Carnegie Mellon University

Carnegie Mellon School of Computer Science

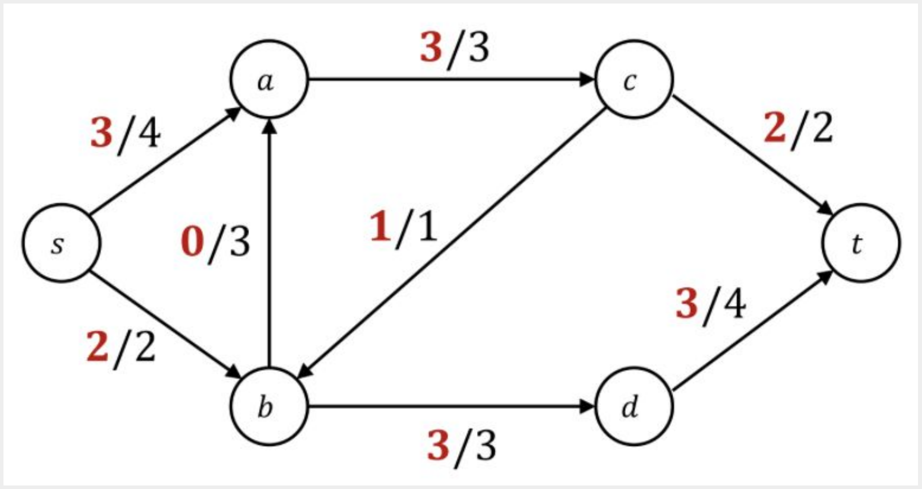
Background

Summary

- We parallelized Dinic's algorithm using the shared-address-space framework OpenMP.
- We evaluated the performance of our two parallel versions, the edge flow reservation approach and the disjoint paths approach, on various inputs using CPU machines and analyzed their trends of scalability and sensitivity to inputs.
- We demonstrated the overall desirability of the edge-disjoint approach and showed the input graph characteristics such as density and size have noticeable impact on speedups.

Maximum Flow

- Directed graph with edge capacities
- Flow constraints
  - For all edges, *flow* ≤ *capacity*
  - For all nodes except source and sink, *flow in* = *flow out*
- What's the maximum flow that can be sent from source to sink?
- Many applications: network routing, bipartite matching, image segmentation….



Background

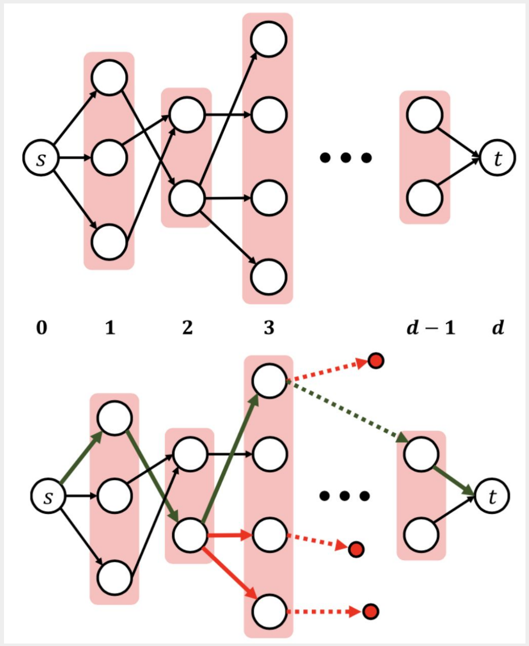
Dinic's Algorithm

```
while successfully builds a level graph:
    while can find an augmenting path p:
        send flow along p
        update residual capacities
    return total flow from s to t
```

- Alternating phases until converge

Build Level Graph
- Level as implicit graph
- Building frontiers

Compute Blocking Flow
- Find augmenting paths
- Mark used/dead edges
- Multiple augmenting paths for a blocking flow



