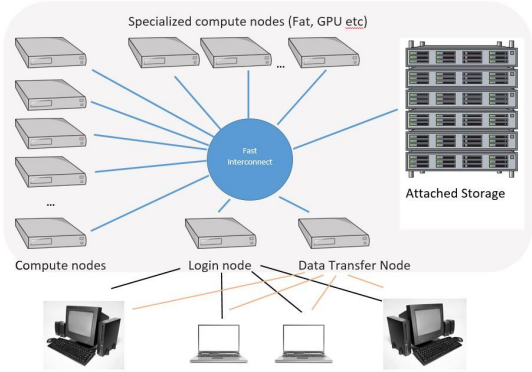
Lazily Batched Parallel Execution Framework for Concurrent Data Structures in C++

Aranya, Bharat, Mayur

Why

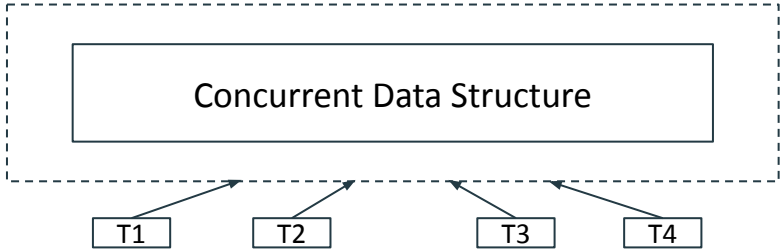
- Parallel and Distributed computing environment all around us
- Critical demand for efficient and scalable data processing solutions
- No previous framework or research in C++
- No data points to reason if batch parallel is good or bad



https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.hpc.iastate.edu%2Fguides%2Fintroduction-to-hpc-clusters%2Fwhat-is-an-hpc-cluster&psig=AOvVaw05JrT7eXhduSsWsqW8fOC1&ust=1710357470272000&source=images&cd=vfe&opi=89978449&ved=0CBMQjRxqFwoTCLDN1Ju474QDFQAAAAAdAAAAABAE

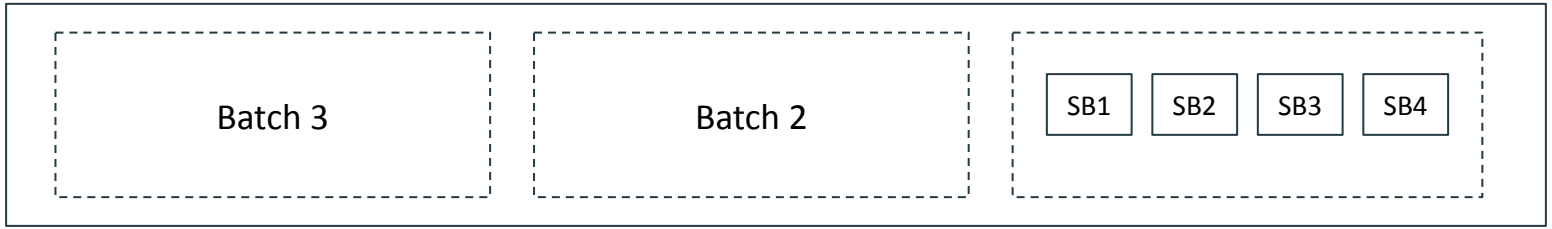
Concurrent Data Structures

- Each operation requires acquisition of lock
- High contention
- Unpredictable performance
- Performance gains diminish with higher thread and operations count



