Avi Kapadia, Aadit Trivedi
March 10, 2025
CS 6423

MVCC Optimization Through GPU Utilization

Problem

Multiversion Concurrency Control (MVCC) is a technique that allows databases to maintain multiple versions of each data item to allow multiple transactions to access and modify data concurrently, using versioning instead of locking. However, this creates increased storage overhead and a need for periodic garbage collection, which can slow down database performance for large-scale or highly concurrent workloads.

Objective

We plan to use the parallel computing capabilities of GPUs to accelerate MVCC-related operations, such as garbage collection and transaction visibility checks. We aim to explore extensions to MVCC using GPUs, which can provide insights into how GPU architectures may benefit concurrency control in databases.

Proposed Solution

In our GPU-based solution:

- We will implement CUDA kernels to scan MVCC metadata in parallel,
- Design and optimize a GPU-driven garbage collector
- Implement GPU kernels to accelerate transaction visibility and ensure rapid snapshot construction, and
- Potentially explore alternative protocols (ex: OCC, 2PL).

Our setup will likely be (or at least start off as) the following for simplicity:

- transactions = [t1, t2, t3, …] # pre-defined set of transactions
- version_table = { } # versioning data structure
- db # We have not yet decided how we are going to represent our database but it will likely be a simple implementation such as a table or a csv so we can focus on GPU optimization for concurrency control.

Evaluation

We will utilize performance benchmarks such as garbage collection and visibility-check execution times compared to CPU implementations across varying workloads. We can also observe database throughput improvements and transaction latency reductions that may result from GPU acceleration.

Goal Scheme

- 75% - [
  Our 75% goal involves core functionality implementation. This includes both multithreaded version checking along with parallel garbage collection. At this point, we should have implemented our GPU kernel(s) for MVCC metadata scanning. We should be able to observe some initial correctness over CPU implementations.
  ]
- 100% - 75% + [
  We will build on our 75% goal by improving memory access patterns with GPU. We will also implement multi-threaded conflict detection. The goal is here to fully optimize and integrate a GPU-driven garbage collection with our simulated MVCC database environment, alongside implementing GPU-based transaction visibility checking and snapshot generation.
  ]
- 125% - 100% + [
  We plan on extending our GPU optimization beyond just multi-version concurrency control into other concurrency protocols if time permits. There are particularly two other concurrency protocols that we are interested in exploring GPU's impact in: Optimistic Concurrency Protocol (OCC) and 2PL (Two Phase Locking)
  + potential Firebase DB integration
  ]

Resources

For successful implementation and testing of this project, we will need access to GPUs. We will use Georgia Tech's PACE ICE cluster for this purpose. Our code will be running CUDA C++ on these GPUs. We also plan to use the CPU from our own laptop to do a comparison between our GPU-optimized concurrency protocol implementation and a naive implementation of concurrency protocol.

References

- https://www.usenix.org/conference/osdi24/presentation/qian
- https://arxiv.org/pdf/2406.10158
- https://arxiv.org/pdf/2007.04292
- Concurrency control slides from lecture
- We will continue to add to this list as we do more reading…