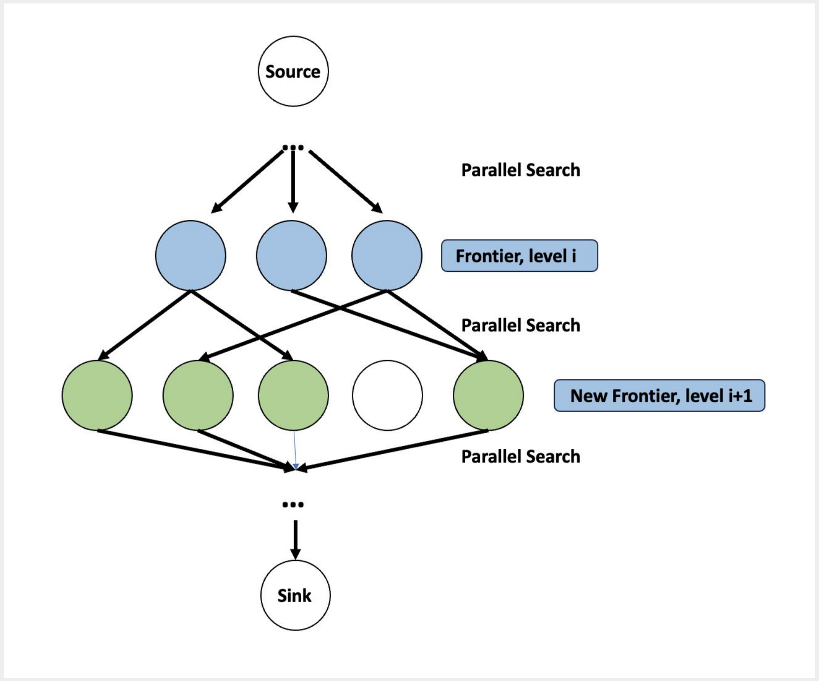
Inputs

Sparse Graphs:
- #edges far from maximum
- Randomly generated given specified number of vertices and out-degree
- Randomized capacities

Dense Graphs:
- #edges "close" to maximum
- Structurally generated as layered graphs with specified size and depth
- Randomized capacities

