Parallel Topological Sorting: Project Proposal

Ezra Fu (erzhengf), Robert Benjamin Osborne (rbo)
https://codeplay0314.com/parallel-topological-sorting/

Summary

We plan to explore and implement parallel solutions for computing the topological ordering of Directed Acyclic Graphs (DAGs), using both CUDA for GPU acceleration and a Domain-Specific Language (DSL) optimized for expressing dependencies.

Background

Topological ordering of DAGs is essential in various domains such as scheduling, deep learning computation order, and dependency resolution in rendering. Traditional topological sort algorithms, like Kahn's algorithm or depth-first search (DFS), are inherently sequential and can become bottlenecks in systems requiring high performance.
Our project aims to accelerate the topological sorting process by exploiting parallelism through:

- **Serial Implementation** (Baseline): We will implement a standard sequential topological sort algorithm to serve as a baseline for measuring performance improvements.
- **CUDA for Matrix Manipulation**: By representing the DAG as an adjacency matrix, we can leverage CUDA's parallel processing capabilities to perform simultaneous operations on multiple nodes and edges.
- **Domain-Specific Language** (DSL): Utilizing or developing a DSL tailored for expressing dependencies, we aim to optimize parallel processing by abstracting the complexity of parallelism and allowing for more efficient execution of topological sorting algorithms.

The parallelism in this problem arises from the ability to process independent nodes concurrently, which can significantly reduce computation time compared to the serial approach.

Challenge

Workload Description

Dependencies: Handling inherent dependencies in DAGs is challenging. Nodes can only be processed after all their predecessor nodes have been processed, requiring careful synchronization in a parallel environment.
Memory Access Characteristics: Graph algorithms often involve irregular memory access patterns, which can hinder performance on GPUs due to cache misses and uncoalesced memory accesses.

Divergent Execution: Threads may follow different execution paths based on the graph structure, leading to warp divergence on GPUs and reducing efficiency.

Constraints

Synchronization Overhead: Managing dependencies introduces synchronization mechanisms that can limit parallel scalability.
Irregular Data Structures: Efficiently mapping graph data structures to parallel hardware is challenging due to their irregularity.

By comparing the serial and parallel implementations, we hope to learn how to effectively parallelize graph algorithms and understand the trade-offs involved.

Resources

Hardware: CPUs and NVIDIA GPUs (GHC and PSC machines)
Software: C++, CUDA, and a suitable DSL framework

We are starting from scratch for this classic problem with code.

Goals and Deliverables

Implementation

- Optimize memory access patterns to reduce latency and increase throughput
    - Develop a serial topological sort algorithm as a performance baseline
    - Implement a CUDA-based parallel topological sort using adjacency matrix manipulation
    - Implement a DSL-based solution

Performance Analysis

- Benchmark both parallel implementations against the serial baseline
- Analyze speedup factors, scalability, resource utilization, and overheads

Demo for Poster Session

- Performance Graphs: Show speedup graphs and resource utilization charts to highlight efficiency gains.
- Code Walkthrough: Briefly explain key sections of the code to demonstrate optimization strategies.
- Interactive Visualization: Display real-time comparisons of the serial and parallel algorithms processing sample DAGs.

Schedule

Week 1 (Nov 11 - 17)

- Review literature on topological sorting
- Set up development environments for C++, CUDA, and DSL tools
- Implement the standard sequential topological sort
- Validate correctness with sample DAGs and establish baseline performance metrics

Week 2 (Nov 18 - 24)

- Begin coding the CUDA-based topological sort
- Optimize kernel functions for better memory access patterns

Week 3 (Nov 25 - Dec 1)

- Select a DSL
- Implement the topological sort algorithm using the DSL

Week 4 (Dec 2 - 8)

- Test and debug issues related to correctness, concurrency, data races, and synchronization
- Collect data on execution time, speedup, scalability, and resource usage
- Analyze performance and efficiency
- Test implementations on real-world DAGs from applications like deep learning and rendering

Week 5 (Dec 9 - 15)

- Compile findings into the final report
- Reassess goals and ensure all deliverables are met
- Rehearse the demo and prepare for the poster session