Parallelizing Dinic’s Algorithm for Max Flow

Matei Budiu and Benedict Ozua

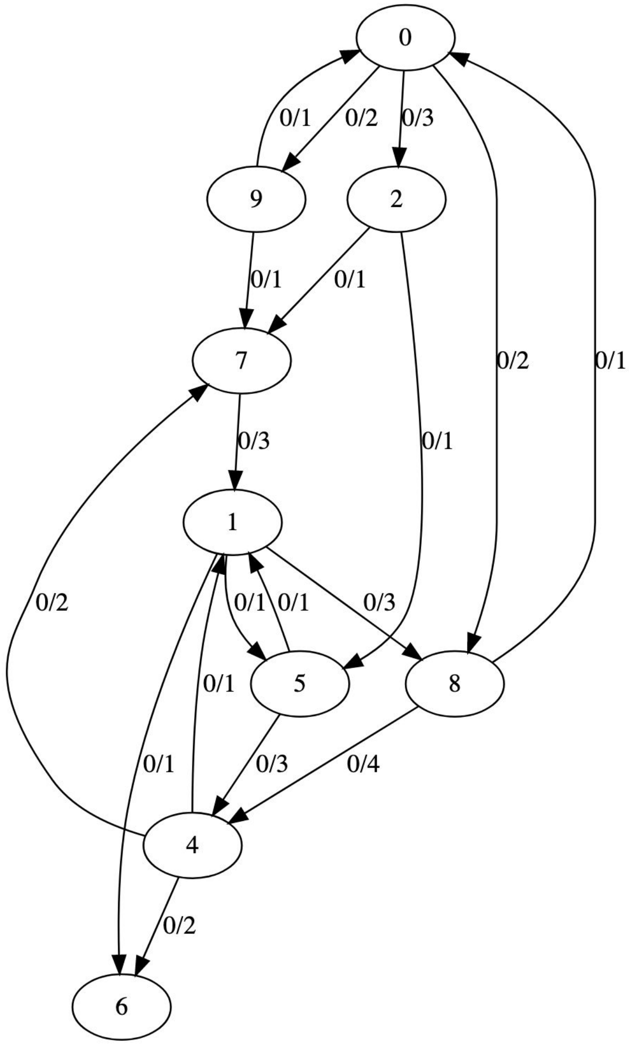
We attempted several different approaches to parallelizing Dinic’s algorithm for max flows in flow networks, which is primarily composed of depth-first search and breadth-first search.

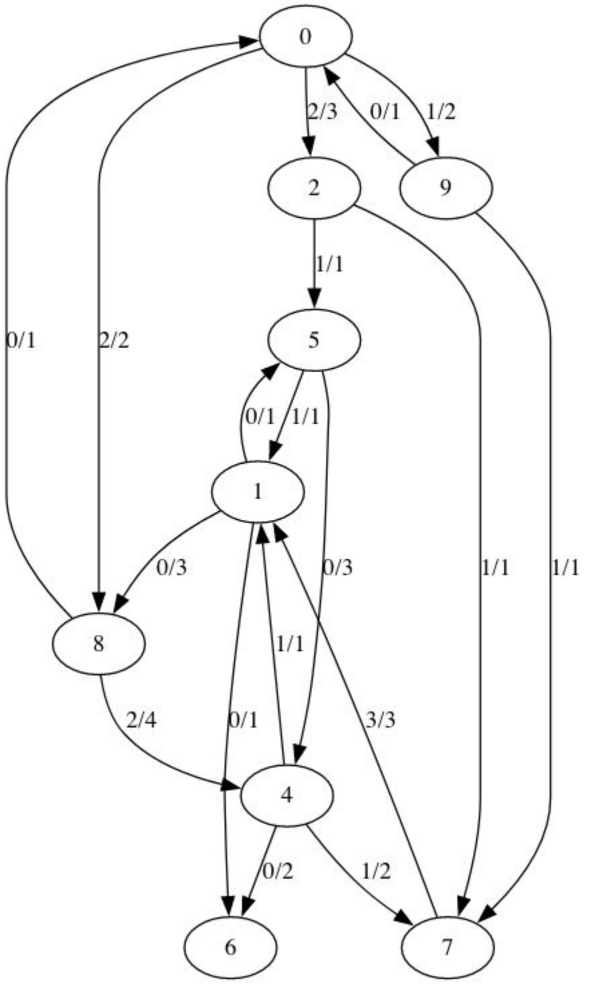
We found that BFS was by far the most time consuming part of the algorithm, and took several approaches to parallelizing it.

Some included using a per-vertex parallelism approach, and using GPU acceleration with CUDA.