Why control size/density/depth?
- Sensitivity studies

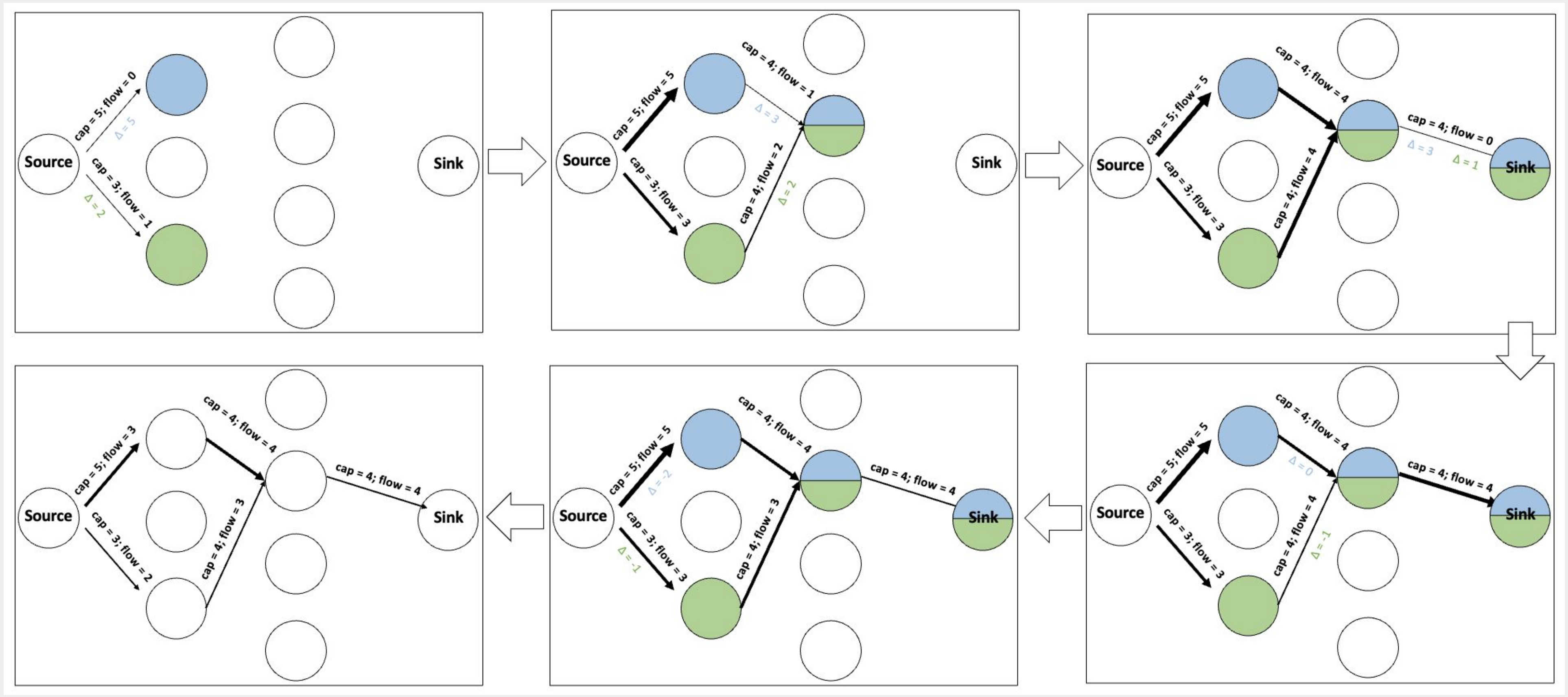
Method: Parallel Level Construction



- Builds level graph one layer at a time
- Data parallelism: Parallelized neighborhood search over vertices within a layer
- Merge all the neighborhoods of vertices in the previous layer to form the new frontier (layer)
- Issue: Without further management, repeated vertices may be added, the number of repeated vertices explode as the search goes to higher levels
- Solution: Use atomic compare and exchange primitives to ensure each vertex is only added once
- Synchronization: adding vertices to the global new frontier list must be protected as critical sections

Method: Parallel Path Augmentation

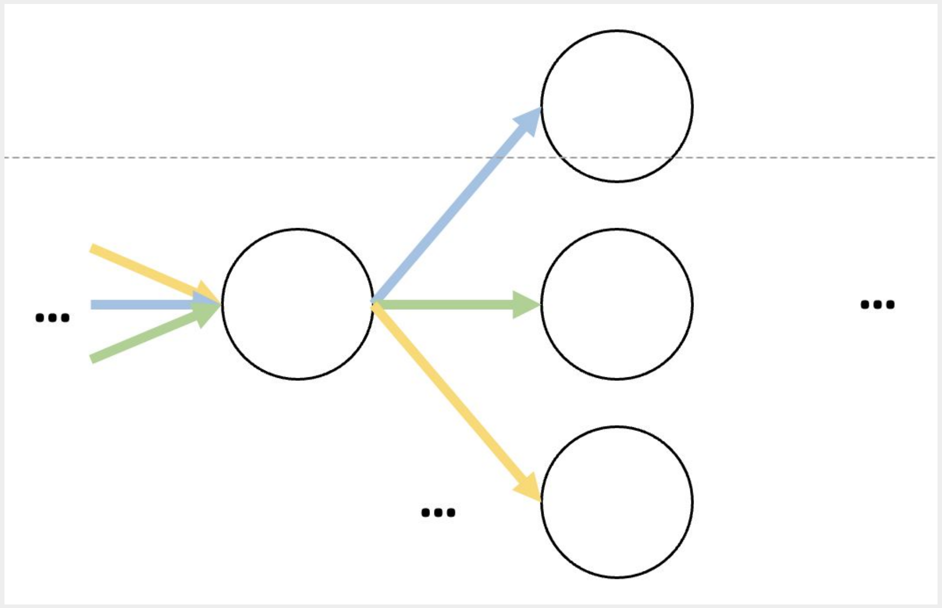
Edge Flow Reservation Approach



- Parallelly consider several path candidates
- Augmented each path and reserve blocking flow
- Backtrack after reach sink
- Return excessively reserved flow

