Work-efficient parallel union-find

Natcha Simsiri, Kanat Tangwongsan, Srikanta Tirthapura, Kun-LungWu

Presented by Áron Ricardo Perez-Lopez

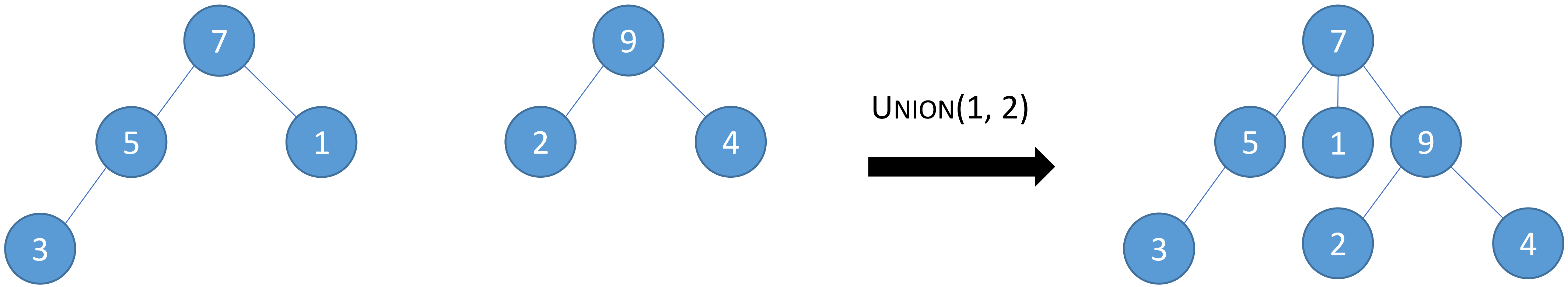
Incremental Graph Connectivity

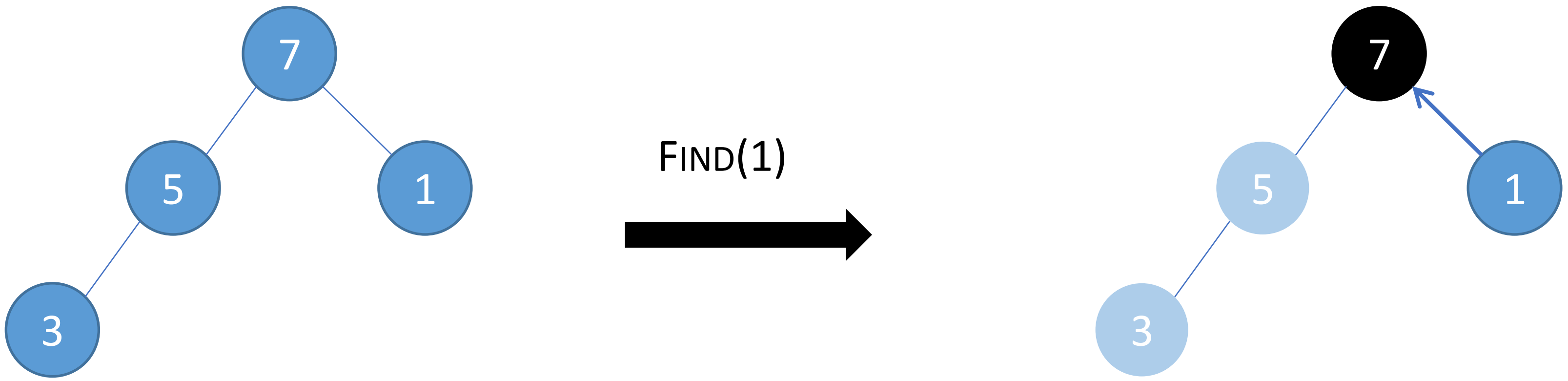
- Maintains information about connected components.
- Edges can be added.
- No edge deletion.
- **Model of computation**: homogeneous batches of queries.

Union-Find Data Structure

- **UNION(*u*, *v*)**:
  - Combine sets containing *u* and *v*.
  - Return the combined set.
- **FIND(*u*)**:
  - Return the set containing *u*.
- Complexity:
  - $O((m+q)\alpha(m+q))$ for m UNION and q FIND operations.
  - Equivalently, $O(\alpha(n))$ amortized for both operations.
- Applications:
  - Kruskal's algorithm.

