SHREC SURG OpenMP

SUMMER UNDERGRADUATE RESEARCH GROUP

Parallelized Eager Prim Algorithm

Shixiong Jing, Evan Gretok, Alan George

Mission-Critical Computing
NSF CENTER FOR SPACE, HIGH-PERFORMANCE, AND RESILIENT COMPUTING (SHREC)

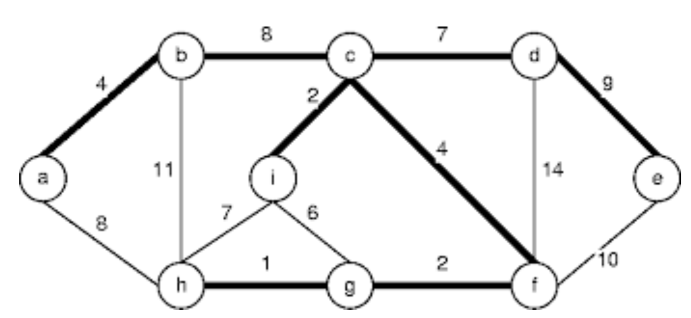
PITT SWANSON ENGINEERING
ELECTRICAL & COMPUTER

Introduction

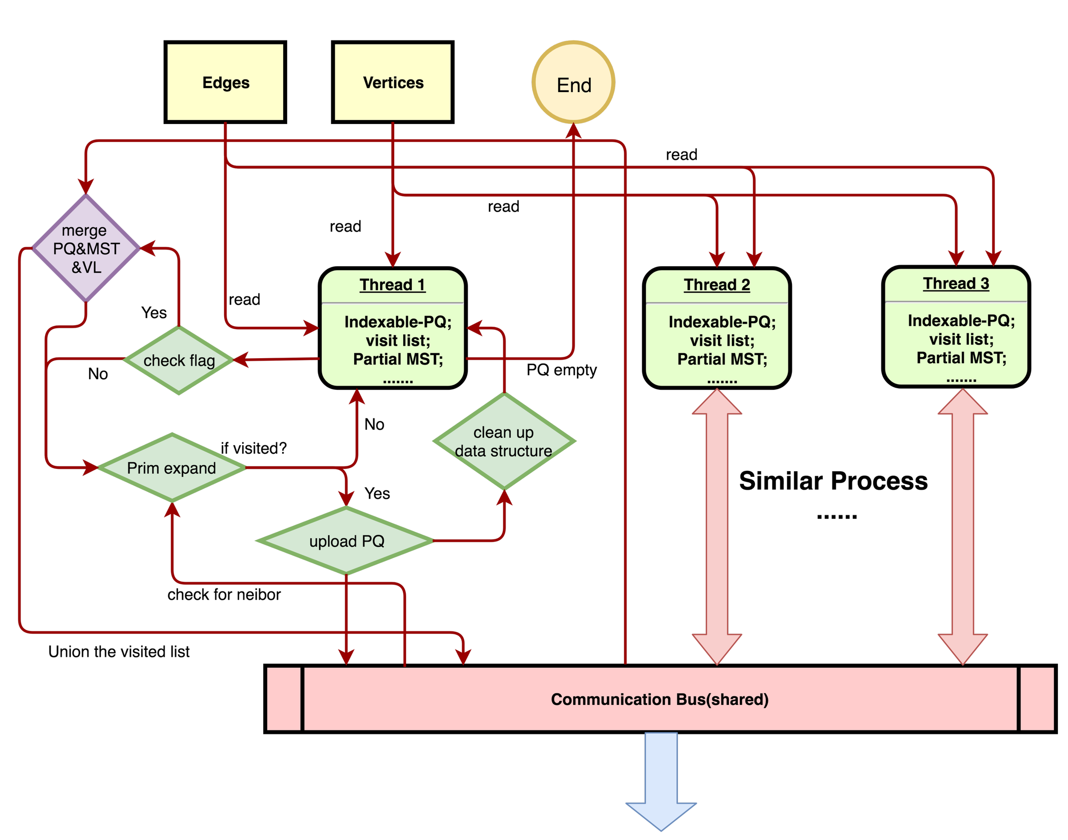
- **Goals**
  - Parallelize and optimize the eager Prim's algorithm to run on multi-core machines
  - Design test cases to analyze the performance of the parallelized algorithm
- **Motivations**
  - Eager Prim serves as an important method to find the minimal spanning tree of a graph.
  - Prim's algorithms can be used in printed-circuit-board design to optimize connections between components; It is also used in computational military reasoning.
- **Challenges**
  - Spanning tree formation is usually a serial process, there are not many parallelized implementations for these problems.
  - Indexable priority queue used by eager Prim causes difficulty in parallelization and data sharing between threads.
  - Complex data sharing mechanism require careful design to ensure data integrity.

