Lazily Batched Parallel Execution Framework for Concurrent Data Structures in C++

Aranya Banerjee
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, US
aranyabanerjee@gatech.edu

Bharat Bhushan Reddy Thotti
School of Computer Science
Georgia Institute of Technology
Atlanta, US
bharat.reddy@gatech.edu

Mayur Peshve Lakshmana Rao
School of Computer Science
Georgia Institute of Technology
Atlanta, US
mrao70@gatech.edu

I. Introduction

In the rapidly evolving domain of high-performance computing, the demand for efficient and scalable data processing solutions has never been more critical. The need to explore and optimize data structures for parallel execution has become increasingly evident. This project aims to develop a framework in C++ that enhances the performance of operations on standard concurrent data structures through lazily batched parallel execution. Operations such as insertions/updates are batched and executed in parallel at trigger points (e.g., a search or reaching a predefined batch size limit) which need accessing the up-to-date state of the data structure. This approach should allow for significant performance improvements by leveraging the parallel processing capabilities of modern multi-core processors, all while maintaining the simplicity and familiarity of standard data structure interfaces for the user. By batching, the idea is to reduce contention and allow for threads to perform useful tasks rather than waiting on a lock.

II. Problem Description

A. Objectives

Our goal is to design and implement a wrapper/framework that enables batched parallel processing of operations on concurrent data structures in C++, without requiring users to modify their code. These data structures must be concurrent, to begin with, to allow parallel operations to be performed on them. Concurrent data structures allow multiple threads to access and modify shared data simultaneously, while batch-parallel structures process data in fixed-sized batches, often enhancing parallelism by reducing contention.

B. Related Work

There has been substantial research into the algorithms to perform efficient batching given the implementation of a batch parallel data structure [1] but not into the conversion of concurrent data structures into batch parallel data structures. The ones that delve into this conversion either require the batch parallel data structure implementation and provide a wrapper for the algorithm to perform batching or are not in C++ [2]. There has not been much work into the efficient conversion between the two types of data structures in C++

C. Impact

By automating the batching and parallel execution of operations, we aim to significantly reduce the contention and context switch time, thereby improving the performance, efficiency, and predictability of C++ programs, especially those dealing with large data sets or intensive computation tasks consisting of huge number of batch-able operations.

III. Approach

The core of our project involves developing a C++ framework that intelligently batches operations on concurrent data structures and executes them in parallel when a triggering condition is met. This approach requires several key components and strategies

Operation Interception and Batching

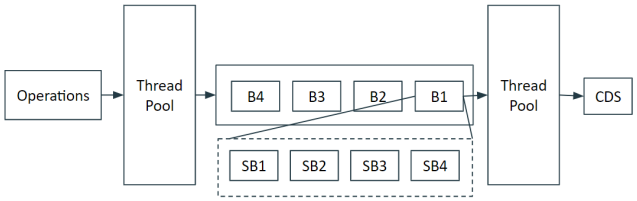
1) **Mechanism:** Implement template wrappers for standard data structures (‘std::set‘, ‘std::unordered_map‘, etc.) that intercept write operations (e.g., ‘insert‘, ‘erase‘). These operations are not immediately executed but are instead stored in an operation queue specific to each data structure instance.
2) **Batch Formation:** The framework monitors the operation queue, forming batches based on specific criteria, such as reaching a certain size (user-defined parameter) or a read operation (‘find‘, ‘search‘) being called.

Thread-Safe Queue Design:

1) **Mechanism:** Implement efficient enqueue and dequeue operations to manage the incoming operations using locks for concurrency and synchronization and to ensure thread safety
2) **Optimizations:** Consider using a circular buffer or linked list to optimize the queue’s performance.

Batching Engine:

1) **Thread Pool Management:** Utilize a fixed-size thread pool to manage worker threads efficiently. The size of the pool could be determined based on the hardware capabilities (e.g., number of available CPU cores).
2) **Subbatch Division:** When a batch is ready for execution divide batches into sub-batches to distribute the workload efficiently among threads in the thread pool.