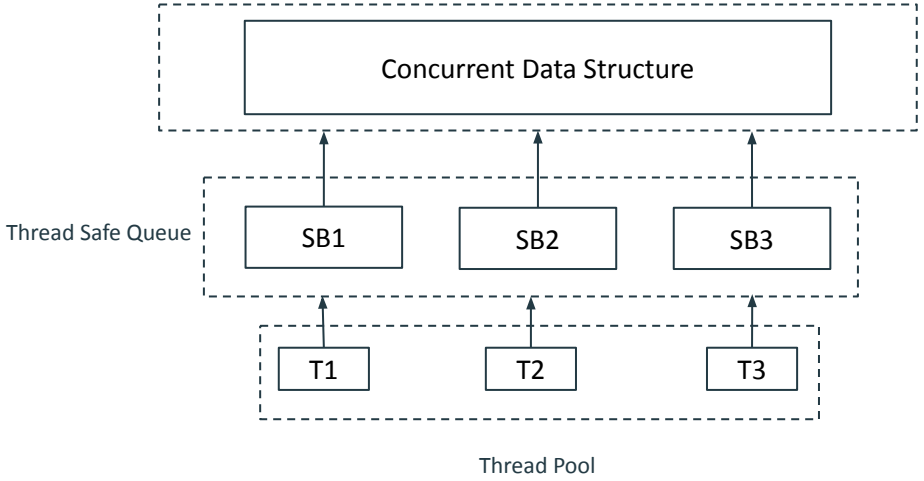
Thread Safe Queues

- Queue of batches
- Batch is a collection of operations
- Each batch divided into subbatches
- Batches are applied **in order** to ensure correctness
- Sub batches are applied **in any order**



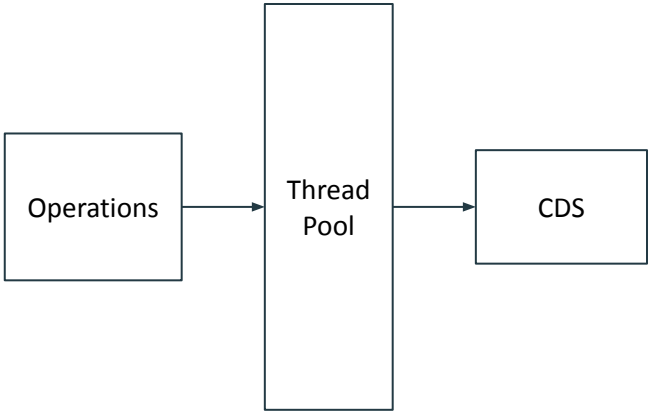
Batch Parallel Data Structures

- Each thread is assigned a sub batch
- Each sub batch is applied to CDS
- Reduced contention
- Predictable performance
- Performance gains at lower thread and operations count may be suboptimal
- Better Performance gains at higher thread and operations count



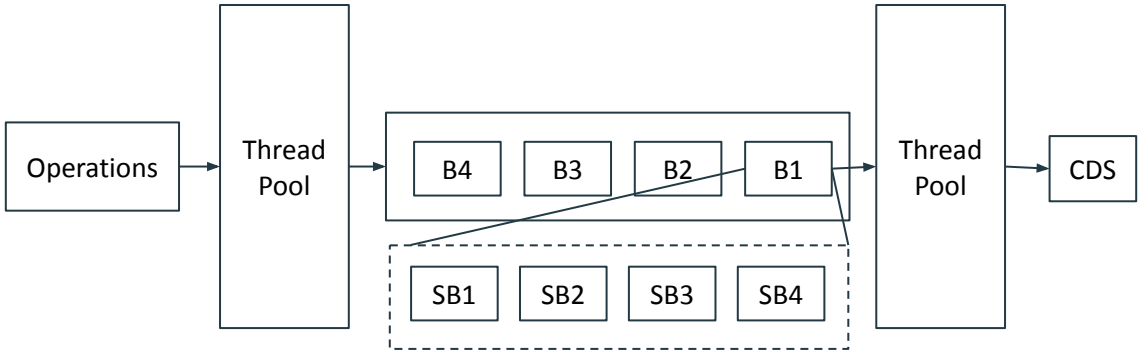
Putting it all together - Before

- Operations are applied concurrently on the data structure
- Each individual operation requires lock acquisition
- High contention at higher thread and operations count
- Reduced predictability



