Parallel Maximum Flow

Siyuan Chen, Xinyue Yang

15-418 Parallel Computer Architecture and Programming, Spring 2024

Summary

We parallelized two maximum flow algorithms (Edmonds-Karp and Dinic's) under the shared address space model using OpenMP. We evaluated their performance on GHC and PSC machines against different network types. We demonstrated that

- across the two algorithms, the former is more parallelizable but overall the latter is more performant; and
- within each algorithm, the top-down and bottom-up parallelism strategies are more suitable for sparse and dense networks, respectively.

Background

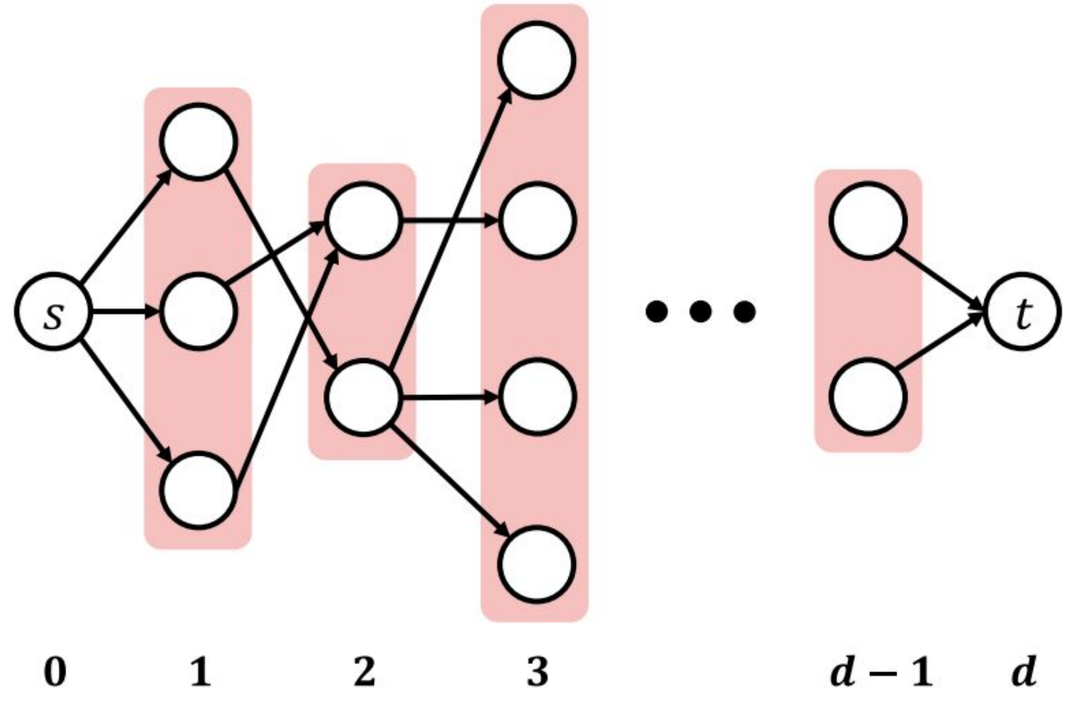
Augmenting-path-based maximum flow algorithms:

```
while there is an augmenting path in the residual network:
   push flow along that augmenting path
```

Edmonds-Karp and Dinic's constructs the layer network to help identify such augmenting paths.

Edmonds-Karp then pushes flow through a single shortest augmenting path.

Dinic's instead pushes flow through all shortest augmenting paths iteratively.



