Contention Adapting Search Trees

Kostis Sagonas
Kjell Winblad

Department of Information Technology
Uppsala University, Sweden

5 Sept 2014

What will be presented?

- Contention Adapting Search Trees (CA trees)
  - Concurrent Data Structure
  - Ordered Sets, Maps, Key-Value Stores
  - Operations: Insert, Remove, Lookup etc
  - In-memory databases
  - Adapts to contention level

What will be presented?

- Contention Adapting Search Trees (CA trees)
  - Concurrent Data Structure
  - Ordered Sets, Maps, Key-Value Stores
  - Operations: Insert, Remove, Lookup etc
  - In-memory databases
  - Adapts to contention level

Why you should care

- Multicores are now everywhere
- Difficult to predict how a data structure will be

Existing Concurrent Data Structures for Ordered Sets

UPPSALA UNIVERSITET

Fine Grained Locking

- Example:
  - A practical concurrent binary search tree, PPoPP'10 N. G. Bronson *et al.*
  - etc

Lock Free Synchronization

- Example:
  - A General Technique for Non-blocking Trees, PPoPP'14 Brown *et al.*
  - etc.

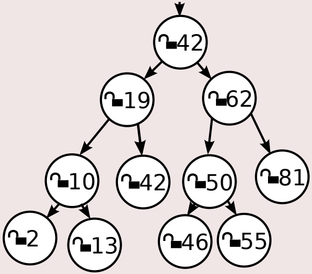
How is CA tree different?

- Adapts according to contention level

UPPSALA UNIVERSITET

Why Adapt to the Contention Level?

Fine Grained Synchronization

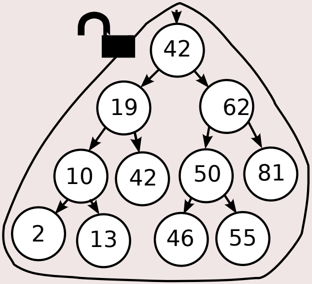


- + Good scalability
- - Memory overhead
- - Performance overhead

UPPSALA UNIVERSITET

Why Adapt to the Contention Level?

Course Grained Synchronization



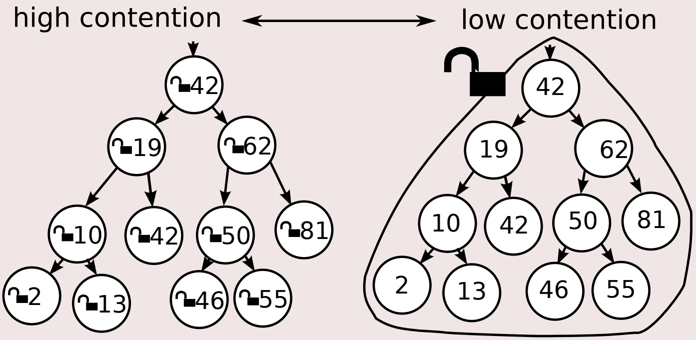
- \- Bad scalability
- \+ Low memory overhead
- \+ Good sequential performance

UPPSALA UNIVERSITET

- **Motivation**
- CA Tree
- Optimizations
- Evaluation
- Future Work
- Conclusion

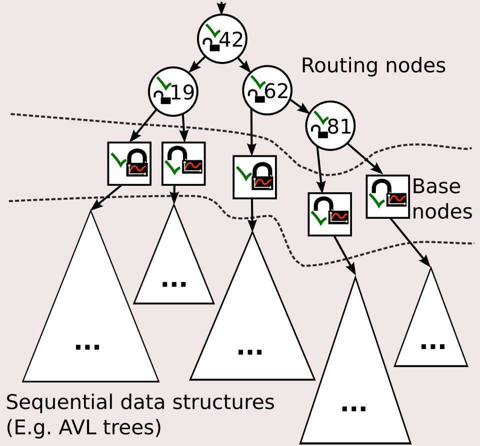
Why Adapt to the Contention Level?

CA Trees adapts between the two



UPPSALA UNIVERSITET

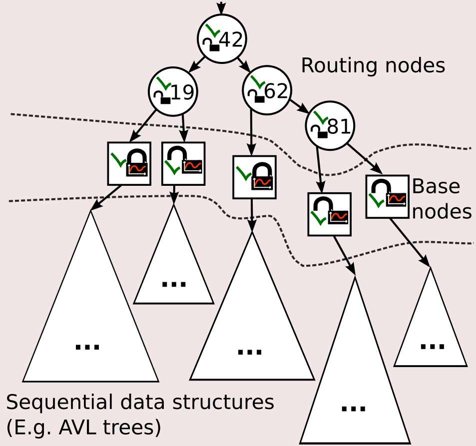
CA Tree Structure



Contention Statistics Collecting Lock

```
void statLock(StatLock slock) {
  if (statTryLock(slock)) {
    slock.statistics -= SUCC_CONTRIB;
    return;
  }
  lock(slock.lock);
  slock.statistics += FAIL_CONTRIB;
}
```