The Framework

Triggering Mechanisms:

1) **Automatic Triggers:** Develop logic for automatically triggering batch processing. This includes monitoring for read operations that require the most recent state of the data structure or reaching predefined thresholds in batch size.
2) **Manual Triggers:** Provide users the option to manually trigger batch processing, offering flexibility for scenarios where the optimal timing for parallel execution may be application-specific. This will also help us in studying how performance changes with batches smaller than those generated from our automatic trigger policy.

Compatibility and Integration

1) **Seamless Integration:** Design the framework to be easily integrable into existing C++ projects with minimal changes, preserving the familiar interface of the original data structures.
2) **Extensibility:** Ensure the framework is extensible, allowing for the addition of support for more data structure types and operations over time.

By adopting this approach, we aim to combine the ease of use of traditional concurrent data structures with the performance benefits of batch parallel processing, making high-throughput and efficient data manipulation accessible to C++ developers without requiring them to be experts in parallel programming.

A. Environment

We propose an efficient solution framework in C++ that uses template programming for generalizing over multiple data structures. The initial goal is to implement this framework for a multithreaded setting for a single data structure. The stretch goal is to implement an optimized distributed generic framework using template programming in C++. We will mostly be limiting ourselves to the standard threads library in C++ and maybe use the promise and future interfaces to achieve the parallelization

IV. Evaluation Plan

The research will involve the implementation and evaluation of both concurrent and batch-parallel data structures within a parallel computing framework. Performance metrics, such as execution time, scalability, and resource utilization, will be carefully analyzed to draw insightful conclusions regarding the efficacy of each approach.

Benchmark Design:

1) Develop 2 sample programs one using standard c+ data structures and another leveraging our framework
2) **Operation Mixes:** Test various ratios of batch-able to trigger operations to explore performance under different usage patterns.
3) **Volume Scaling:** Test with a range of operational volumes from small to large to assess how the framework scales with the number of operations in the program.

Metrics:

1) **Execution Time:** Primary metric would be the execution time which would focus on the complete duration of the program execution.
2) **Space consumption:** Analyze the memory footprint of both the standard and the framework-based data structures, comparing their resource efficiency
3) **Speedup (Strong Scaling):** Analyse how the framework performs as we increase the number of threads but keep the number of operations constant
4) **Efficiency (Weak Scaling):** Analyse how the framework performs as we increase the number of operations along with the number of threads
5) **Thread Usage:** Profile thread usage data to verify if work load is balanced and if contention (waiting) has reduced

Analysis

1) **Performance Gains:** Quantify the speed enhancements achieved through batch-parallel execution compared to traditional sequential methods.
2) **Scalability Insights:** Determine how the framework performed under varying operational mixes and volumes, pinpointing scenarios where it performs the best and the worst.
3) **Assessing Overheads:** Evaluate any trade-offs associated with the framework, such as additional memory usage or initialization complexities.

V. Timeline

Dates	Goals
Mar. 18 - Mar. 22	Generic thread-safe queue of operations and batching algorithm
Mar. 25 - Mar. 29	Generic thread pool, apply the framework on a data structure
Apr. 1 - Apr 5	Testing and validation
Apr. 8 - Apr 12	Apply the framework on more data structures
Apr. 15 - Apr 19	Acquire data points and metrics, prepare for final presentation

Mid-Term Target

Demonstrate the framework's basic functionality in intercepting operations, batching them and parallelly executing them on a simple data structure like 'std::set' with basic performance metrics.

Final-Exam Target

Conduct extensive performance evaluations across multiple data structures and operational mixes, presenting comprehensive comparisons of execution times and space consumption.

REFERENCES

[1] Agrawal, K., Fineman, J. T., Lu, K., Sheridan, B., Sukha, J., & Utterback, R. (2014). Providing good scheduling for parallel programs that use data structures through implicit batching. https://doi.org/10.1145/2612669.2612688

[2] Wen, L. K. (2023). Concurrent structures and effect handlers: a batch made in heaven (By Yale NUS College) [Thesis]. https://ilyasergey.net/assets/pdf/papers/Koon-Wen-Lee-Capstone.pdf

[3] Huang, T.-W., Lin, C.-X., Guo, G., Wong, M., & Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, IL, USA. (2019). Cpp-Taskflow: Fast Task-based Parallel Programming using Modern C++. In 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). https://taskflow.github.io/papers/ipdps19.pdf

[4] https://github.com/khizmax/libcds