Reducing Contention Through Priority Updates

Julian Shun (CMU)  Guy Blelloch (CMU)  Jeremy Fineman (Georgetown)  Phillip Gibbons (Intel, Pittsburgh)

Motivation

- Memory contention can be a serious performance bottleneck in concurrent programs on shared memory machines
- But shared memory accesses can be very useful in parallel algorithms, concurrent data structures, thread communication protocols, all of which are important in cloud-based applications

Sharing vs. Contention

- **Sharing**: Concurrent access to shared memory resource by multiple cores
- **Contention**: Performance penalty associated with sharing

Priority Update

- Takes as arguments a memory location `addr`, a new value `newval` and a comparison function `>_p`
- Atomically compares the value at `addr` with `newval` and stores `newval` at `addr` if and only if it has a higher priority according to `>_p`
- Implemented with compare-and-swap (CAS)
- Pseudocode:

  ```
  procedure PriorityUpdate(addr, newval, >_p)
  oldval = *addr
  while(newval >_p oldval) do
    if compare-and-swap(addr, oldval, newval) then
      return
    else
      oldval = *addr
  ```

- Test & Set: special case of Priority Update over values {0,1}

Concurrent Operations in Action

<table>
<thead>
<tr>
<th>Reads</th>
<th>Test &amp; Set</th>
<th>Write/CAS/Fetch&amp;Add</th>
<th>Priority Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>R</td>
<td>W</td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

Applications of Priority Update

- Deterministic algorithms: using priorities to resolve conflicts (breadth-first search, maximal matching, Delaunay triangulation)
- Correctness of algorithms: using write-with-min in shortest paths, connected components, minimum spanning forest
- Progress/termination: using priority updates to guarantee a “winner” in each iteration
- Case study: Breadth-first search using priority updates, using plain writes (write-BFS), write-once, sequential (serial-BFS)