Graphics Processing Units (GPUs): Architecture & Programming Project

Project Title: Deque Data Structure Library for GPUs

Project Group Number: 12

Project Group Members:

1. Darshan Dinesh Kumar (dd3888, N10942768)

NYU

Problem Definition

- Moore's Law no longer holds true
- Need for innovative solutions to augment performance
- GPUs are ubiquitous, ideal candidates for performance improvements
- However, programming for GPUs is neither straightforward nor intuitive
- The unique programming model can hinder GPU adoption
- Further, even when adopted, it may be severely under-utilized due to suboptimal & inefficient software
- Developing and supporting a data structure library for GPUs can bridge this gap
- This project is such an effort to design and implement a data structure library for GPUs
- It specifically implements a generic deque, a highly versatile & fundamental building block for various applications

Literature Survey

High-Level Abstraction Libraries

- Provides C++ STL like library interfaces for GPU Programming
- Abstracts away the complexities of the underlying programming model

- **NVIDIA Thrust** (bundled with CUDA Toolkit):
  - Supports well-optimized data-parallel algorithms and containers
  - Does not support containers like queue or deque

- **stdgpu:**
  - Recent, C++17 based library interface for GPU and other accelerators
  - Closely related to this project as it supports deques
  - Hence, the stdgpu::deque is used for performance comparison

Low-Level Performance Primitives

- Provides highly-optimized low-level building blocks
- Less for end-users and more for expert developers & for developing higher-level libraries

- **NVIDIA CUB** (part of the CUDA Toolkit):
  - Provides high-performance primitives for intra-block threads
  - Supports *BlockScan*, *BlockReduce* and most relevantly, a *BlockQueue*
  - Mainly focused on Intra-Block communication

Academic Research on Concurrent Data Structures

- Includes research on concurrent & parallel data structures for CPUs and GPUs

- **CPU-based Concurrent Deques:**
  - The **ABP** deque and **Chase-Lev** deque are used in SOTA high-performance work-stealing schedulers
  - A direct port would lead to poor performance due to the difference in CPU and GPU architectures & programming models

- **GPU-specific Concurrent Queues:**
  - Include research on lock free FIFO Queues for GPUs by *Stuart et al. and Tzeng et al.*
  - The focus is on FIFO queue and not deques and they aren't developed as easy-to-use libraries

NYU

Proposed Idea

1. **Lock-Based GPU Deque**
   - An initial, naïve lock-based generic deque for global use across different blocks
   - The single, expensive global spinlock (through atomic CAS) serializes all the operations
   - Hence, naïve in a parallel context but serves as an essential, functionally correct starting point

2. **Lock-Free GPU Deque**
   - Uses atomic operations not as locks but as counters to the buffer slots to support a generic global deque
   - The core of every operation is:
     a) An Atomic Claim to a slot serving as a thread's unique ticket
     b) Checking if the operation is valid (full/empty cases)
     c) A Commit or Rollback based on the check
     d) Mapping of the unbounded atomic counter/ticket to a buffer index in case of a commit

3. **Block-Level Shared Memory Deque**
   - Implements a Lock-Free Deque as above that lives entirely in Shared Memory
   - Prioritizes reduced latency & locality targeting applications requiring only Intra-Block Communication

Experimental Setup

Model	NVIDIA GeForce RTX 4070
Architecture	Ada Lovelace
Compute Capability	8.9
CUDA Version	13.0
Driver Version	580.76.05
Total GPUs	1
Number of SMs	46
Number of SPs per SM	128
Total SPs	5888 (46 * 128)
PCIe Version	Gen3
GPU Clock	1,920 MHz
Max GPU Clock	3,105 MHz
Power Draw	96.6 W
Power Limit	200 W

Configuration of *cuda5* CIMS machine used for Experiments

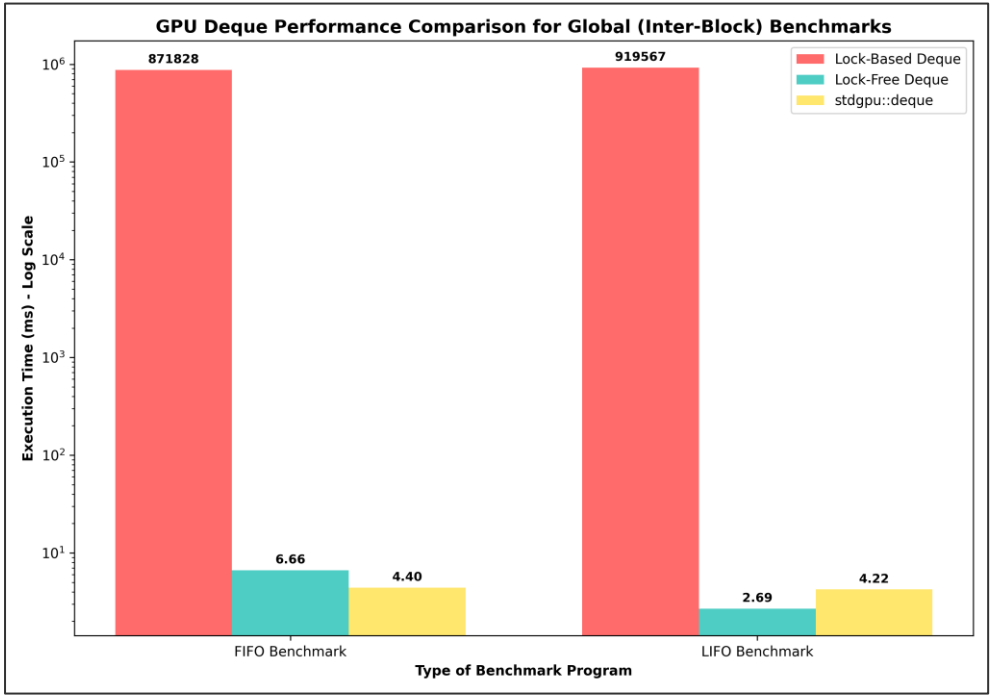
Experiments

1. **Functionality Verification**
   a) Sequence of Deque operations testing functionality
   b) Verifying generic features:
      i. int
      ii. float
      iii. struct representing Complex numbers
      iv. FixedString<N> template for string processing