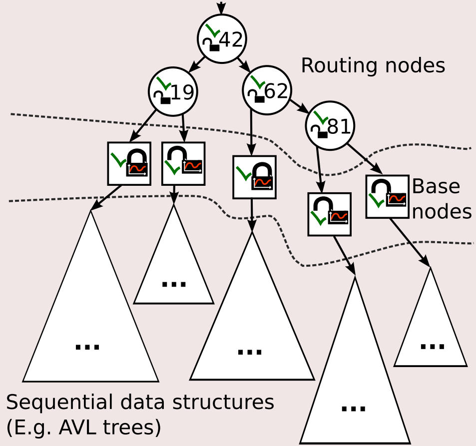
CA Tree Structure



Contention Adaptation

```
if (base.lock.statistics > MAX_CONTENTION) {
  if (size(base.root) < 2) base.lock.statistics = 0;
  else highContentionSplit(tree, base, prevNode);
} else if (base.lock.statistics < MIN_CONTENTION) {
  if (prevNode == null) base.lock.statistics = 0;
  else lowContentionJoin(tree, base, prevNode);
}
```

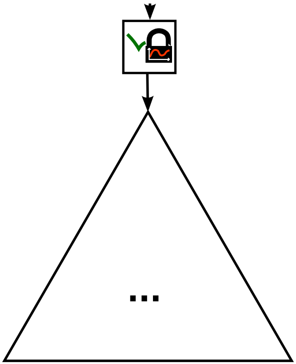
CA Tree Structure



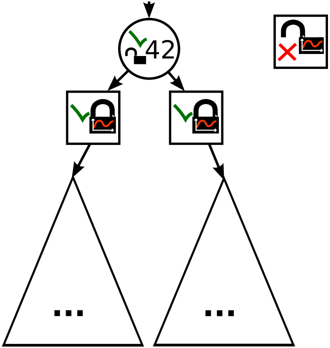
Sequential Ordered Set Data Structures

- Requires support for split and join
  - The **split** operation splits a tree into two so that all keys in one tree are smaller than the keys in the other
  - The **join** operation merges two trees given that all keys in one tree are smaller than the keys in the other
  - $\mathcal{O}(log(N))$ implementations for **AVL trees**, **Red-Black trees**, **Treaps**, **Skip lists** etc.