What is a max flow

- Directed graph
- Edges have capacities of flow they can carry
- How much flow can be pushed between the source and sink vertex?



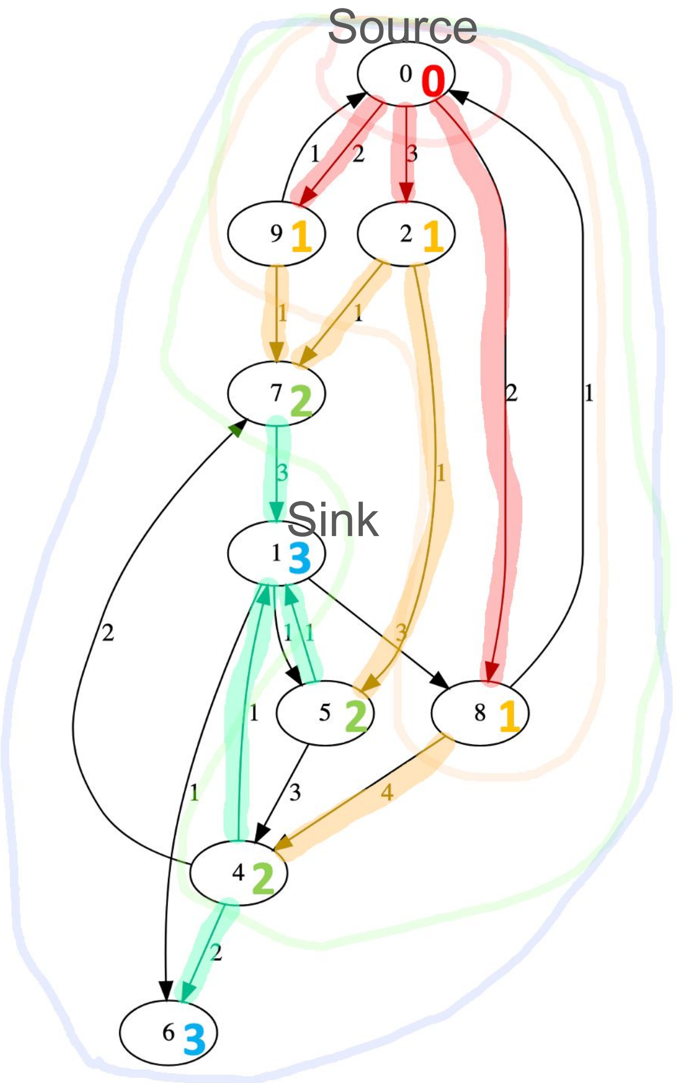


Dinic’s algorithm for Max Flow

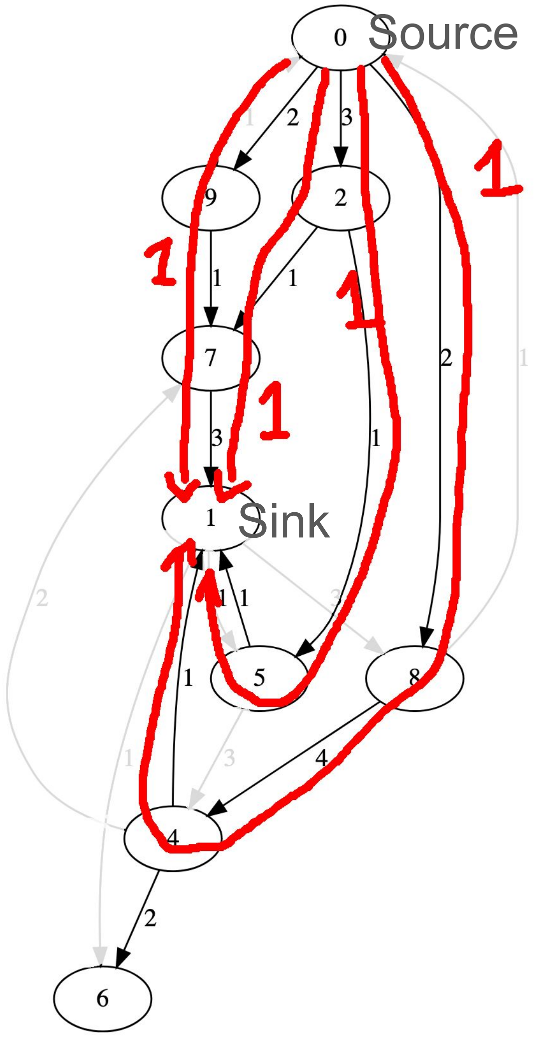
An $O(mn^2)$ algorithm with 2 repeating steps:

- Create a directed acyclic level graph with a BFS
- Repeatedly run DFS through the level graph to find paths from source to sink
    - Push flow through these paths
    - Stop when there are no more paths - a blocking flow