2. **Performance Benchmarking**
   a) Global (Inter-Block) Benchmarks – FIFO and LIFO
   b) Global (Inter-Block) FIFO & LIFO Benchmarks for stdgpu
   c) Intra-Block Deque Benchmarks – FIFO and LIFO

NYU

Results and Analysis

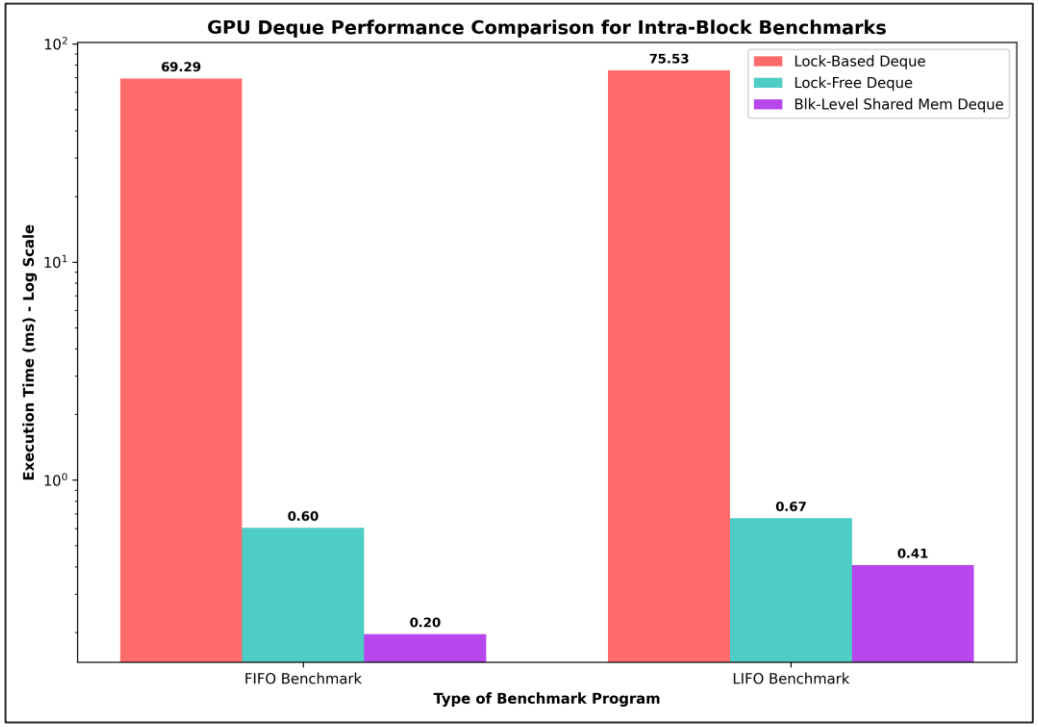


Global (Inter-Block) Benchmarks:

- Metric: Execution Time (in ms) of the benchmarking kernels
- Lock-Based has the worst perf due to global spin lock
- Lock-Free and stdgpu deques show 100000x Speedup due to removal of global locks
- Lock-Free has 50% slowdown against stdgpu for FIFO
- Lock-Free has 36% improvement against stdgpu for LIFO
- Possible Reasons: memory coalescing optimizations, chunked memory layout instead of a single contiguous array, and better cache usage in stdgpu deque

NYU

Results and Analysis



Intra-Block Benchmarks:

- Metric: Execution Time (in ms) of the benchmarking kernels
- Lock-Based has the worst perf due to global spin lock
- Lock-Free and stdgpu deques show 100x Speedup due to removal of global locks
- Block-Level has the best perf with improvements of 66.67% for FIFO and 38.8% for LIFO as compared to Lock-Free
- This is because Lock-Free houses its buffer in global memory whereas Block-Level houses its buffer in faster and nearer shared memory

NYU

Conclusion

1. The project implements 3 versions of a Deque & its operations for GPUs as an easy to integrate, header-only library:
   i. Naïve Global Lock-Based implementation
   ii. Lock-Free version using ticketing for slots with atomic counters
   iii. Block-Level shared memory deque for Intra-Block communication

2. The lock-free version (ii) performance results are quite promising, showing a 36% improvement for LIFO against stdgpu, while the block-level version (iii) shows improvements up to 67% against the global version (ii) for Intra-block benchmarks

3. Admittedly, there is substantial scope for future work:
   - Despite the improvement for LIFO, the lock-free version (ii) has 50% slowdown for FIFO against stdgpu
   - This needs to be investigated further to deduce possible reasons
   - Relevant optimizations like memory coalescing, chunking, improving cache efficiency could be experimented with
   - The current design uses a fixed size buffer and hence does not allow for resizing. It needs to be explored if this can be supported considering the constraints like can't invoke CudaMalloc within a kernel, no global synch to stop threads while resizing.