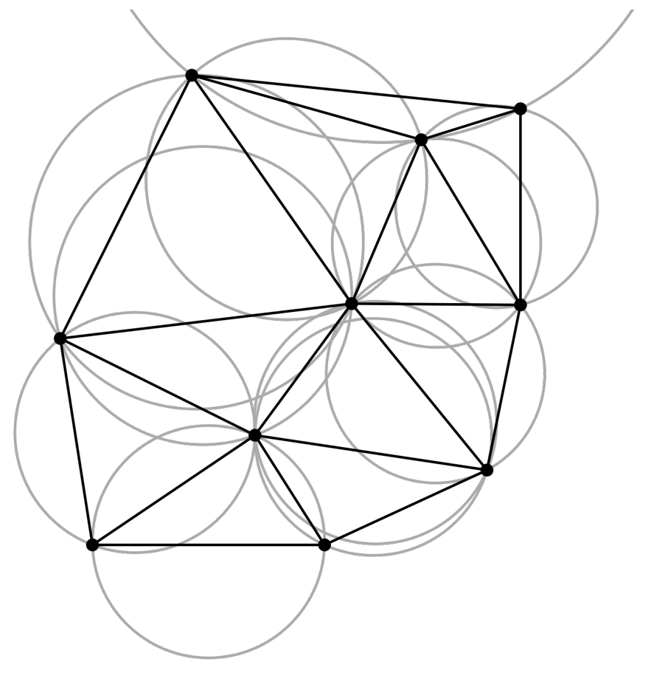
Networks

We evaluated the performance of our sequential and parallel algorithms on several different types of networks:

- random graphs
- dense graphs (cliques)
- sparse graphs (from Delaunay triangulations)
- grid graphs (fully connected networks)

We care about the following characteristics:

- average degree of a vertex
- typical “width” (frontier size)
- typical “depth” (augmenting path length)



Sparse graph

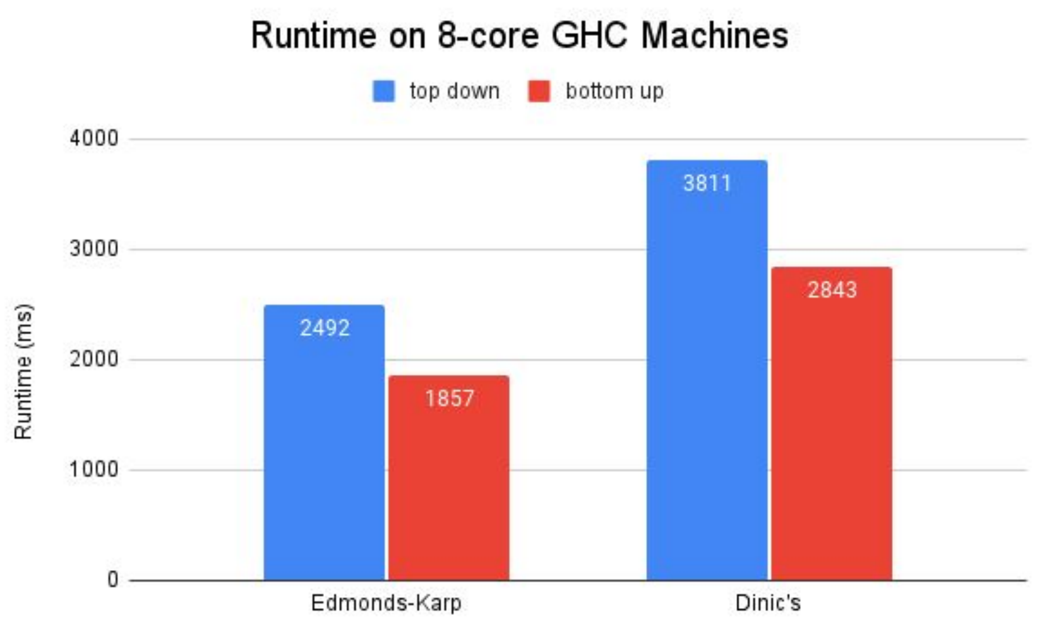
Parallelism strategies

Push flow is inherently sequential, so we parallelize the build layers step (essentially extending the frontier in breadth-first search).