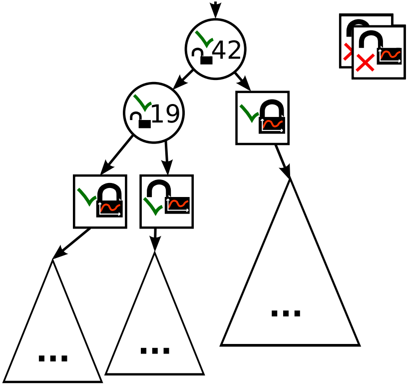
CA Tree Animation



CA Tree Animation

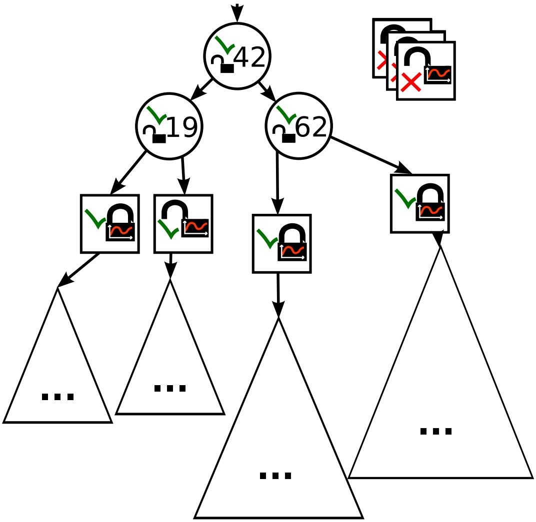


CA Tree Animation

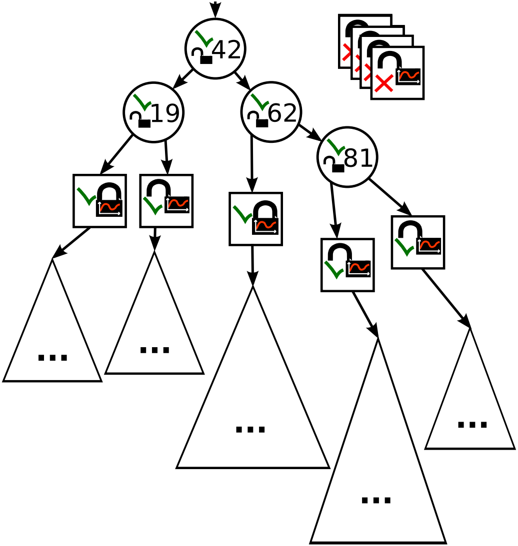


UPPSALA UNIVERSITET

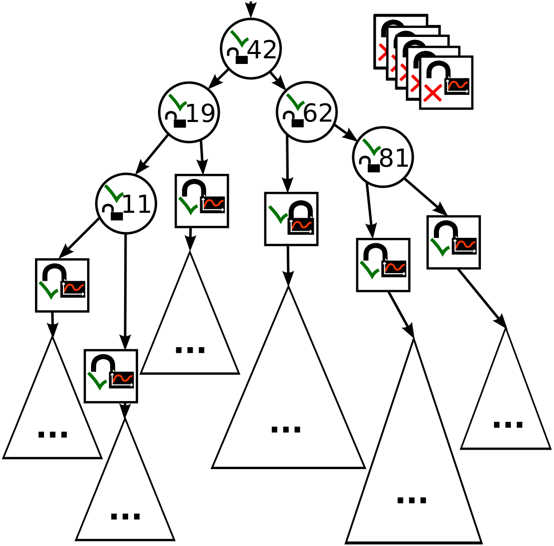
CA Tree Animation



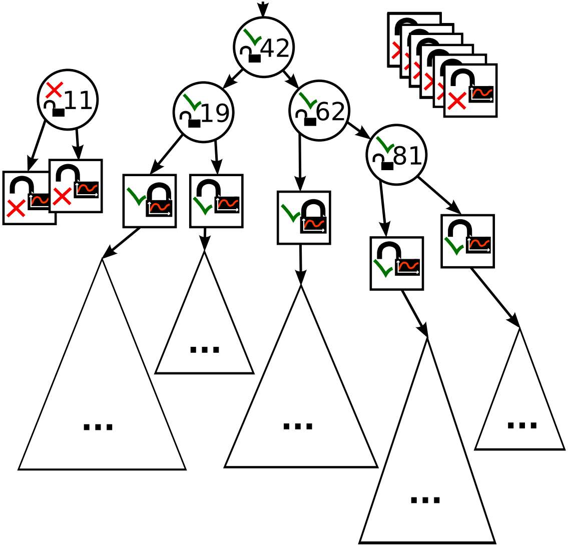
CA Tree Animation



CA Tree Animation

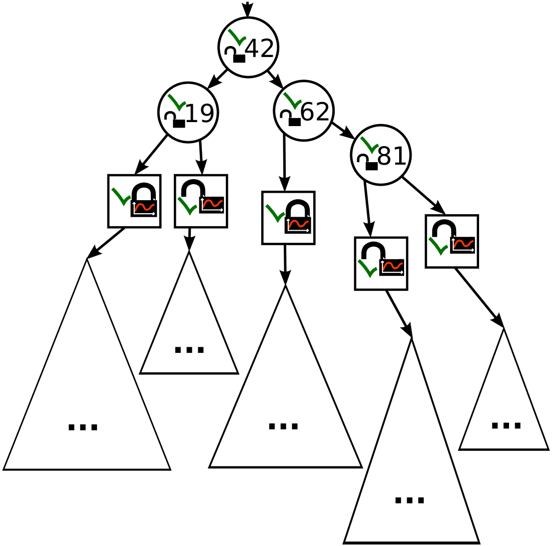


CA Tree Animation



UPPSALA UNIVERSITET

CA Tree Animation



Properties

- Deadlock freedom
- Livelock freedom
- Linearizable

Optimisations for Read-Only Operations

Sequence lock

- Uses a counter (Sequence number)
  - Initially zero
- Lock
  - Increment to uneven (Compare-and-Swap)
- Unlock
  - Increment to even again
- Optimistic reads
  - Check sequence number before and after CS

Transformation to “Lock-free” base nodes

- Optimization for contended base nodes with one or less elements
- Without optimization
  - Lock
  - Store
  - Unlock
- With optimization:
  - Single compare and swap

Evaluation

- $X/2\%$ Insert
- $X/2\%$ Remove
- $100 - X\%$ Lookup
- NUMA with four:
  Intel(R) Xeon(R) CPU E5-4650 CPUs (2.70GHz)
  eight cores each
  $+$hyperthreading
  $=$ 64 hardware threads
- Java