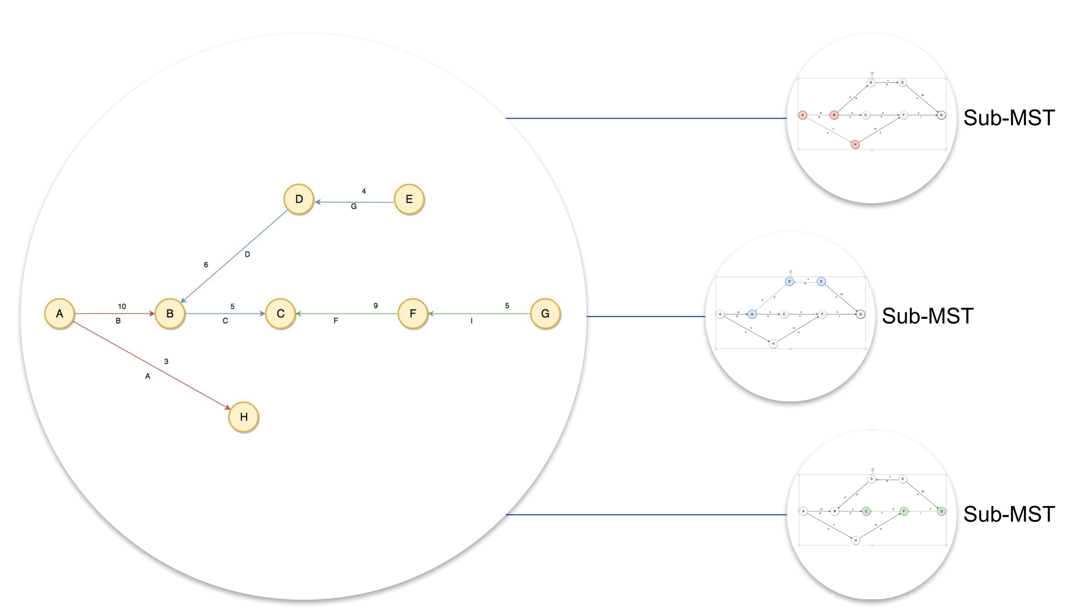
Approach

- **Characteristics of Eager Prim**
  - Unlike the original Prim's algorithms, eager Prim can achieve O(e log(v)) runtime instead of O(n^2) by taking advantage of an indexable priority queue structure.
  - Eager Prim always generates the spanning tree with minimum weight regardless of the start point.
- **Project Methodology**
  - Start from different nodes at the same time
  - Use shared memory to communicate between threads
  - Merge threads when two threads meet



- **Design**
  - C++ with OpenMP
- **Parallelization Layout**
  - Shared: PQ Table for data transfer, overall visited list, array of progress record, upload flags, etc.
  - Private: private visited list, partial minimum spanning tree, Priority Queue, etc.
- **Other Optimization**
  - Threads stop working when 60% vertices has been visited
  - Use vector in C++ instead of array to increase speed
  - Using hash-related method to randomize start point selection

