifically designed for modern GPU architectures. The key features of the Agile queue and our technical contributions are summarized as follows.

- It utilizes fetch-and-add (FAA) operations along with a per-element ticketing system to achieve linearizability.
- It efficiently manages boundary conditions within the queue without the need for additional atomic counters.
- It efficiently consolidates overlapping requests using a single global update.
- We evaluate and compare the proposed design using real-world benchmarks to assess its performance under practical conditions.

2 AGILE QUEUE

The Agile queue is based on a circular array and implements techniques that allocate elements at specific positions within the queue array without failing CAS operations. As such, we coalesce memory requests with array indices and update them with a non-failing atomic exchange operation. The data structure of the Agile queue consists of 1) *head* and *tail* pointers representing the positions within the circular buffer after a modulo operation and establishing the priority of operations that are atomically incremented in response to dequeue and enqueue requests, 2) *items* that store data, and 3) *tickets* that assign a ticket for each request and align them with the respective buffer index position.

2.1 Tickets for FIFO Semantics

When an operation increments `head` or `tail` via non-failing FAA, it not only updates these pointers but also establishes the priority of the subsequent operation right after the increment. It is vital for dequeue or enqueue operation to wait for the enqueue or dequeue in progress for the same index position to complete in order to maintain the FIFO semantics. Consider a group of 5 concurrent requests listed up by their invocation order (`T1-T5`). Depending on thread scheduler, these requests might execute in a different order (e.g., `T1,T2,T4,T3,T5`), deviating from the original invocation order. In Figure 1, `T3,T5` threads execute later than `T1,T2,T4`. Three requests `(enq(x), deq:x, enq(x))` occur on index 0, expecting x on the dequeue operation. Without any technique to maintain the order, the sequence can become jumbled, leading to dequeuing of a different value z instead of x.

One effective approach to maintain order is by attaching a ticket to requests when they are initiated to establish their order. For each index in the queue, we employ a strategy where we attempt to match the ticket associated with the operation before proceeding; if not matched, we wait until it becomes available. Specifically, for an enqueue operation, we attach a ticket which in later incremented by one for a dequeue operation on the same index position. After

employing ticketing mechanism, the previous example changes its behavior as

- T4 initiates before T3, yet it waits until the ongoing dequeue operation (deq:x) at the same index completes its execution.
- Once T3 completes its operation (publishing the node and issuing a new ticket which is the old ticket + 1), the subsequent enqueue operation (enq(z)) can match the ticket associated with the zero-th index position.

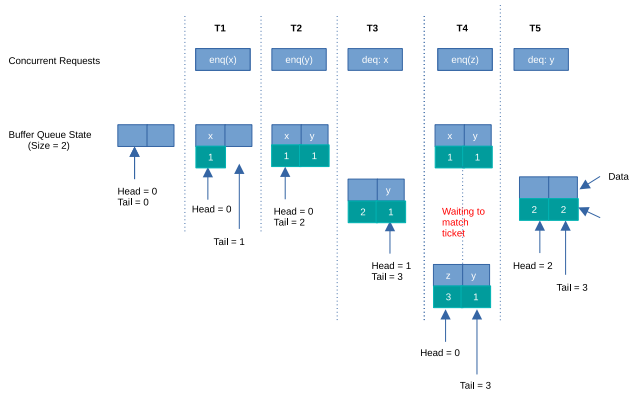

Figure 1: Example of ticketing mechanism.

2.2 Algorithm Variants

The Agile queue implements two variants at different granularity (thread block and warp). The thread block granularity, called *master thread*, utilizes shared memory to minimize slow global memory accesses. However, it exhibits a challenge of thread divergence when enqueue and dequeue requests are mixed across threads. The warp granularity, called *coalescing active threads*, overcomes thread divergence by coalescing only active threads of the diverged path within a warp. The Agile queue achieves high performance by blending two variants. This hybrid approach of two variants leverages the benefits of utilizing shared memory while effectively managing thread divergence within warps.