The level graph after BFS

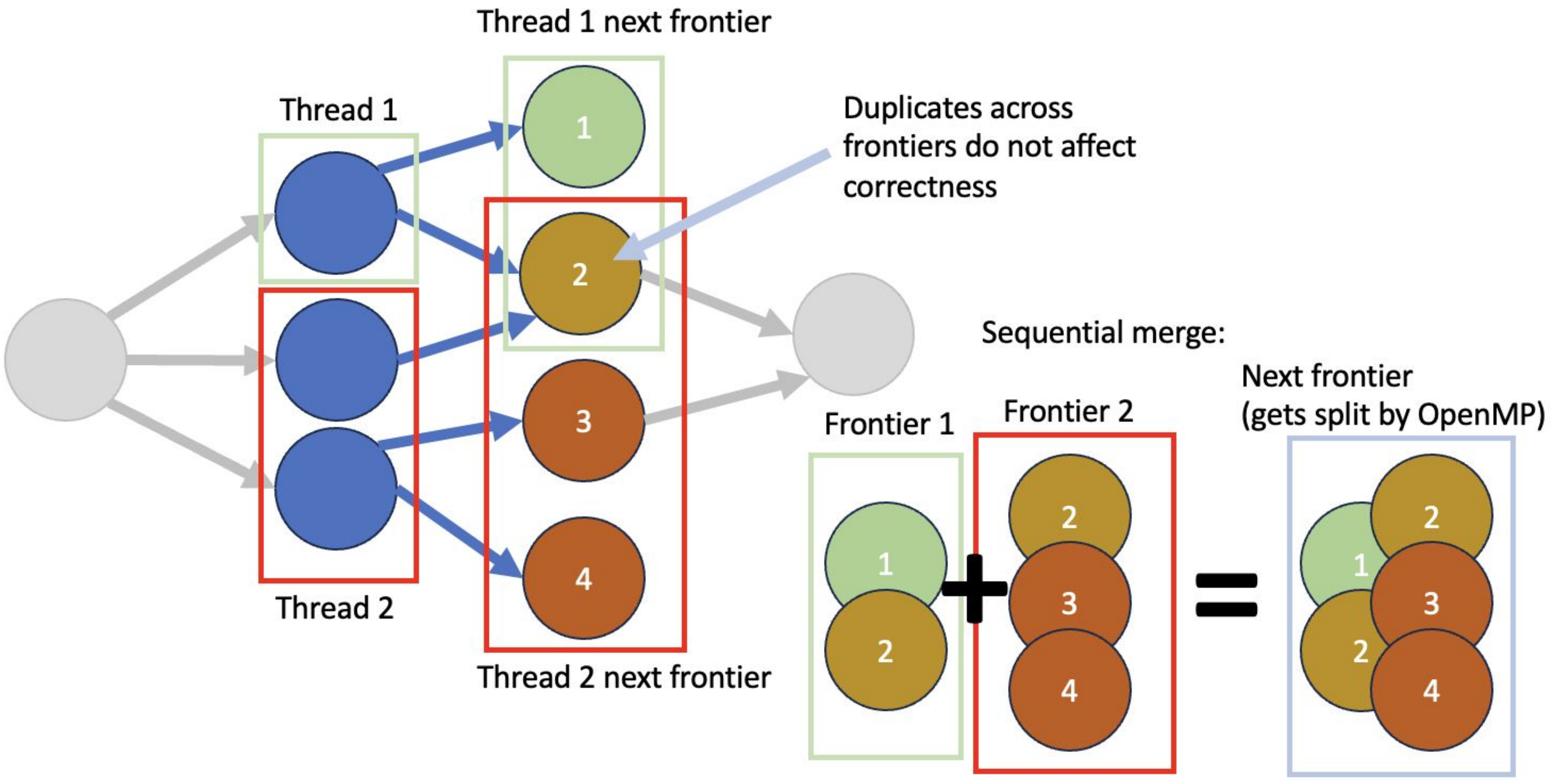


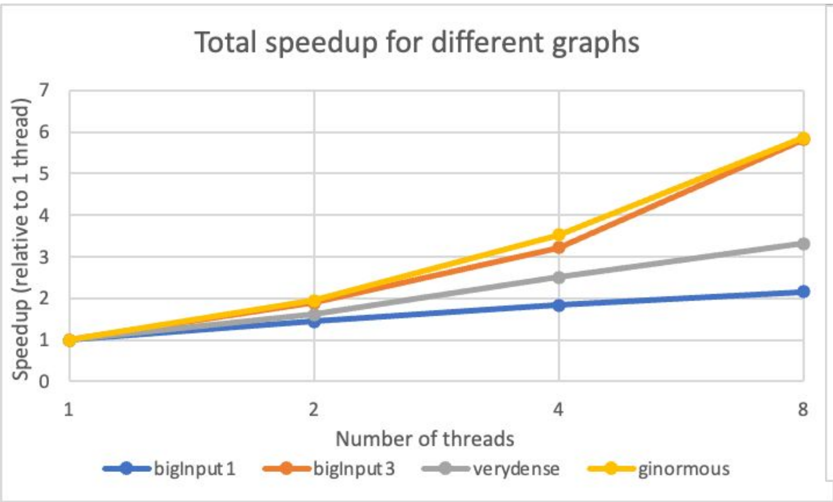
A blocking flow in that level graph (from 4 DFS)

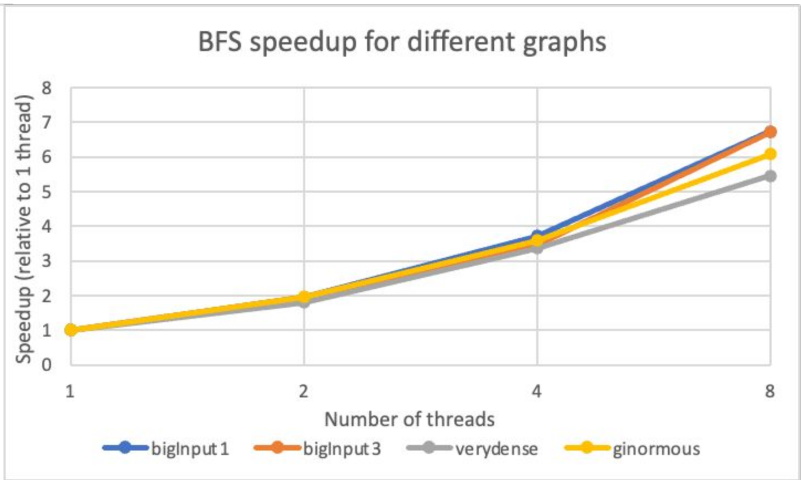


FastBFS (frontier + merge)

- Parallelize per vertex in frontier
- Each thread gets its own frontier
- Frontiers are merged sequentially (not a bottleneck)
- No locks prevent the same vertex to appear in multiple frontiers, duplicates caught later
- Separate layered-graph neighbors per thread, loop through all in sequential DFS