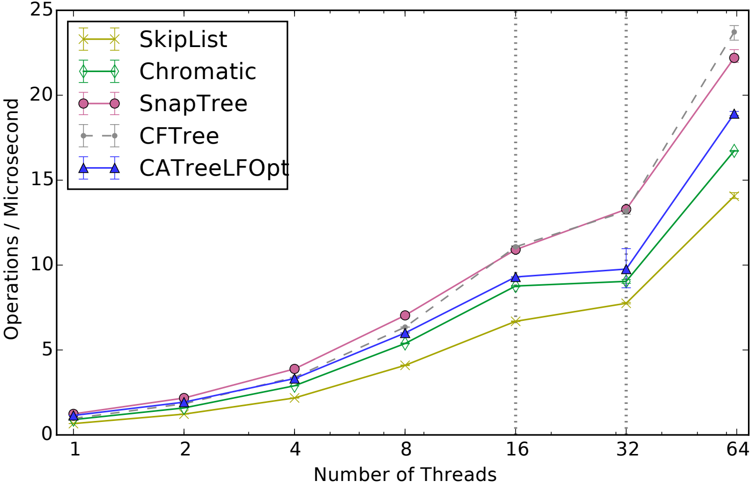
Results Summary Optimizations

- Sequence lock
  - Improved performance in read heavy scenarios
- Transformation to "lock-free" base nodes
  - Improved performance when contention is high

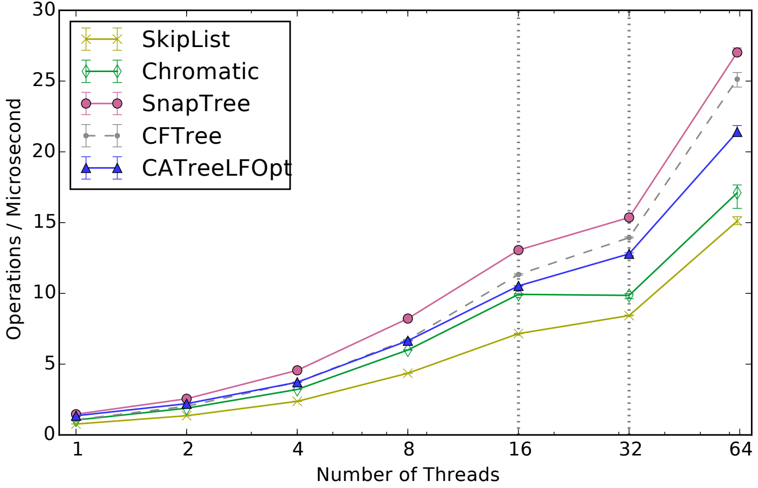
Evaluation of CA trees compared to other data structures

- Chromatic – lock free relaxed AVL tree
  - PPoPP'14 T. Brown, F. Ellen, and E. Ruppert
- SkipList – Lock-free skip list
  - From Java Fundation Classes (Doug Lea)
- SnapTree – Fine grained locking and optimistic reads
  - PPoPP'10 N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun
- CFTree – Balancing roations made by separeate thread
  - Euro-Par 2013, T. Crain, V. Gramoli, and M. Raynal.

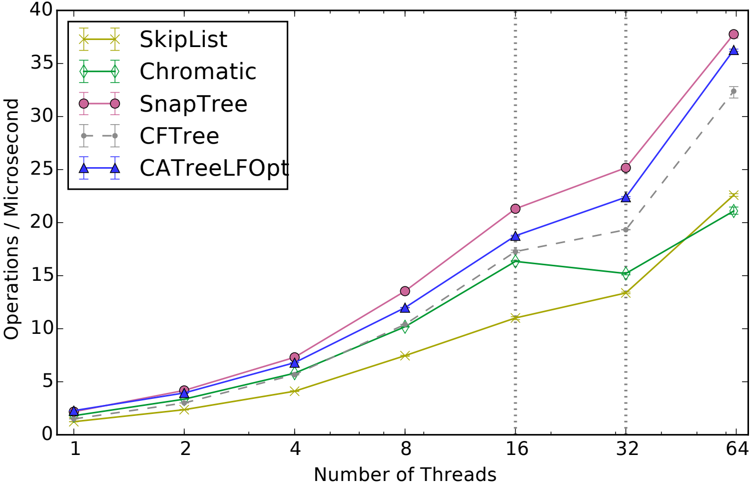
Size 1000000, Update only (Remove and Insert)



Size 1000000, 50% Update, 50% Lookup)



Size 1000000, 1% Update, 99% Lookup)



