Cache-Conscious Concurrent Hash Array Mapped Trie

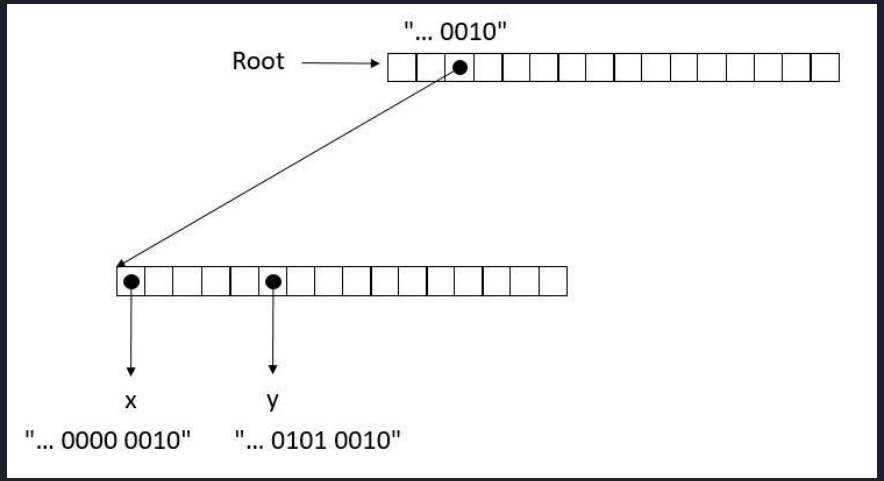
Chi-Chun Chen
Shaojie Wang
Princeton Ferro

Topics

- Hash Array-Mapped Tries
- Why Rust?
- Cache Consciousness
- Lock-free HAMT

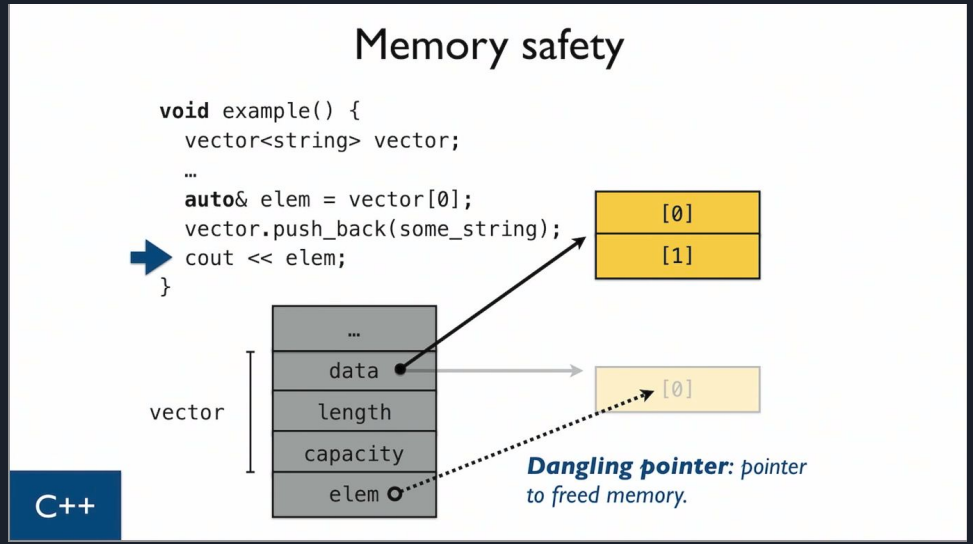
Hash Array-Mapped Tries

1. A HAMT is a data structure where the data is stored at different levels.
2. A 64-bit hash of the key is divided into groups of 4 or 2 bits, and each group indexes into the next level.



Hash Array-Mapped Tries

- Space efficient
- Persistent data structure. Get rid of dangling pointer issues.
- Small O(1) costs for principle operations (insertion, search, removal), and guaranteed small bounded worst case times. (Bagwell, 2001, Ideal Hash Trees.)



Why Rust?