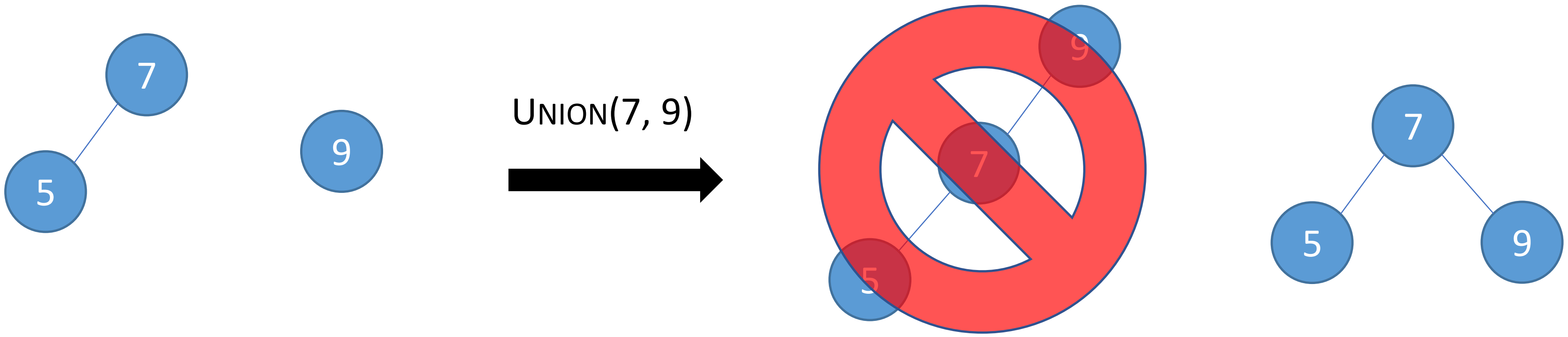
Basic idea: Trees!



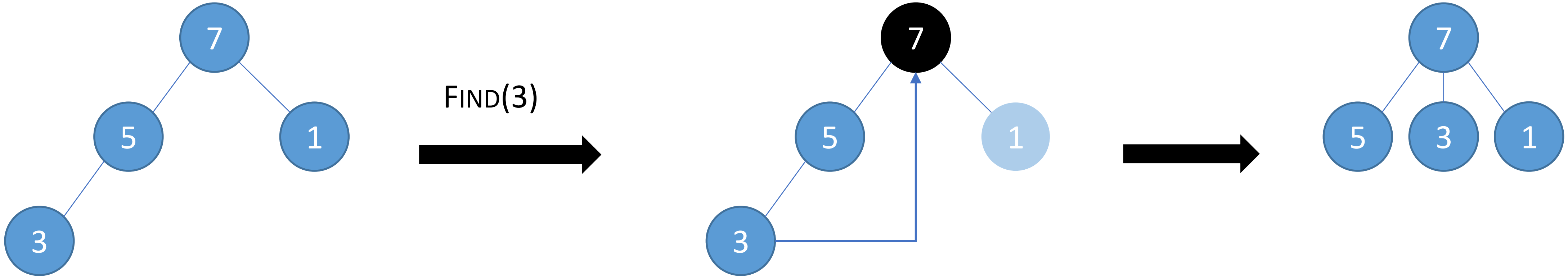


Improvements

- Union by size (or rank)



- Path compression



Parallelizing Union

- Can we just execute all operations in parallel?
  - No, because of races.
  - Also, not efficient.
- **Idea 1**: group operations into disjoint sets.
- **Idea 2**: join one set using divide-and-conquer.
- Algorithm: **BULK-UNION**
  1. Relabel each edge with its root.
  2. Remove self-loops.
  3. Compute connected components in this graph.
  4. Join each component in parallel. Within one component:
     1. Divide edges in half (minus middle edge) and recurse.
     2. Add middle edge.

Parallelizing Find – the Simple Way

- Without path compression, FIND is read-only.
- We can just execute all queries in parallel.
- Runtime is $O(\log n)$.
- Can we do better?

Parallelizing Find – Two-phase Algorithm

- **Idea**: separate process into two parts: search and compress.
- Algorithm:
  1. **Search**: BFS from all queried vertices simultaneously.
     - Stop when we would repeat work.
     - Store (reverse) edges for second phase.
     - Store found roots.
     - All operations inside the loop are parallelized.
  2. **Compress**: BFS backwards, from the roots to the original vertices.
     - Also compute answers (roots) on the way.