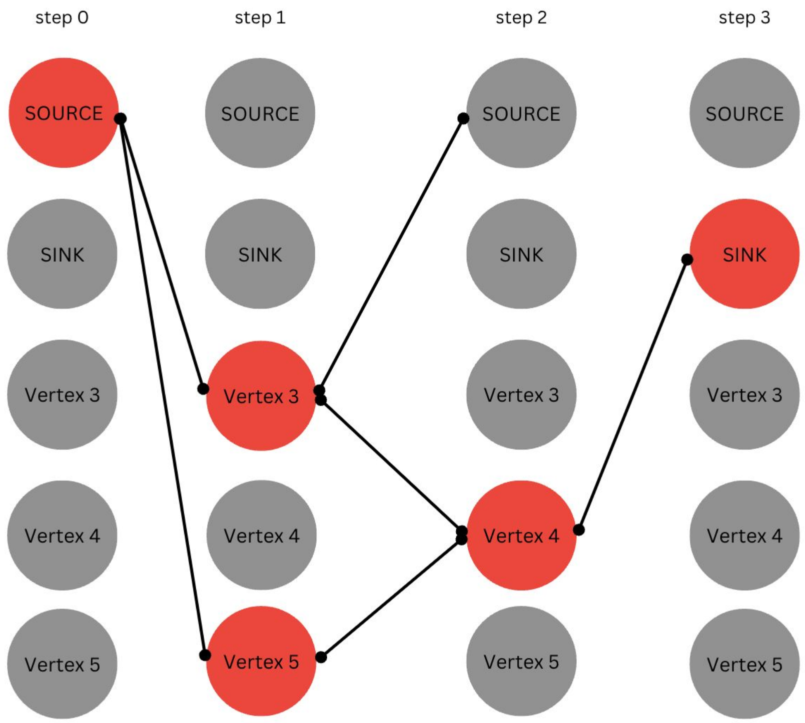


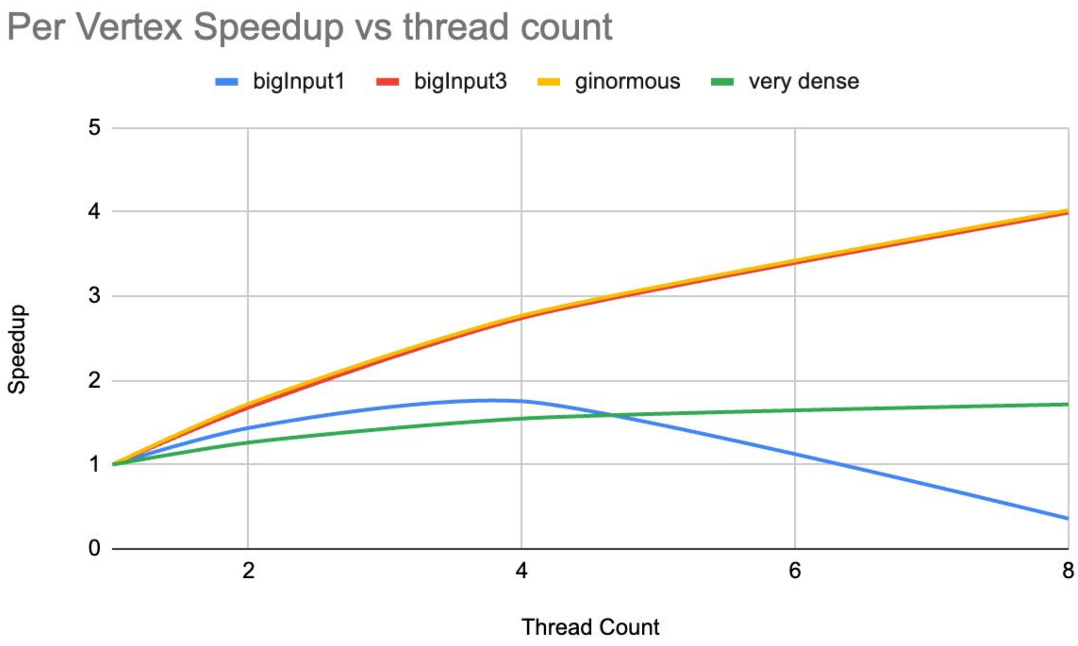




Per-Vertex