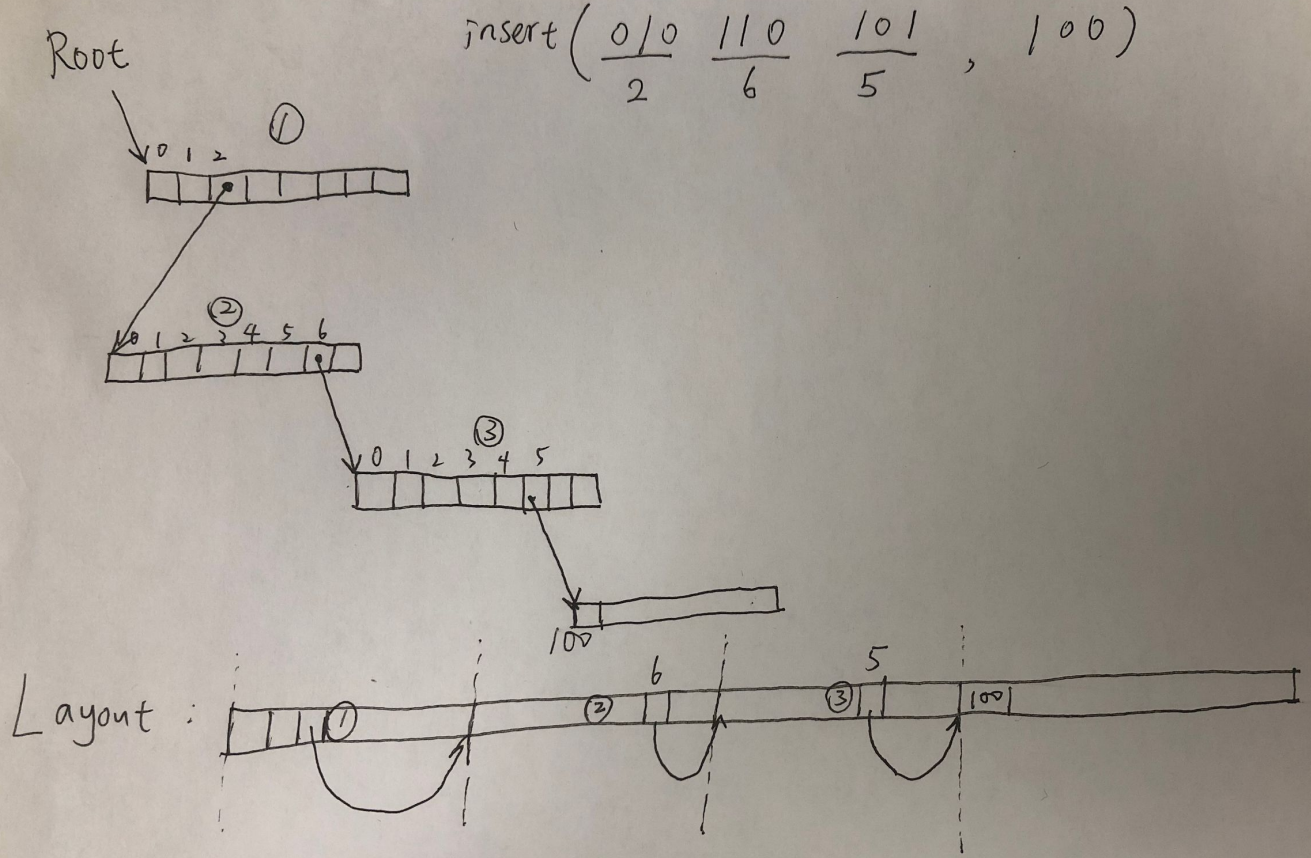
- Same niche as C/C++
  - No garbage collector
- Backed by llvm
- Static + Strong type
- Smart pointer by default
- Safe

Why Rust?

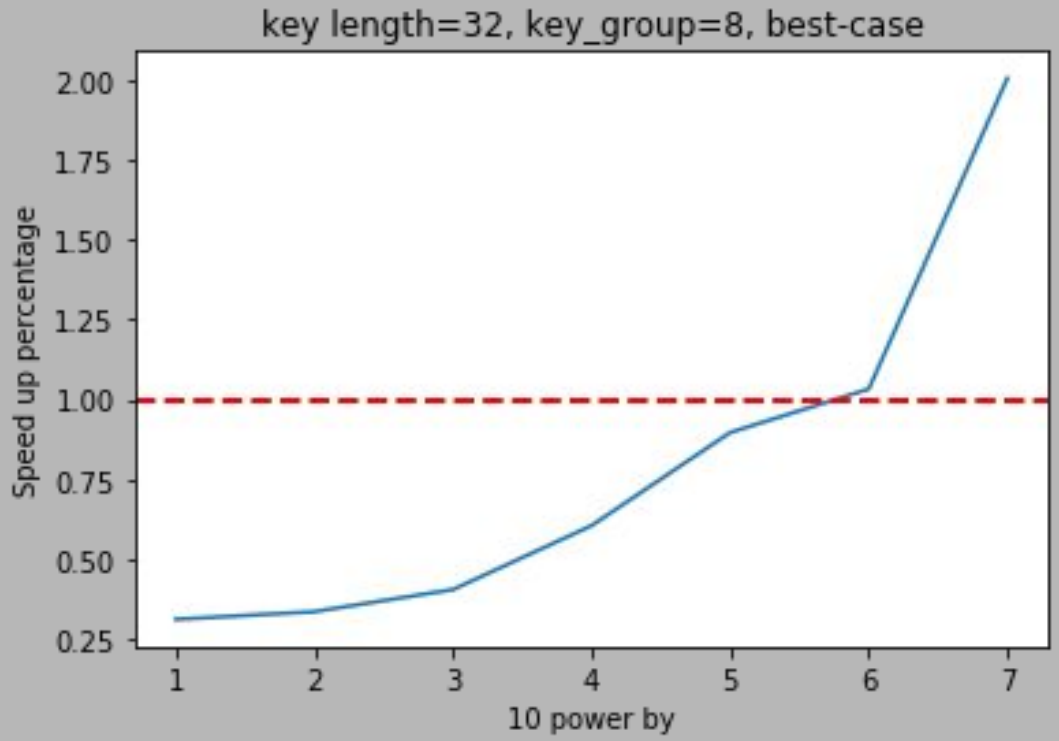
- No concurrent Data Structure in standard library
- Few cache conscious implementation

