Work-efficient parallel union-find

Natcha Simsiri, Kanat Tangwongsan, Srikanta Tirthapura, Kun-Lung Wu

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 \(m \text{ UNION}\) and \(q \text{ FIND}\) 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**: \textbf{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

<table>
<thead>
<tr>
<th>Graph</th>
<th>UF (no p.c.)</th>
<th>UF (p.c.)</th>
<th>Bulk-Parallel using batch size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>500K</td>
</tr>
<tr>
<td>random</td>
<td>44.63</td>
<td>18.42</td>
<td>65.43</td>
</tr>
<tr>
<td>3Dgrid</td>
<td>30.26</td>
<td>14.37</td>
<td>61.10</td>
</tr>
<tr>
<td>local5</td>
<td>44.94</td>
<td>18.51</td>
<td>65.84</td>
</tr>
<tr>
<td>local16</td>
<td>154.40</td>
<td>46.12</td>
<td>114.34</td>
</tr>
<tr>
<td>rMat5</td>
<td>33.39</td>
<td>18.47</td>
<td>66.98</td>
</tr>
<tr>
<td>rMat16</td>
<td>81.74</td>
<td>35.29</td>
<td>83.27</td>
</tr>
</tbody>
</table>
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?