Response Distributor

- **Challenge**: we only have an edge list for the second phase, but we want to compute the next frontier in linear work and polylog depth.
- Idea 1: sort edges by source vertex – practically equivalent to CSR.
- Idea 2: sort by hash instead of actual value.

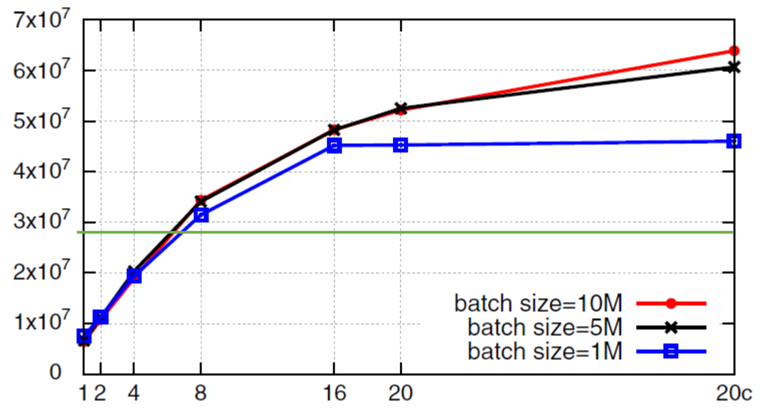
Experimental Results I: Serial Runtime

TABLE 2 Running times (in seconds) on 1 thread of the baseline union-find implementation (`UF`) with and without path compression and the bulk-parallel version as the batch size is varied

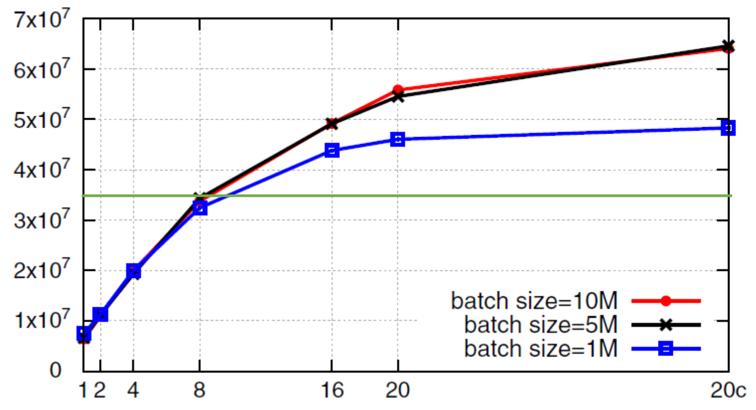
Graph	`UF` (no p.c.)	`UF` (p.c.)	Bulk-Parallel using batch size 500K	1M	5M	10M
random	44.63	18.42	65.43	66.57	75.20	77.89
3Dgrid	30.26	14.37	61.10	62.00	71.74	75.07
local5	44.94	18.51	65.84	66.77	75.33	78.23
local16	154.40	46.12	114.34	108.92	114.80	117.55
rMat5	33.39	18.47	66.98	68.48	74.97	78.69
rMat16	81.74	35.29	83.27	76.64	76.03	77.62

Experimental Results II: Parallel Speedup

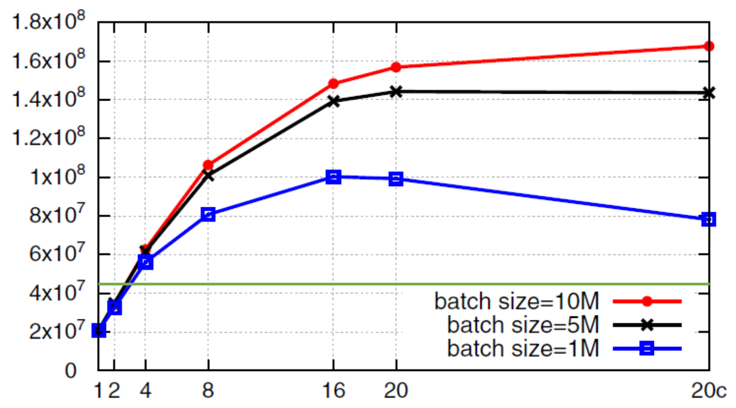
random



local16



rMat16



Green line: performance of optimized single-threaded implementation.

Discussion Questions

- What are your thoughts on the difference between the proposed algorithm and the one the authors actually implemented?
- How does this algorithm compare to the one from McColl et al? Is it worth exchanging the ability to remove edges for near-constant runtime?