15418 milestone report

Jason Yuan (laiy), Rui Zhou (ruizhou)

April 14, 2026

1 Updated Schedule

We also updated our project page https://jyuan2003.github.io/task-scheduler.

- Week 1: Generate graphs with varying topologies, edge densities, and sizes and re-organize the graph structure to improve locality and storage through graph sharding and compression.
- Week 2: Implement graph and scheduler utilities and an MPI pipeline that supports all graph representations. Implement all-to-all communication.
- Week 3:
  - Week 3.1:
    1. Simulate real runtime communication cost. (Jason)
    2. Implement a ring interconnect topology. (Jason)
    3. Build an OpenMP pipeline that supports all graph representations. (Rui)
  - Week 3.2:
    1. Implement experiment pipelines and necessary utilities to record results and plot the communication/computation cost, and cache behaviors by types and representations of graphs. (Jason)
    2. Implement coarse and fine-grained lock/lock-free data structures in OpenMP. (Rui)
- Week 4:
  - Week 4.1 Debug and continue to optimize current OpenMP and MPI implementations. Benchmark their scalability on PSC machines. (Both)
  - Week 4.2 Summarize and analyze the final results. (Both)

2 Progress So Far

We have implemented a graph generation pipeline using Python and Networkx to generate weighted directed graphs, with weights on both nodes and edges, various input sizes, and topologies. We have also represented graphs and relevant graph utilities (i.e. read/write, get stored node/edge information, and some iterators) using raw (no further processing beyond the initial generation), sharded, compressed, and sharded+compressed, following the lecture slides. For the message passing model, we have set up a pipeline that supports all graph representations and implemented all relevant utilities for I/O, scheduler representation, statistics reporting, and all-to-all communication.

3 Goals and Deliverables

We believe that we are able to achieve goals described in the project proposal. More concrete goals and deliverables are listed below.

Plan to Achieve:

- Have a correct graph generation pipeline that supports various node sizes, edge density, node and edge cost, and random/clustered topologies.
- Have a correct implementation of a task scheduler that supports all four graph representations (raw, sharded, compressed, and sharded + compressed).
- On clustered graphs, our baseline models (with minimal optimizations) generally show improved cache behavior and reduced memory usage compared to the raw representation under our graph representations.
- On clustered graphs, our communication/synchronization optimizations generally demonstrate good scalability and cache behaviors.
- We will also evaluate and analyze the performance of our task scheduler on random graphs.

Hope to Achieve:

- Our communication/synchronization optimizations also demonstrate good scalability and cache behaviors on random graphs.

4 Poster

We plan to show figures of communication/synchronization and computation cost by types and representations of graph. We will also show the figures of cache behaviors (cache misses, etc).

5 Preliminary Results

We explored the relationship between graph representations and types of graphs. We found that

1. In general, the sharded representation has around 2% higher memory overhead than the raw representation.
2. For both clustered and random graph topologies, compression reduces memory usage most of the time. It helps more for clustered graph, which achieves around 10% greater savings compared to random graphs. However, when the graph is random and sparse, compression can lead to higher memory usage.

6 Concerns

At this point, several challenges still remain.

1. Although our current utilities expose a uniform interface (e.g. there is a shared interface for out-neighbor retrieval), these implementations differ across graph representations. For the sharded representation, this is an $O(1)$ operation, whereas for raw and compressed representations, this is much more involved. Our current implementations naively iterate over all edges. We believe that the difference in utility cost may confound the analysis of our program's performance on the graph itself.

2. We formulate the edge cost between two nodes as the communication cost between different processes (colors), but our code currently just treats them as numerical values instead of letting the program actually hangs. We will need to carefully design this latency, as if it's too large, we won't have enough time to run all the experiments, or if it's too short, the communication difference between different optimizations may not be meaningfully reflected.

3. We still need to fix the issues of the current utilities and potentially to expand them to support later implementations.