Method: Parallel Path Augmentation

Disjoint Paths Approach

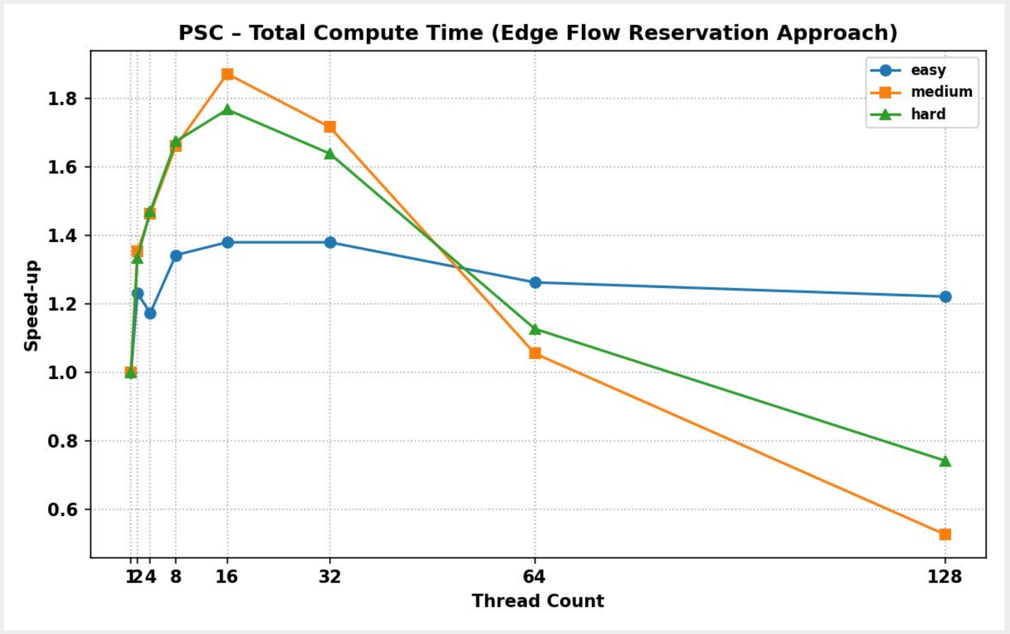


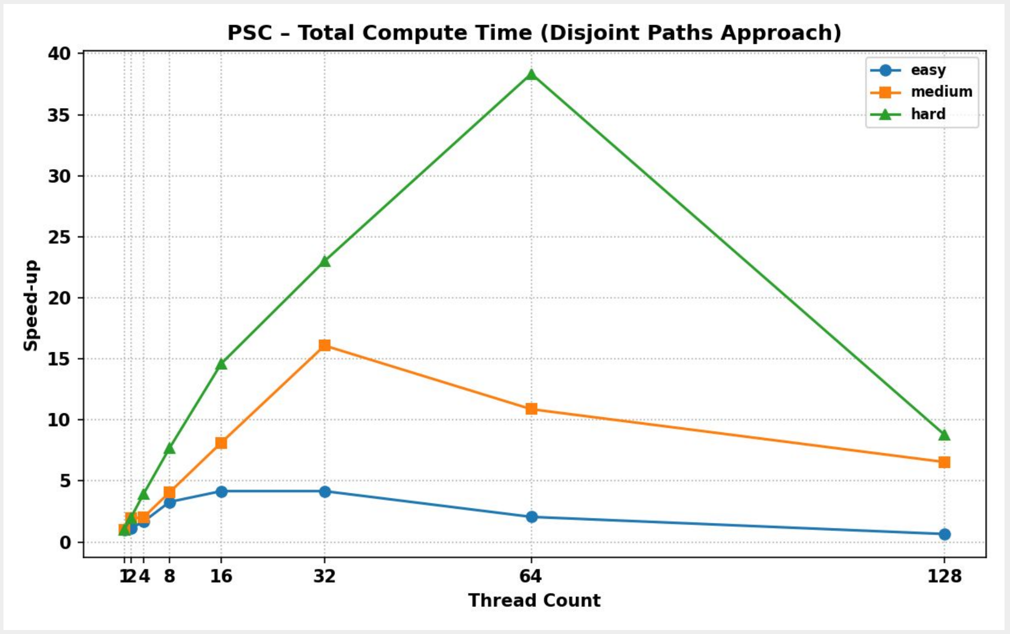
- Parallelism over augmenting path searching directions
- Multiple threads explore edge-disjoint augmenting paths
- Recall that a next-edge pointer array is maintained to ensure one edge is only traversed once
- Synchronization: maintain the atomicity of operations on next-edge pointer arrays to ensure correctness of traversal
- Simultaneous updates in edge-disjoint paths: no data race issue since threads don’t update the same edge