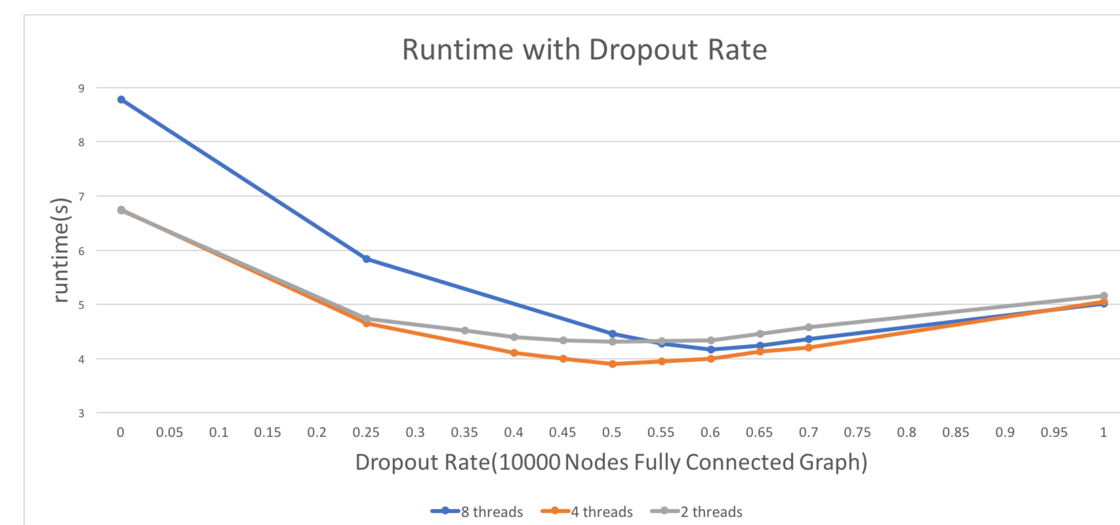


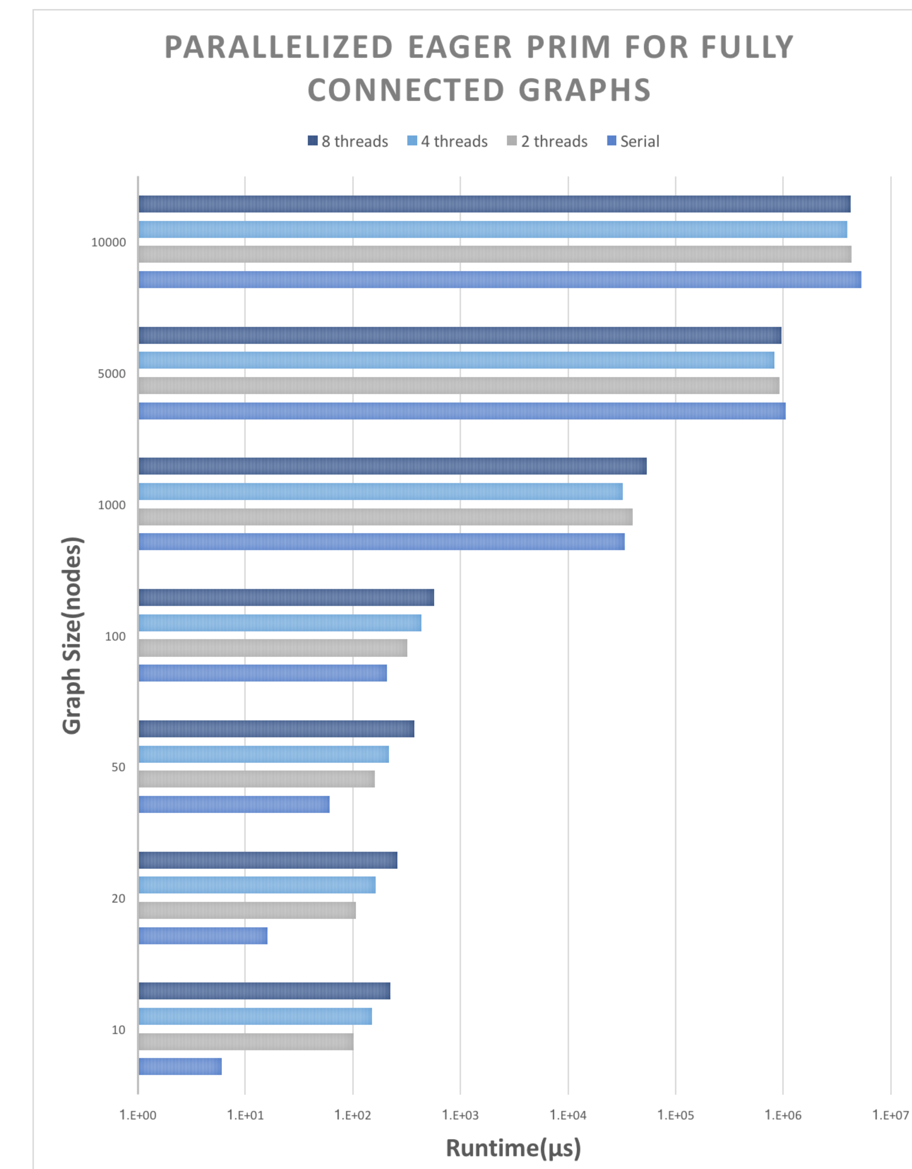


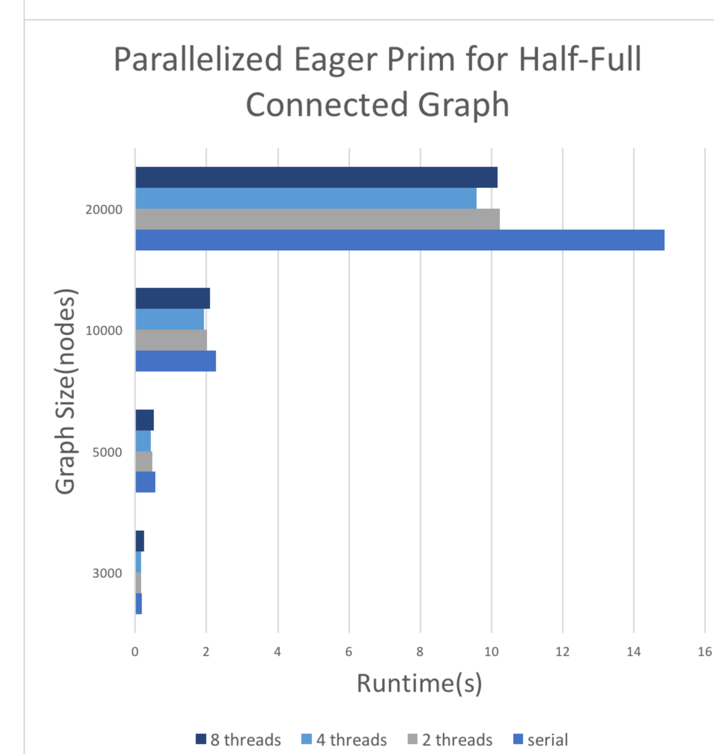
Results

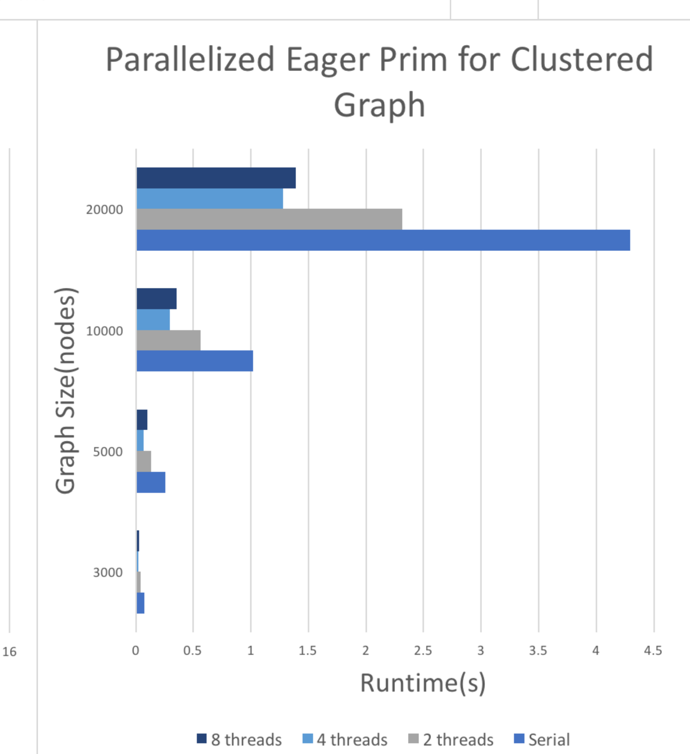
Fully-Connected	Serial	2 threads	4 threads	8 threads
10 nodes	0.000006	0.000102	0.000151	0.000222
20 nodes	0.000016	0.000106	0.000163	0.00026
50 nodes	0.000061	0.000161	0.000218	0.000376
100 nodes	0.000207	0.000323	0.000432	0.000567
1000 nodes	0.033503	0.040029	0.032328	0.053831
5000 nodes	1.056755	0.917145	0.826626	0.96174