UPPSALA UNIVERSITET

Motivation

CA Tree

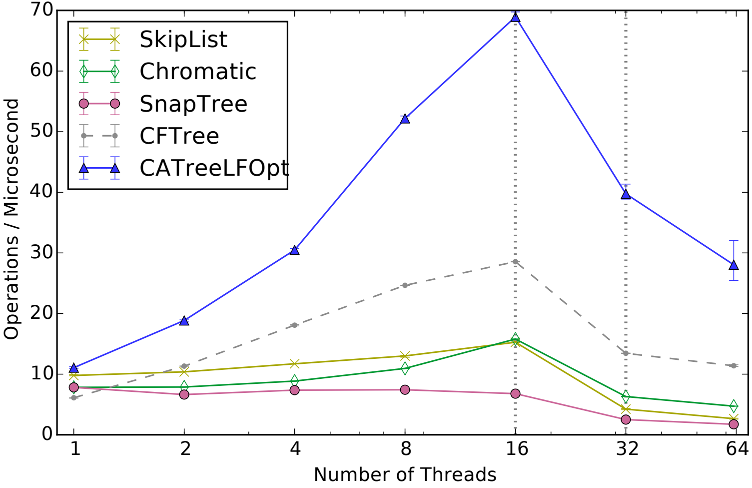
Optimizations

Evaluation

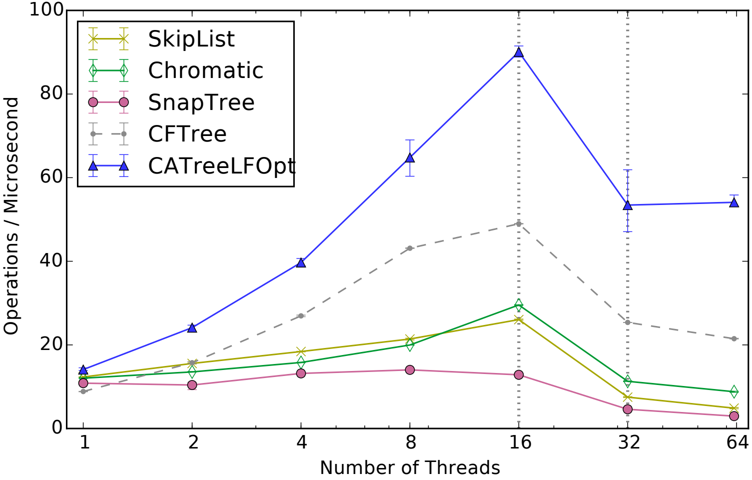
Future Work

Conclusion

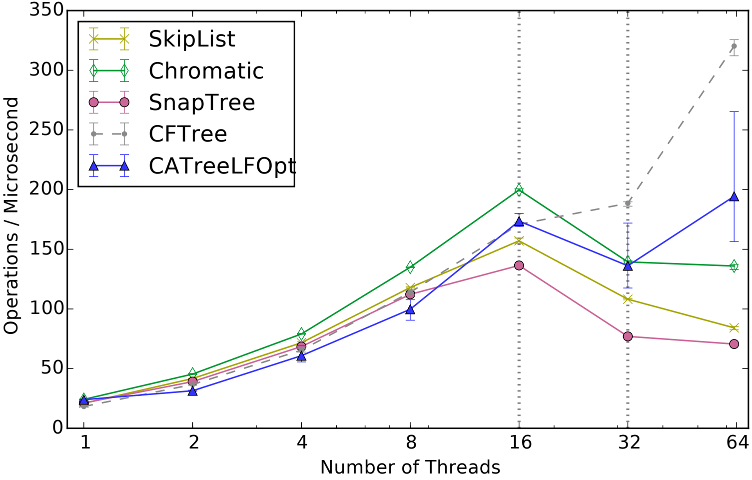
Size 10, Update only (Remove and Insert)



Size 10, 50% Update, 50% Lookup)



Size 10, 1% Update, 99% Lookup)



Summary: Comparison to other data structures

- Large Set Sizes
  - Similar to state-of-the-art
- Small Set Sizes
  - Shows the power of adapting to the contention
- Sequential Performance (Not in Graphs)
  - Overall the best

Other optimizations

- Hardware Lock Elision
  - Uses Hardware Transactional Memory
- RW-locks in base nodes

Future Work

- range queries, bulk operations etc
- Use in-memory data base
- Change sequential data structure dynamically depending on type of operations

UPPSALA UNIVERSITET

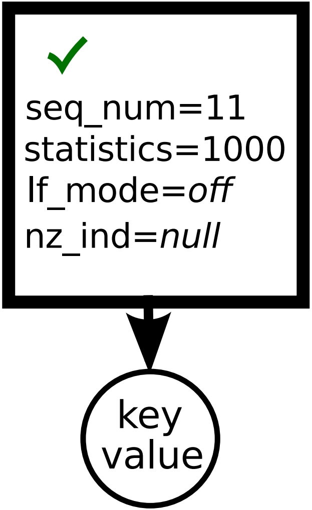
Conclusion

- Adapting to the contention level
  - Can give:
    - Good scalability
    - Good sequential performance
- Interesting properties:
  - Different structure in different parts depending on the contention distribution
  - Flexibility