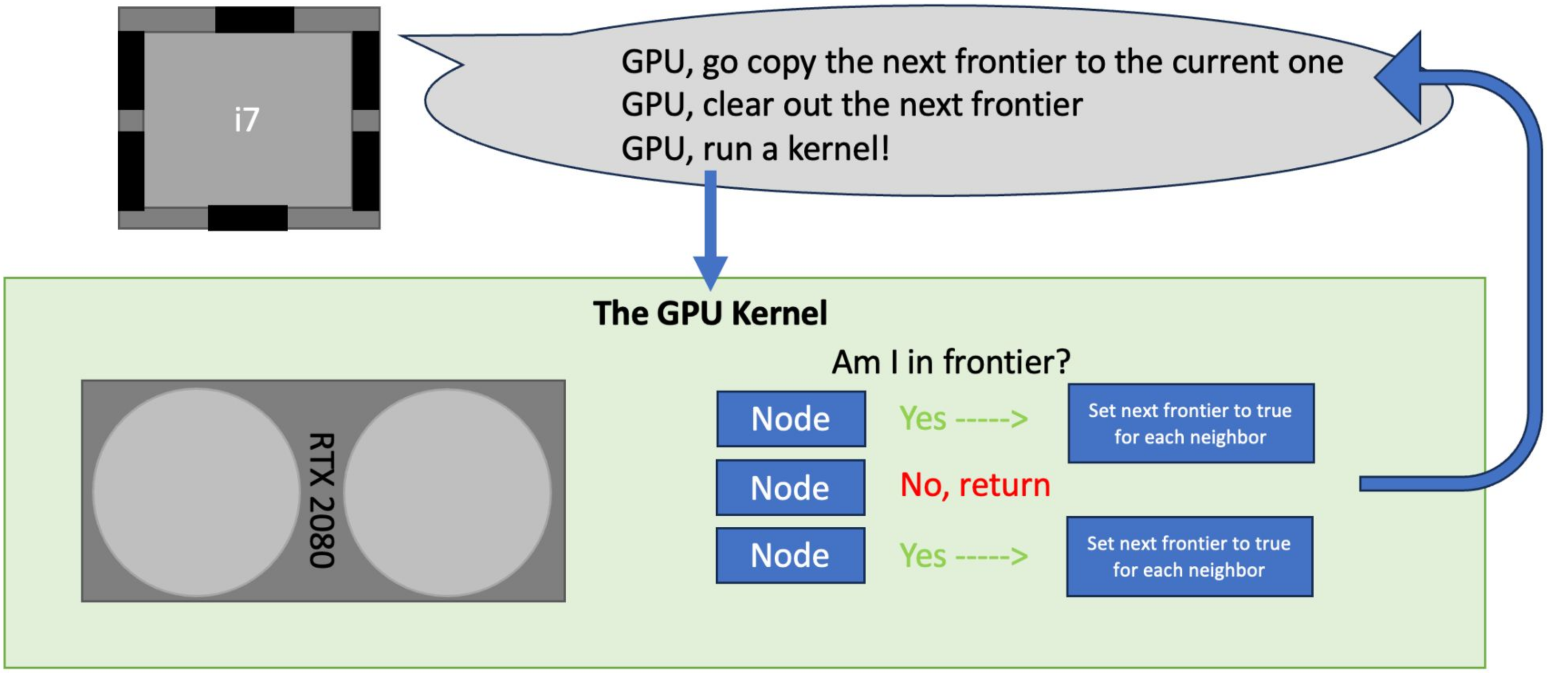
- Loop over all vertices in parallel
- See if node in step before advancing its neighbors