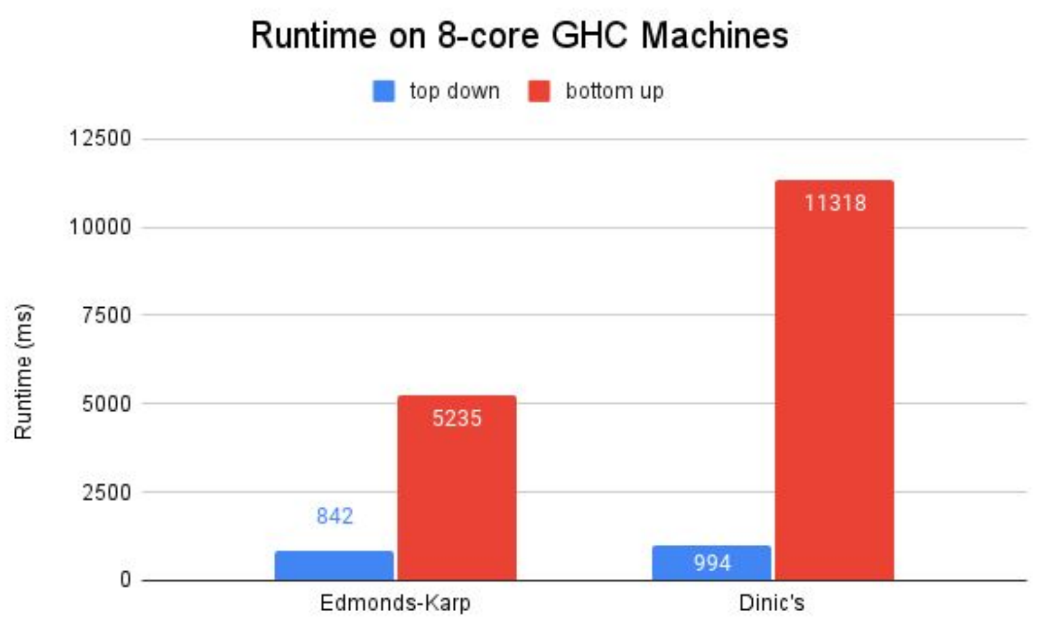
- The **top-down** strategy parallelizes across the current frontier and concurrently constructs the next frontier.
  - tried **coarse-** and **fine-grained locking**; settled with **atomic CAS**
- The **bottom-up** strategy parallelizes across the vertices and determines individually whether it belongs to the next frontier.
  - requires no explicit synchronization; experimented with **scheduling**

GHC results

Dense graphs afford **better parallelism** for the bottom-up strategy, whereas top-down has less **overhead** and **artifactual computation**, and is more suitable for sparse graphs.