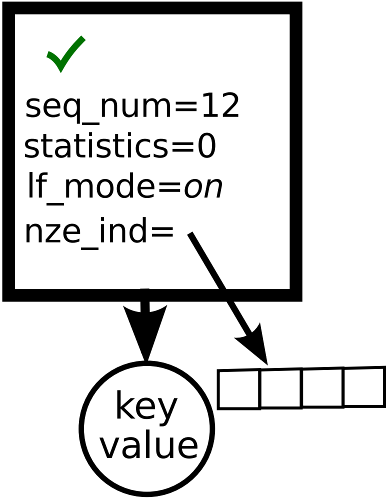
Thank you

- Code online:
  http://www.it.uu.se/research/group/languages/software/ca_tree

Transformation of base nodes containing few elements



Transformation of base nodes containing few elements

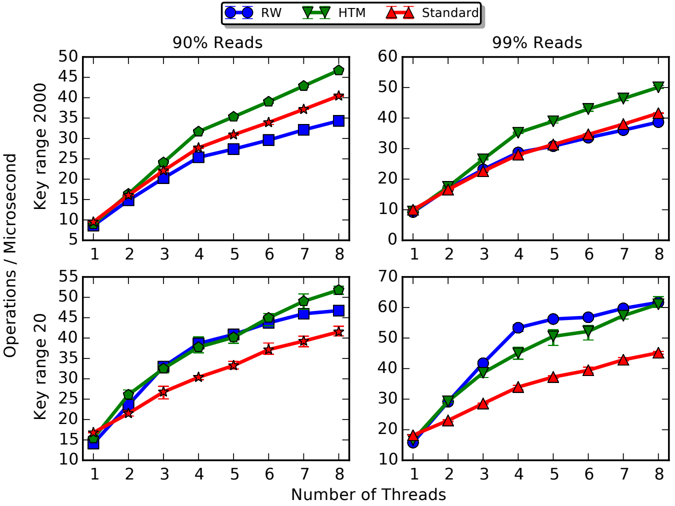


UPPSALA UNIVERSITET

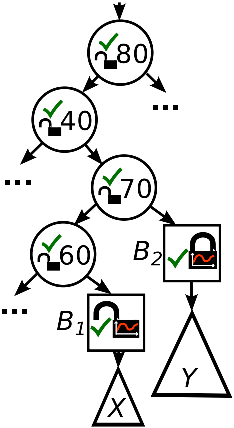
Other optimizations

- Evaluated on C benchmark
- Intel(R) Xeon(R) CPU E3-1230 v3 (3.30GHz)
  4 cores with hyperthreading
  8 hardware threads