Results

Scalability

- Randomly generated, relatively sparse large inputs
- Disjoint paths approach has better scalability overall
- Edge flow reservation approach provide little speedup: high contention and extra rollback work
- Decreasing performance at high thread counts: contention with coincided next edge pointer accesses
- Bad locality: irregular pointer chasing, high cache miss rate

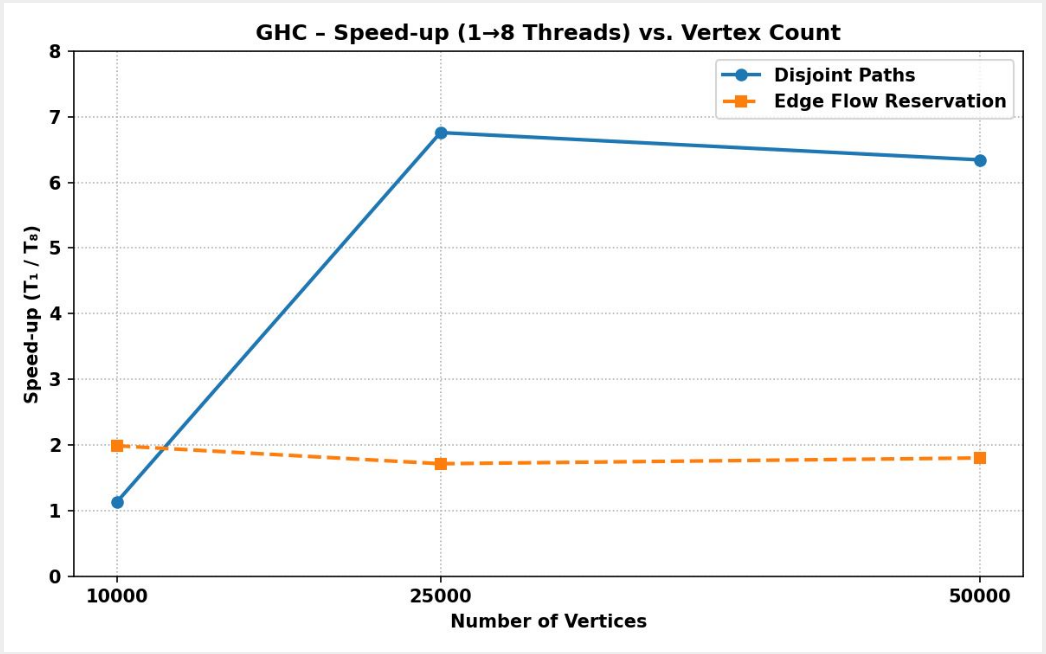


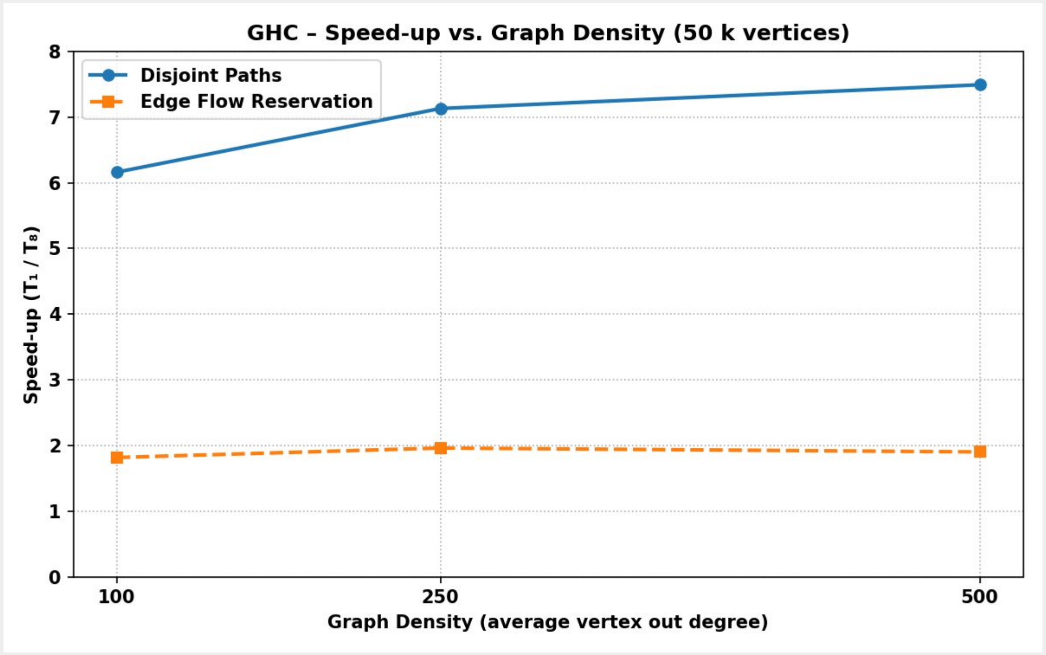


Results

Sensitivity

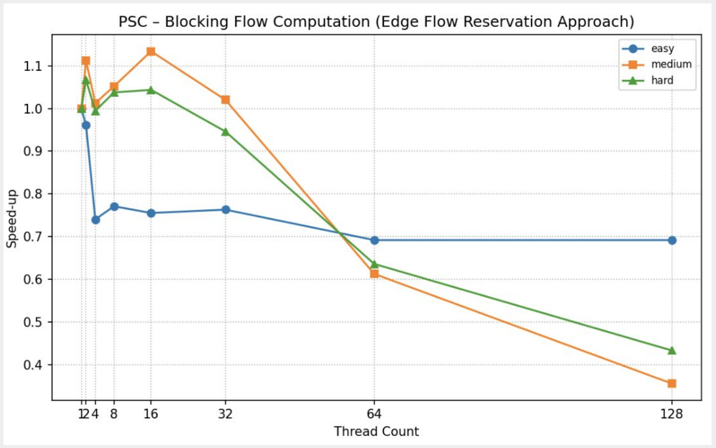
- 8-thread speedups
- Edge flow reservation approach shows low sensitivity: overall inefficiency dominates behavior
- Disjoint path approach shows higher speedup for denser graphs: parallelization over outgoing edges