3 EXPERIMENTAL RESULTS AND ANALYSIS

We measured the performance of the Agile queue in various settings (i.e., master thread, coalescing active threads, hybrid and dynamic approach) on an NVIDIA RTX 4090. We also compare them with the Broker Queue [4], Broker Queue Work Distributor (BWD) [4], Gottlieb Queue [1].

In our investigation of the balanced scenario (Figure 2), we find similar performance between two variants of Agile queue. Notably, the blend of master thread-based enqueue and coalescing active thread-based dequeue demonstrates superior performance over all other competitors. In an imbalanced scenario where workload distributions favor either enqueue or dequeue operations, most queues exhibit roughly 50% lower throughput than the Agile queue hybrid approach. This disparity arises due to boundary conditions encountered under such scenarios. We benchmark for 50:50 enqueue and dequeue ratios (Figure 3). Compared to our queue, Broker Queue,

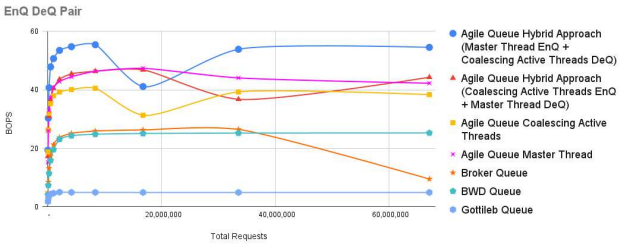

Figure 2: Performance of Enqueue-Dequeue Pair.

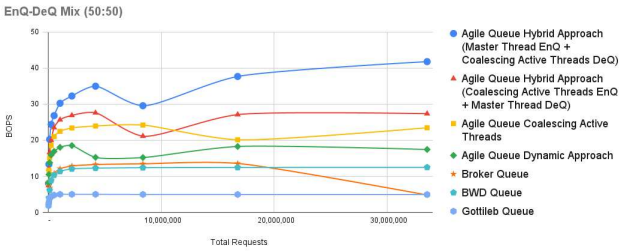

Figure 3: Performance of Enqueue-Dequeue Mix (50:50).

BWD and Gottileb Queue throughput saturate or drop under such high contention as they suffer from atomic operations and thread divergence.

4 CONCLUSION

This extended abstract introduces Agile Queue, a concurrent queue that is designed to achieve high performance and scalability on modern GPUs. Unlike previous approaches which either compromise FIFO semantics or exhibit blocking behavior at queue boundaries, the Agile queue leverages modern GPU features to exploit parallelism. Our experiments demonstrate that the combination of two granularity-based approaches yields the highest throughput compared to existing concurrent queues, in both balanced and imbalanced scenarios. Additionally, we ensure the linearizability of our queue through a ticketing mechanism.

REFERENCES

[1] Allan Gottlieb, Boris D Lubachevsky, and Larry Rudolph. 1983. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5, 2 (1983), 164–189.

[2] Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir. 2010. Flat combining and the synchronization-parallelism tradeoff. In *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*. 355–364.

[3] Maurice Herlihy, Victor Luchangco, and Mark Moir. 2003. Obstruction-free synchronization: Double-ended queues as an example. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.* IEEE, 522–529.

[4] Bernhard Kerbl, Michael Kenzel, Joerg H Mueller, Dieter Schmalstieg, and Markus Steinberger. 2018. The broker queue: A fast, linearizable fifo queue for fine-granular work distribution on the gpu. In *Proceedings of the 2018 International Conference on Supercomputing*. 76–85.

[5] Daniel Orozco, Elkin Garcia, Rishi Khan, Kelly Livingston, and Guang R Gao. 2012. Toward high-throughput algorithms on many-core architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 1–21.

[6] Thomas RW Scogland and Wu-chun Feng. 2015. Design and evaluation of scalable concurrent queues for many-core architectures. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. 63–74.