CUDA

- Each vertex gets its thread
- Each kernel advances the frontier one level



GPU, go copy the next frontier to the current one
GPU, clear out the next frontier
GPU, run a kernel!

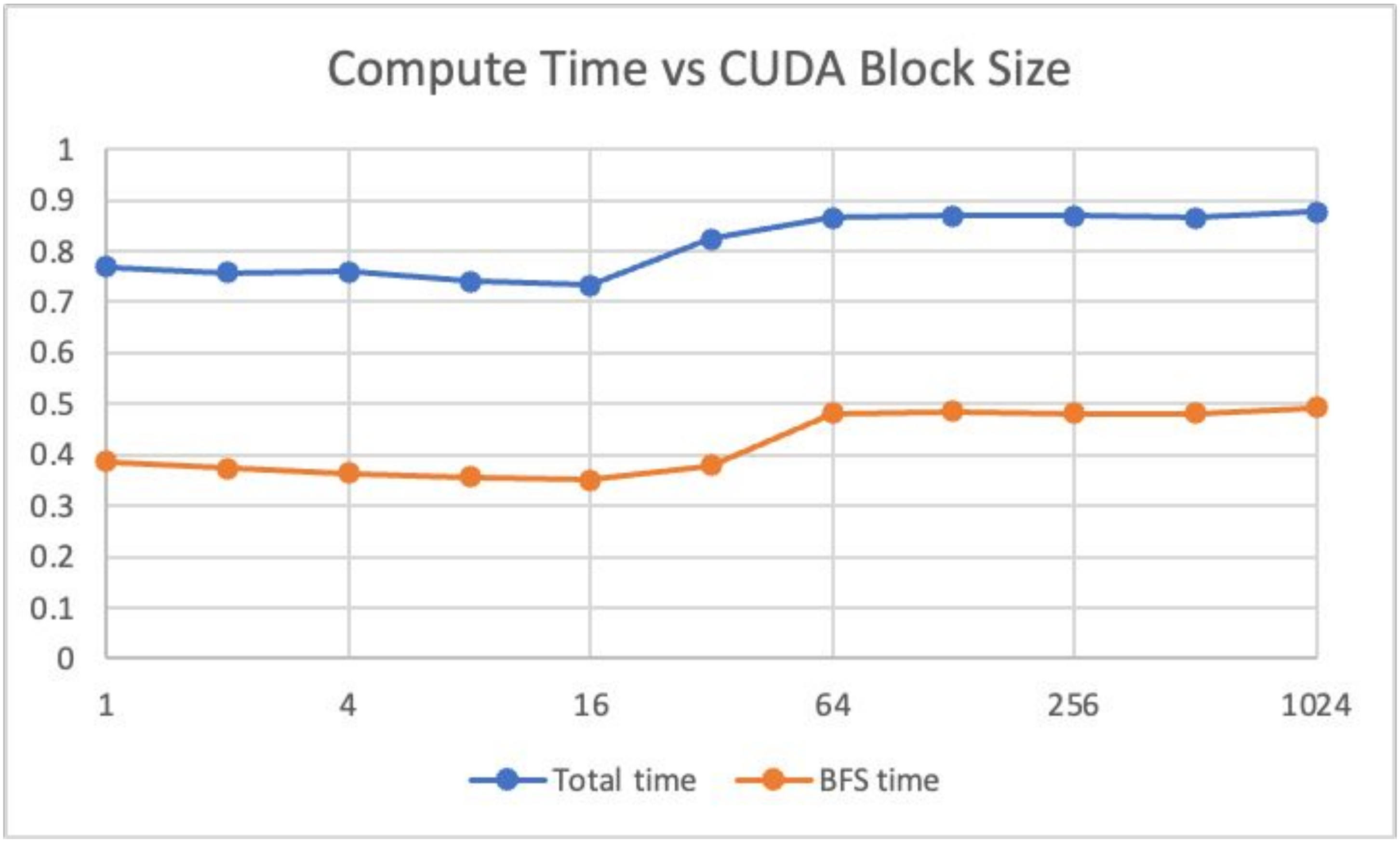
The GPU Kernel

Am I in frontier?