10000 nodes	5.31742	4.323927	3.943249	4.275426

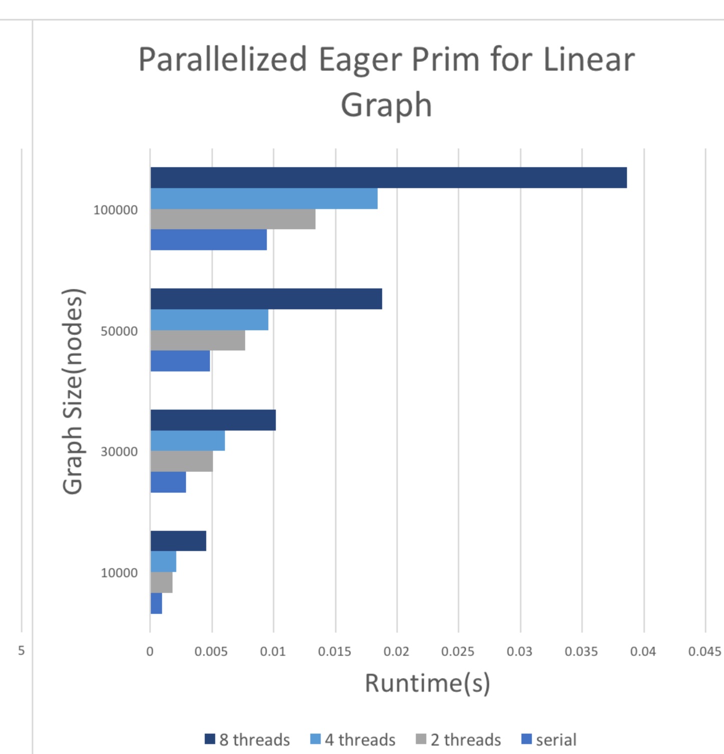
Half-Full	Serial	2 threads	4 threads	8 threads
3000 nodes	0.187135	0.166533	0.160802	0.236557
5000 nodes	0.557811	0.468449	0.428743	0.520169
10000 nodes	2.259167	2.011236	1.922603	2.09933
20000 nodes	14.857574	10.235791	9.57303106	10.17263











Conclusions

- **Parallelization cause improvement for dense graphs on large size**
  - Speedups of 1.34 and parallel efficiency of 33.7% for fully connected graph on 10000 nodes
  - Speedups of 1.17 and parallel efficiency of 29.3% for half-fully connected graph on 10000 nodes
- **Parallelized Eager Prim achieves best performance for highly clustered graphs with speedups up to 3.35 and parallel efficiency as high as 83.8% (20000 nodes)**
- **Parallelized Eager Prim performs poorly on sparse graphs**
- **Thank Dr. Alan George for providing this opportunity and thank Evan Gretok for guidance and support**