- Disjoint path approach shows higher speedup for larger graphs: balanced workload and less contention

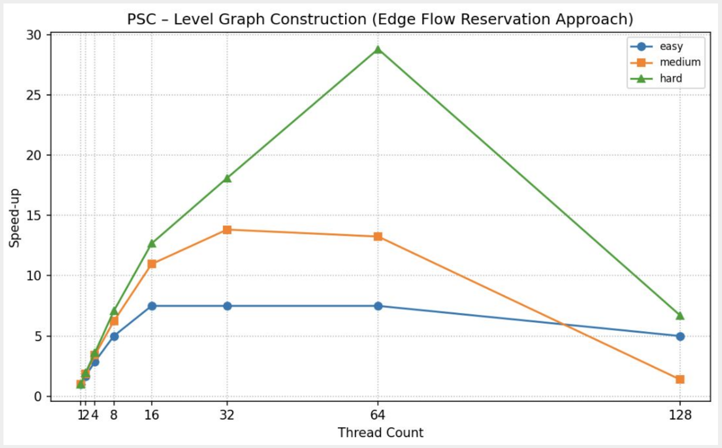


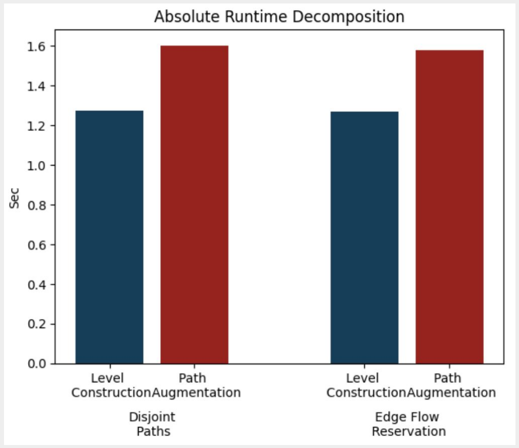


Results

Deeper Analysis







- Edge flow reservation approach has acceptable speedup for the first phase but terrible slowdown in the second phase
- Edge flow reservation approach suffers from contention and extra work that scale with thread count
- Runtime breakdown for two approaches tested on the standard hard input: no drastic difference between runtimes spent in the two phases
- Parallelism in level graph construction phase and blocking flow computation equally influential to overall compute time speedup.