Putting it all together - After

- Operations are batched before being applied
- Subbatches allow for parallel execution of batches
- Dependencies are handled implicitly by the batching logic
- Batching reduces contention



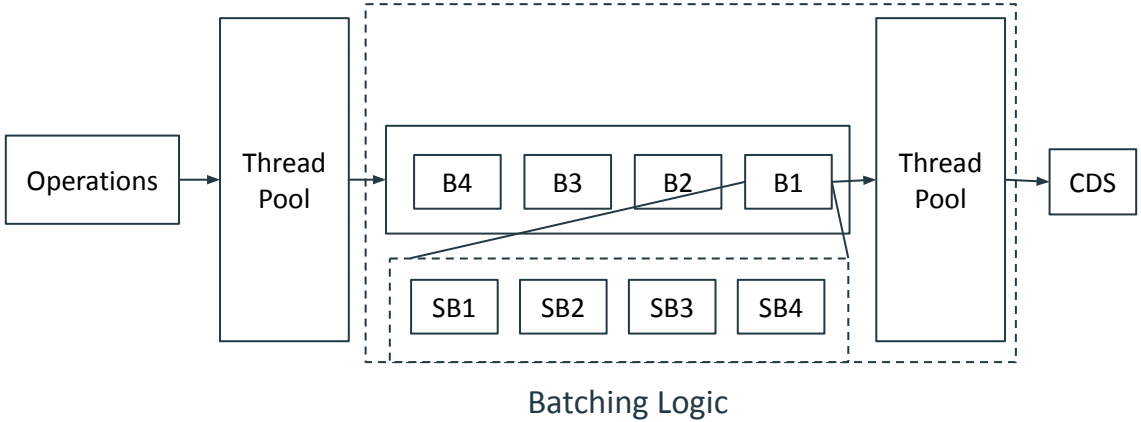
Related Work

- No C++ wrapper/framework related to this work
- Efficient batching algorithms but batch parallel data structures are user pain [1]
- Batch parallel conversion of a specific data structure (skiplist) in OCAML [2]

[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.pd

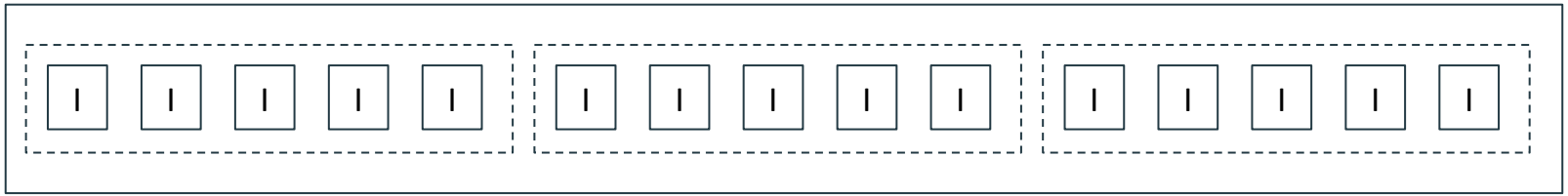
Approach - Batching Logic

- Implement wrappers to intercept operations on CDS
- Form batches when certain conditions are met
- Optimise thread safe queue to reduce insertion time



Implicit Batching

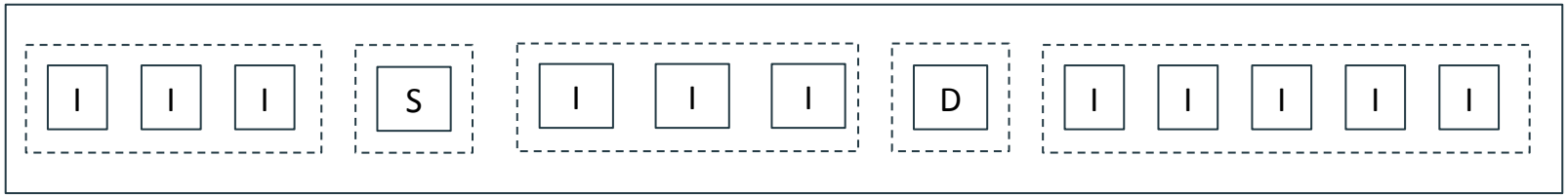
- Occurs when each operation in a set of operations is independent
- Batch size is a user parameter
- Each operation within a batch can be applied in any order (low Fairness)