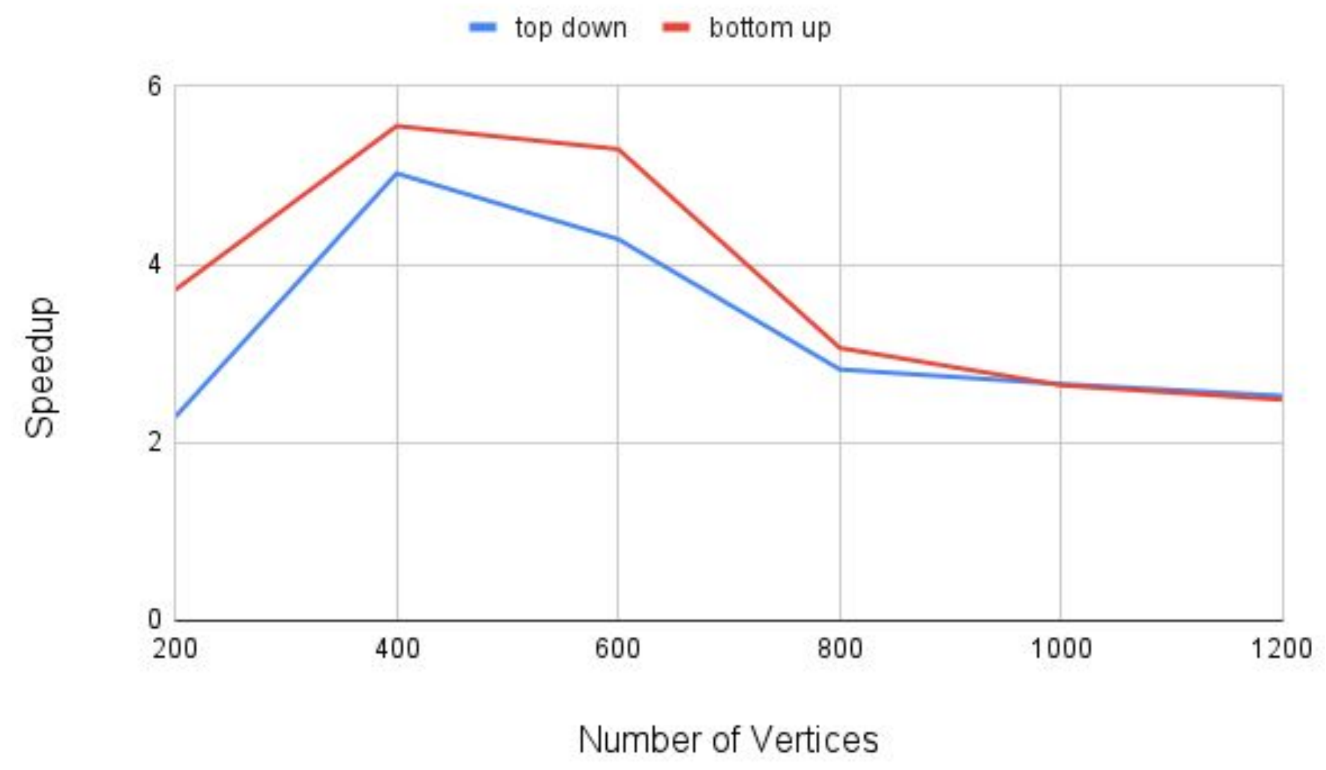
Dense graph performance



Sparse graph performance

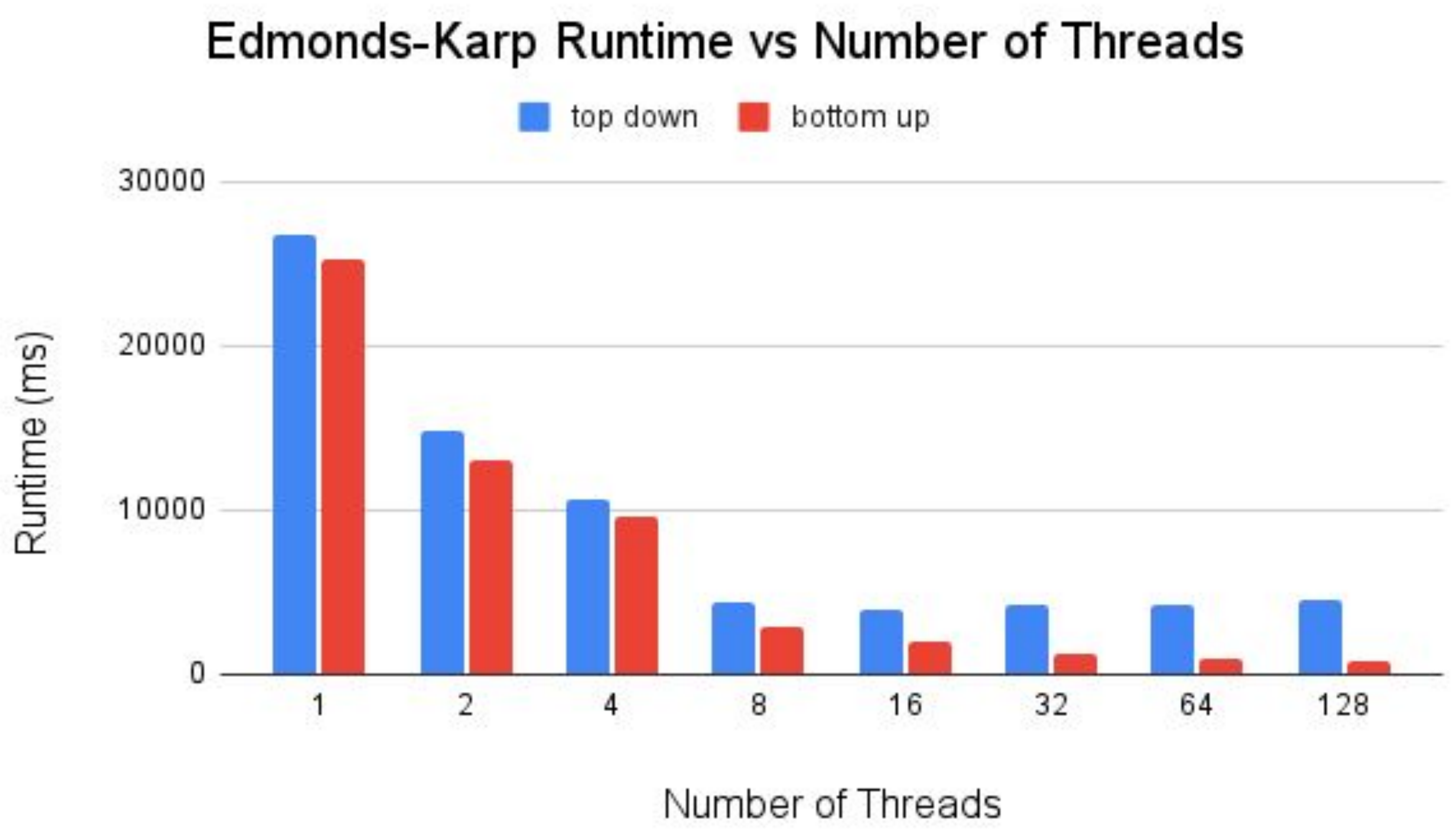
Problem size sensitivity

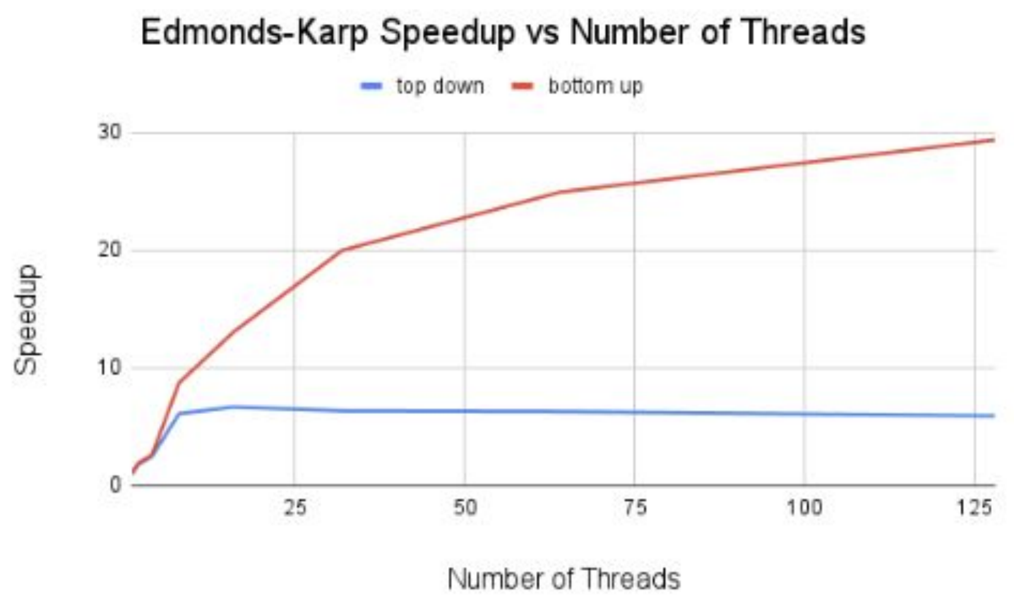
For Edmonds-Karp on dense graphs, speedup for 8 cores plateaus as number of vertices increase past a threshold. This illustrates that **cache effects** impact the performance of both parallelism strategies.