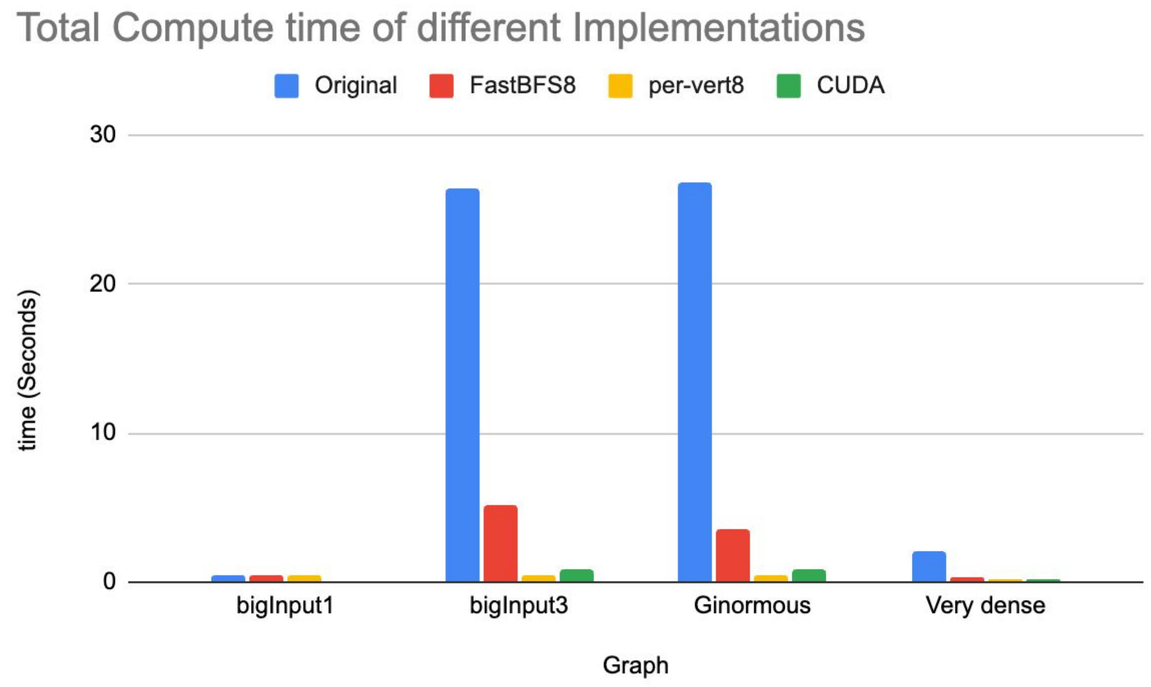
Node Yes -----> Set next frontier to true for each neighbor

