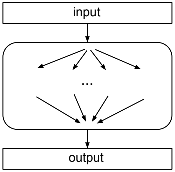
Parallel Combining: Benefits of Explicit Synchronization

Vitaly Aksenov, ITMO University
Petr Kuznetsov, Telecom ParisTech
Anatoly Shalyto, ITMO University

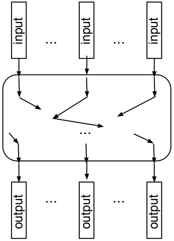
OPODIS 2018

Parallel Programs and Concurrent Data Structures

Parallel programs



Concurrent data structures



Batched Data Structures

- Given a "batch" and a state produces a new state and a vector of responses.
- Parallel batched data structures
  - Static multithreading: PRAM, Bulk synchronous [Val90], asynchronous PRAM [Gib89], ...
  - Dynamic multi-threading: spawn, sync, parallel-for, work-stealing
- Can we use the benefits of parallel batched data structures?

Combining [Oyama et al., 1999], [Hendler et al., 2010]

- Put request into publication list;
- Then, compete for a lock: if won — becomes a combiner, otherwise, becomes a client;
- The combiner applies requests sequentially.

- Hierarchical Flat-Combining [Hendler et al., 2010]
  - Two levels of combining.

Parallel Combining

- Put request into publication list;
- Then, compete for a lock: if won — becomes a combiner, otherwise, becomes a client;
- The combiner and clients apply requests in parallel using a parallel batched data structure.

Parallel Combining

```
execute(method, input):
  req ← new Request()
  request.method ← method
  request.input ← input
  req.status ← INITIAL
  if ℂ.addRequest(req):
    A ← ℂ.getRequests()
    COMBINER_CODE
    ℂ.release()
  else:
    while req.status = INITIAL:
      nop
    CLIENT_CODE
  return
```

Read-Optimized Data Structures

- Operations of two types:
  - Read-only may proceed in parallel;
  - Updates not always
- Combiner collects requests.
  - Read-only are performed in parallel on clients;
  - Updates are performed sequentially by the combiner.

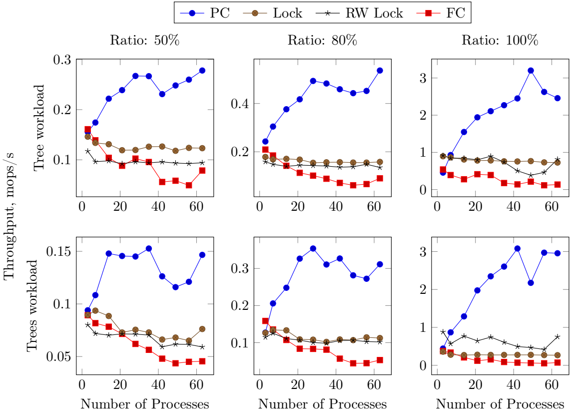
Read-Optimized Data Structures

- The resulting concurrent data structures are linearizable.

Read-Optimized Data Structures. Example

- Dynamic graph [Holm et al., 2001]:
  - Read-only: `areConnected(u, v)`
  - Update: `addEdge(u, v)`, `removeEdge(u, v)`

Dynamic graph. Experiments



Priority queue

- Ordered set of values;
- `Insert(v)`;
- `ExtractMin()`.

- Challenge: find a parallel batched algorithm with complexity that depends only on the batch size and the size of the heap.

Binary heap

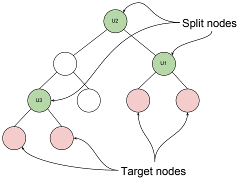
- Binary heap stored in array `a[1,..., s]`: node i has children $2i$ and $2i+1$ with higher values.
- Algorithm [Gonnet and Munro, 1986]
  - `ExtractMin()`: swap `a[1]` and `a[s]`, then sift-down;
  - `Insert(v)`: traverse the path from the root to `a[s + 1]`.

Parallel Batched Binary Heap. ExtractMin

- Combiner with E `extractMin` requests:
  - Locate E nodes with the smallest values using Dijkstra-like algorithm;
  - Swap the values with E latest values `a[s - E + 1]`, ..., `a[s]`;
  - Initiate parallel sift-down on clients from the located nodes;
  - Done using hand-over-hand locking.