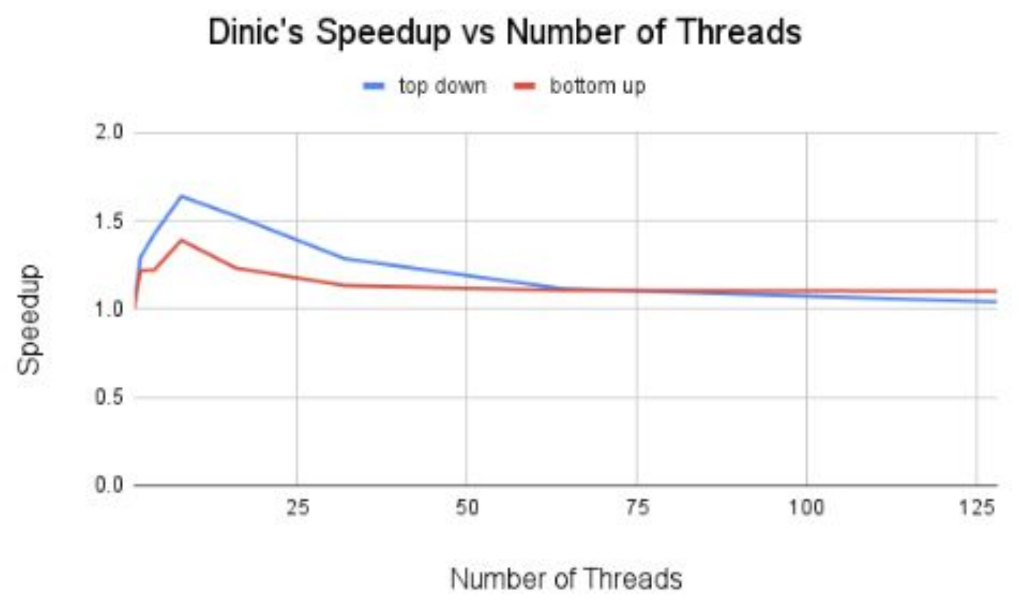


PSC results

For Edmonds-Karp on dense graphs, as the number of cores increase, the top-down approach achieves **limited speedup** due to **poor utilization** of the parallel threads, whereas bottom-up proves to be more scalable, achieving a **30x speedup** at 128 cores.







Analysis

We attribute the theoretical-practical speedup gap to the following factors:

- **Amdahl’s law**: the inherent sequential nature of push flow limits speedup for Edmonds-Karp (to a lesser extent) and Dinic’s (to a greater extent);
- **poor utilization** (top-down): small frontier size hinders parallelism
- **artifactual computation** (bottom-up): sequentially inefficient computation overshadows parallelism benefits
- **cache effects** for larger networks

… and to a lesser extent:

- **memory contention** (top-down): occasional locking conflicts
- **abstraction overhead** due to OpenMP

References

1. 15-451/651: Network Flow II, lecture notes, School of Computer Science 15-451, Carnegie Mellon University, Fall 2023.
2. 15-210 Lecture Notes, Fall 2022, Parallel Graph Algorithms
3. Delaunay Graphs. 10th DIMACS Implementation Challenge
4. Delaunay Triangulation, Wikipedia