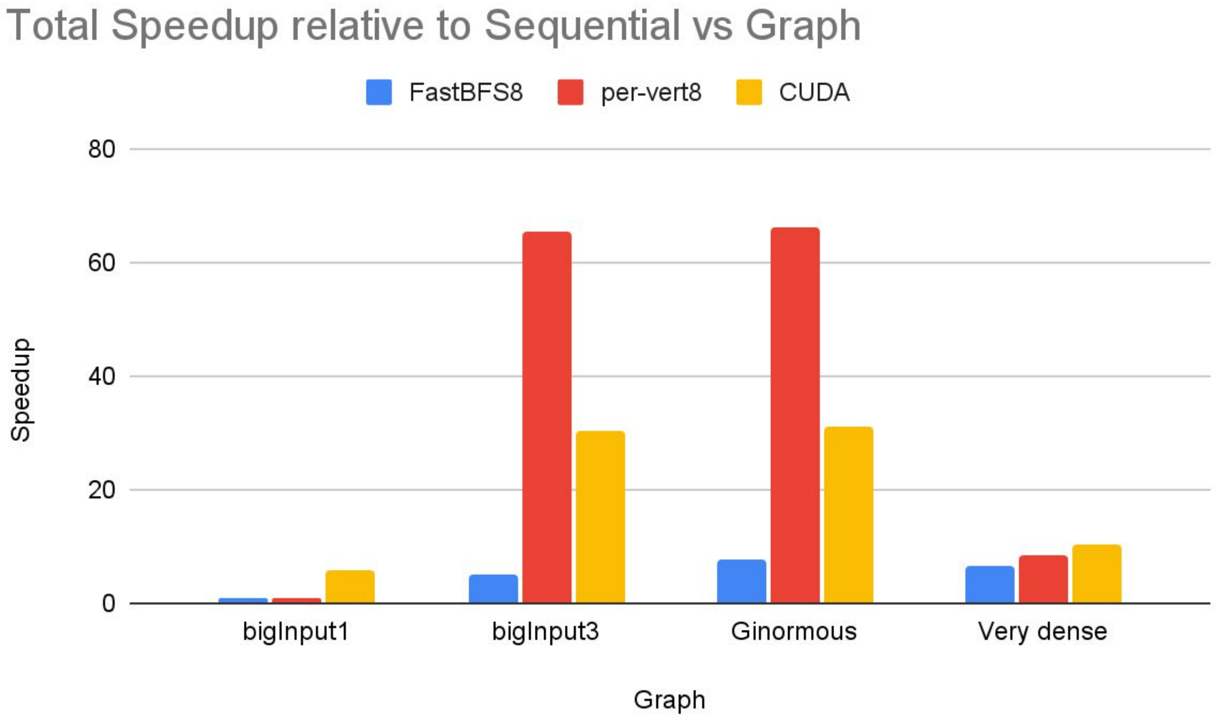
Node No, return

Node Yes -----> Set next frontier to true for each neighbor



Implementation Performance

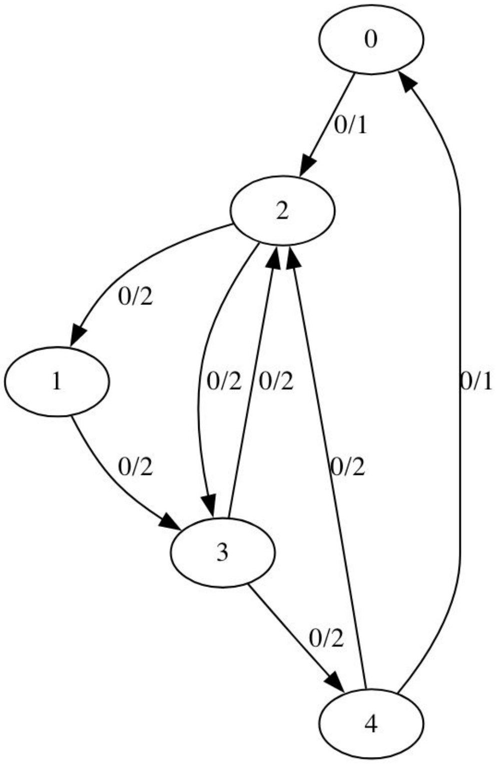




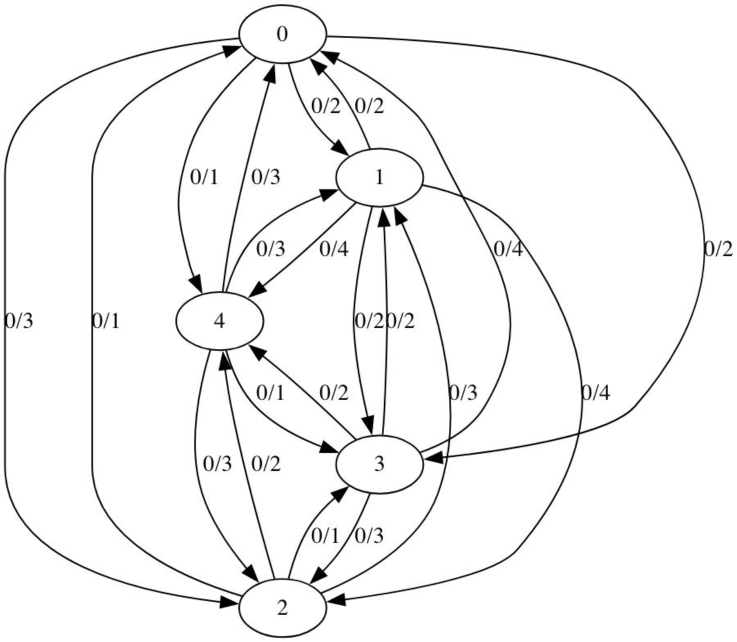
Graph Characteristics

- CUDA always outperforms FastBFS
- Per-Vertex outperforms CUDA in some cases

Sparse Graph
FastBFS does well



Dense Graph
Per-Vertex does well


0
0/2
0/2
0/1
0/3
1
0/3
0/4
0/4
0/2
0/3
0/1
4
0/2
0/2
0/1
0/2
0/3
0/4
0/3
0/2
3
0/1
0/3
2