Parallel Batched Binary Heap. Insert

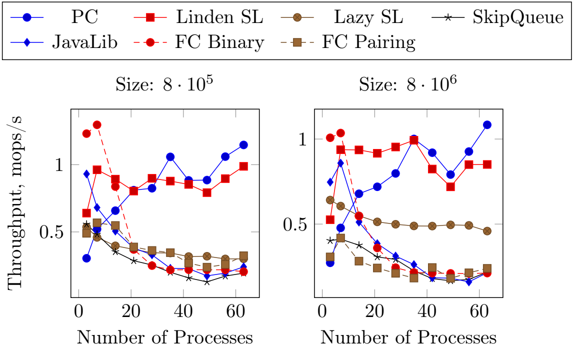
- Combiner with I `insert` requests:
  - Target nodes: `a[s + 1], ...,` `a[s + |I|]`;
  - Locate `|I| - 1` split nodes;
  - Sort $v_{s+1}, \ldots, v_{s+|I|}$ values to insert;
  - Initiate a traversal from the root to target nodes, splitting set of values to insert into two sets in split nodes.



Parallel Batched Binary Heap

- The resulting concurrent binary heap is linearizable.
- Combiner and clients perform $O(c + \log s)$ RMRs in CC and DSM models each and $O(c \cdot (\log c + \log s))$ RMRs in CC and DSM models in total.

Priority Queue. Experiments



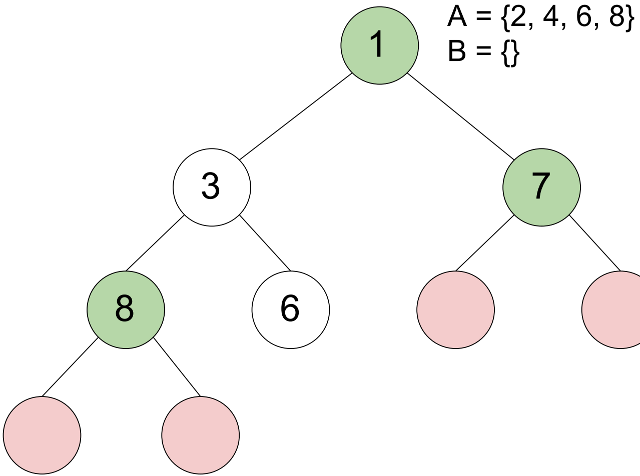
Conclusion

- It is possible to build efficient concurrent data structures from their parallel batched counterpart.
- We affirm it by considering two data structures: dynamic graph and priority queue.
- Which other data structures that can benefit from parallel combining?
    - For example, dynamic tree.

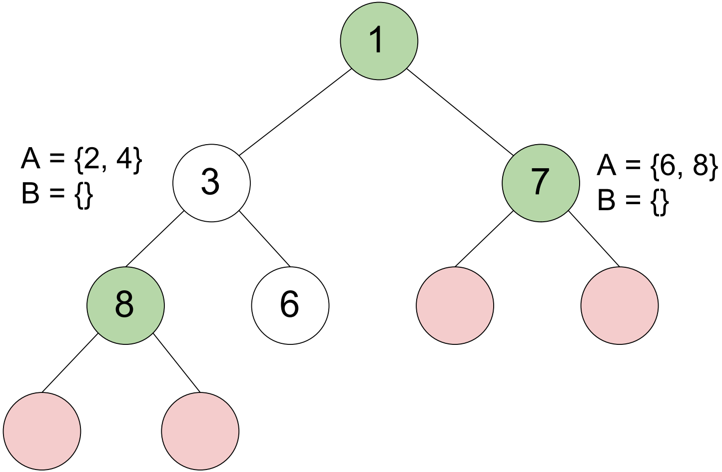
Thank you for your attention

Questions?