Queue with a batch size 5

Explicit Batching

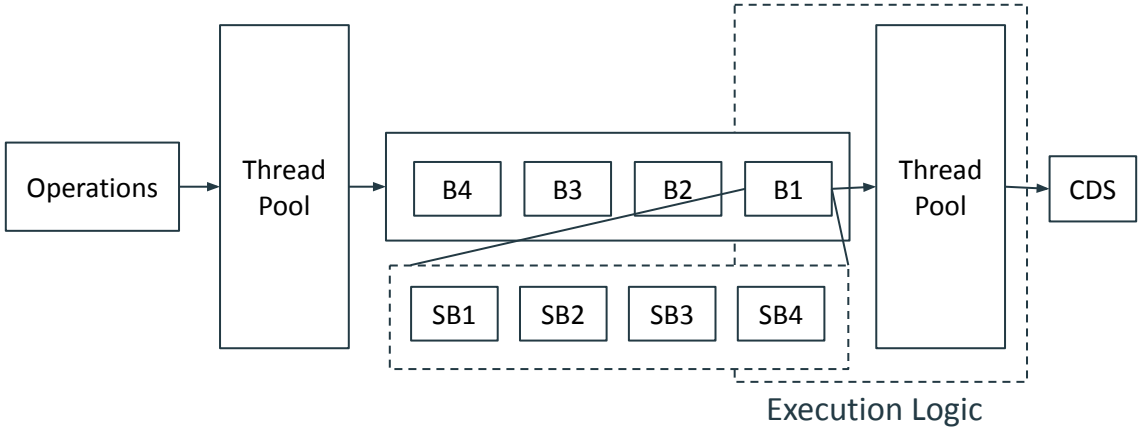
- Occurs when an operation needs to be executed before other future operations
- Example: Delete, Search (based on fairness)
- Handles dependencies implicitly
- Necessary to maintain correctness



Queue with a batch size 5

Approach - Execution Logic

- Maintain a thread pool to execute operations
- De-queue a batch from operations queue and form optimised subbatches
- Execute the subbatches on the CDS using the threadpool



Environment

- Efficient C++ framework using Template Programming
- C++ standard threads library
- C++ standard synchronisation primitives
- C++ promise and future

Evaluation Plan

- Compare execution time and space requirements of sequential program vs Batch parallel version
- **Operational Mixes**, eg: 90% inserts + 10% searches, etc
- **Volume scaling**: Evaluate how time and space varies with increasing operations

Parameters

- Number of Operations
- Operations per Batch
- Thread pool size
- Fairness (Total ordering)

Metrics

- Runtime
- Space Complexity
- Speedup (Strong Scaling)
- Efficiency (Weak Scaling)
- Thread Usage

Goal

Mid Term Target:

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

Final-Exam Target

- Conduct extensive performance evaluations
- (Stretch Goal) Generalise over multiple data structures and operational mixes, presenting comprehensive comparisons of execution times and space consumption.

Timeline

Dates	Goals
March 18 - 22	*Generic* thread safe queue of operations and batching algorithm
March 25 -29 (Mid Term Goal)	*Generic* thread pool, apply the framework on a data structure
April 1 - 5	Testing and validation
April 8 - 12	Apply the framework on more data structures
April 15 - 19 (Final Goal)	Acquire data points and metrics, prepare for final presentation

Generic : Stretch Goal

Thank You