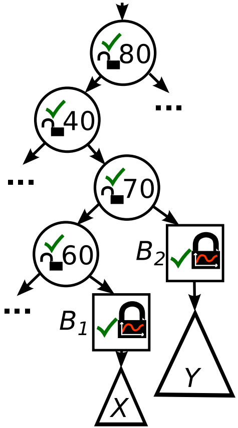
Evaluation Other Optimizations



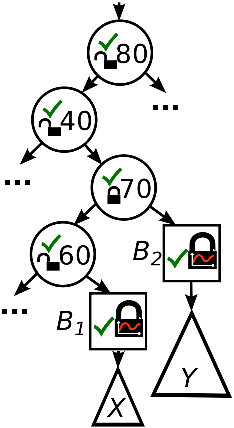
Low-contention join



Low-contention join

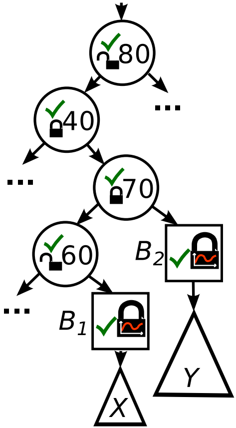


Low-contention join



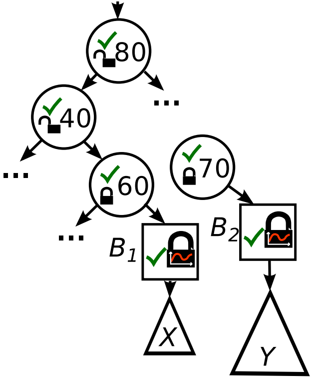
UPPSALA UNIVERSITET

Low-contention join



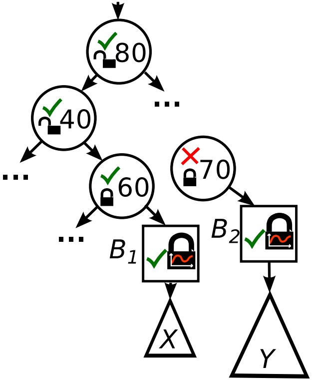
UPPSALA UNIVERSITET

Low-contention join



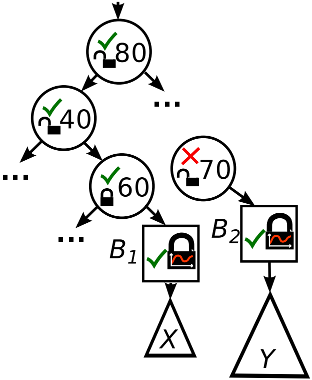
UPPSALA UNIVERSITET

Low-contention join

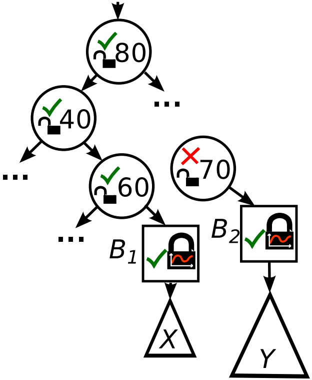


UPPSALA UNIVERSITET

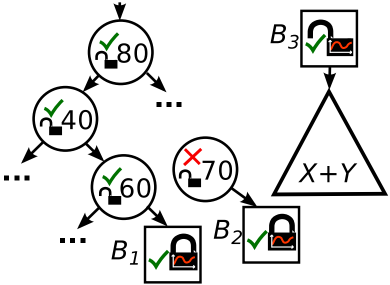
Low-contention join



Low-contention join

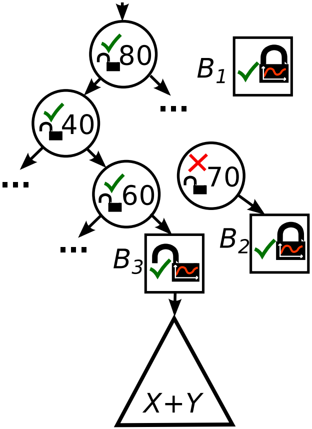


Low-contention join

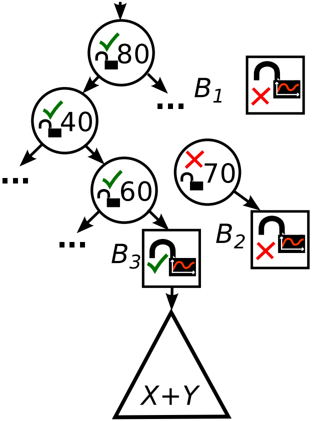


UPPSALA UNIVERSITET

Low-contention join



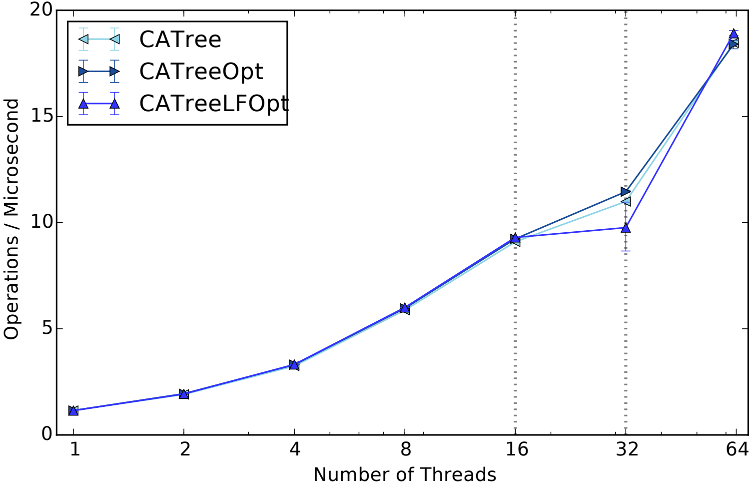
Low-contention join



Evaluation

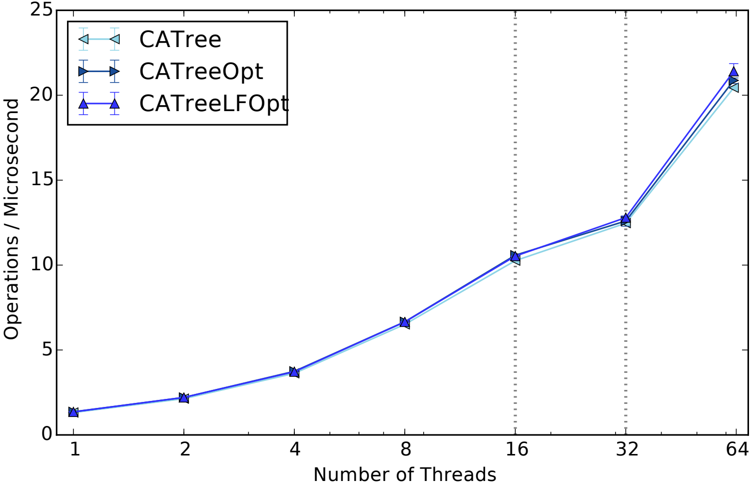
- $X/2\%$ Insert
- $X/2\%$ Remove
- $100-X\%$ Lookup
- NUMA with four:
  Intel(R) Xeon(R) CPU E5-4650 CPUs (2.70GHz)
  eight cores each
  $+$hyperthreading
  $=$ 64 hardware threads
- Java