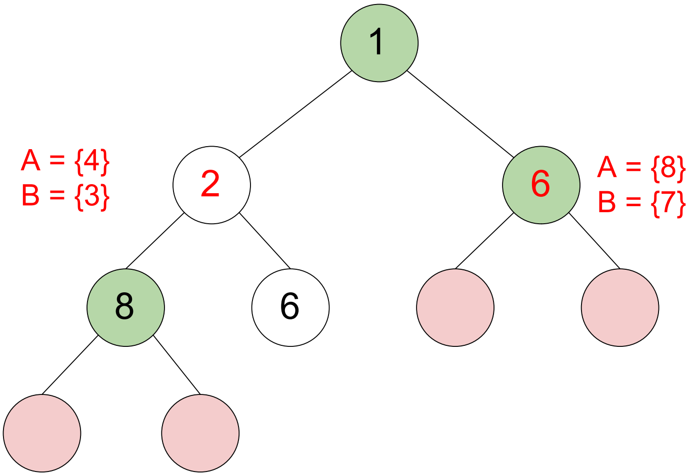
Parallel Batched Binary Heap. Insert



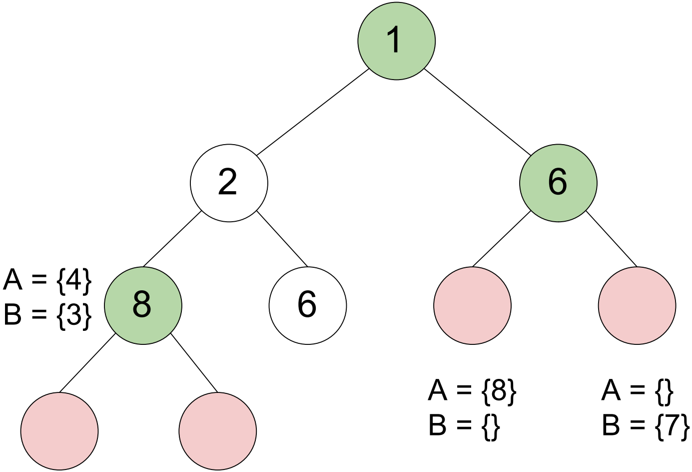
Parallel Batched Binary Heap. Insert



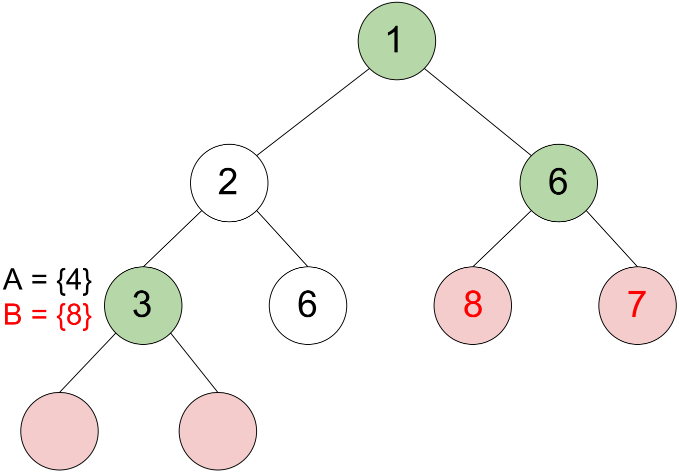
Parallel Batched Binary Heap. Insert



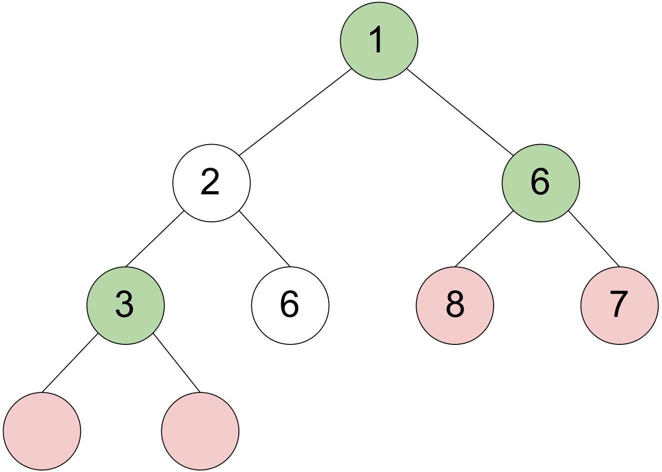
Parallel Batched Binary Heap. Insert



Parallel Batched Binary Heap. Insert



Parallel Batched Binary Heap. Insert



A = {4}
B = {}

A = {}
B = {8}

Parallel Batched Binary Heap. Insert