Size 1000000, Update only (Remove and Insert)

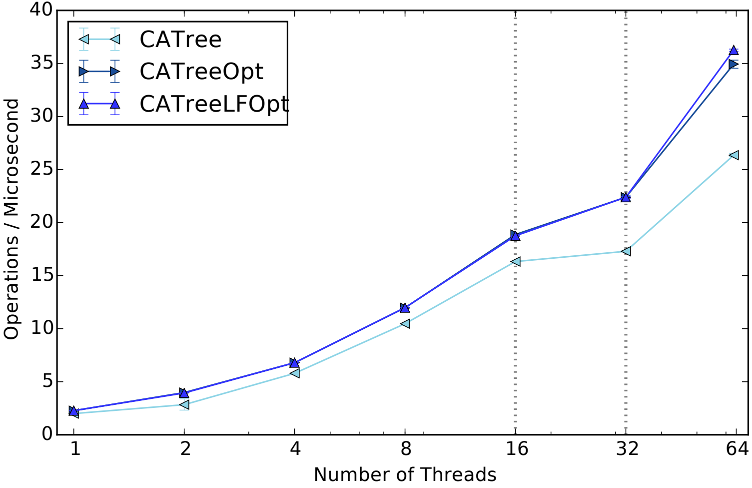


UPPSALA UNIVERSITET

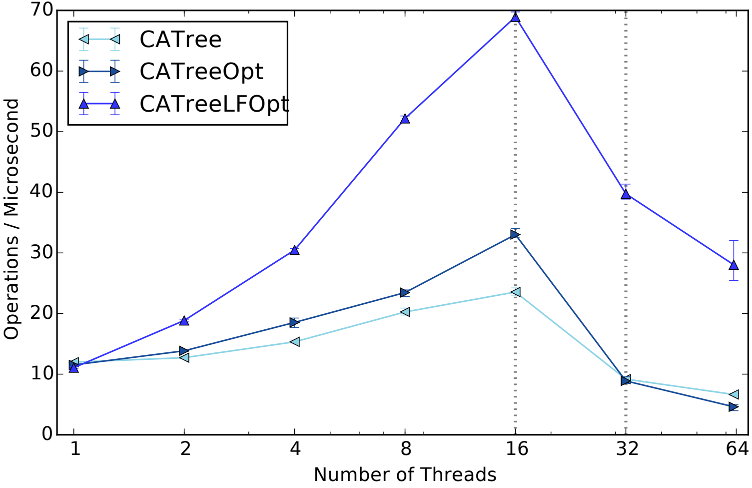
Size 1000000, 50% Update, 50% Lookup



Size 1000000, 1% Update, 99% Lookup

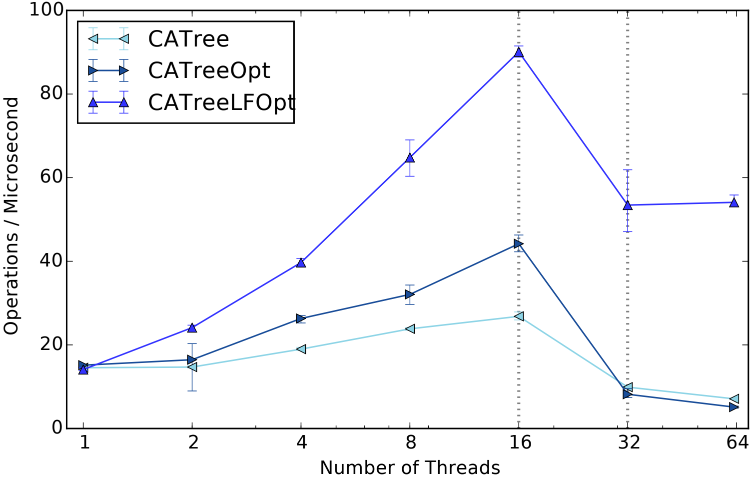


Size 10, Update only (Remove and Insert)



UPPSALA UNIVERSITET

Size 10, 50% Update, 50% Lookup



UPPSALA UNIVERSITET

Size 10, 1% Update, 99% Lookup