Cache-Consciousness

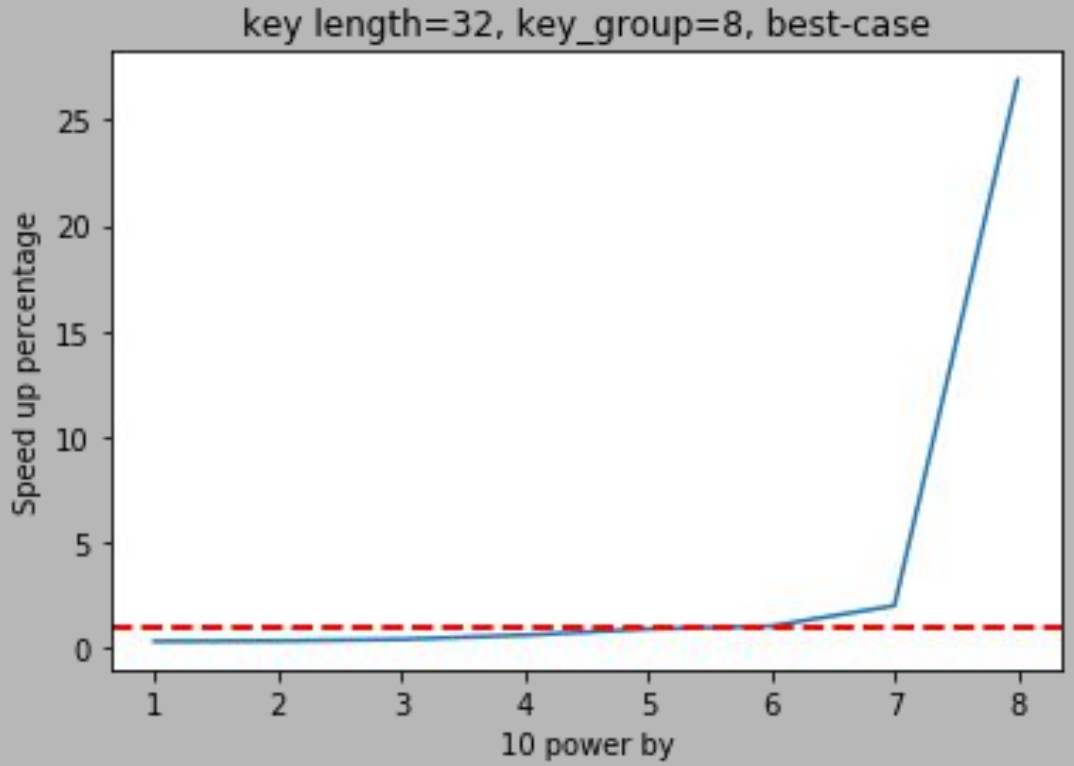
- Design a memory layout
  - Static Memory layout
    - Clustering


Root
insert(010/2 110/6 101/5 , 100)
①
0 1 2
②
0 1 2 3 4 5 6
③
0 1 2 3 4 5
100
Layout :
①
②
6
③
5
100

Cache-conscious trie w/o concurrency



Cache-conscious trie w/o concurrency



Cache-Consciousness

- Design a memory layout
  - Static Memory layout
    - Clustering
- Dynamic Memory layout
  - Packing the data in the order of "First Appearance"
  - Group packing
  - Reference: Chen Ding, Ken Kennedy: Improving Cache Performance in Dynamic Application through Data and Computation Reorganization at Run time

Cache-Consciousness

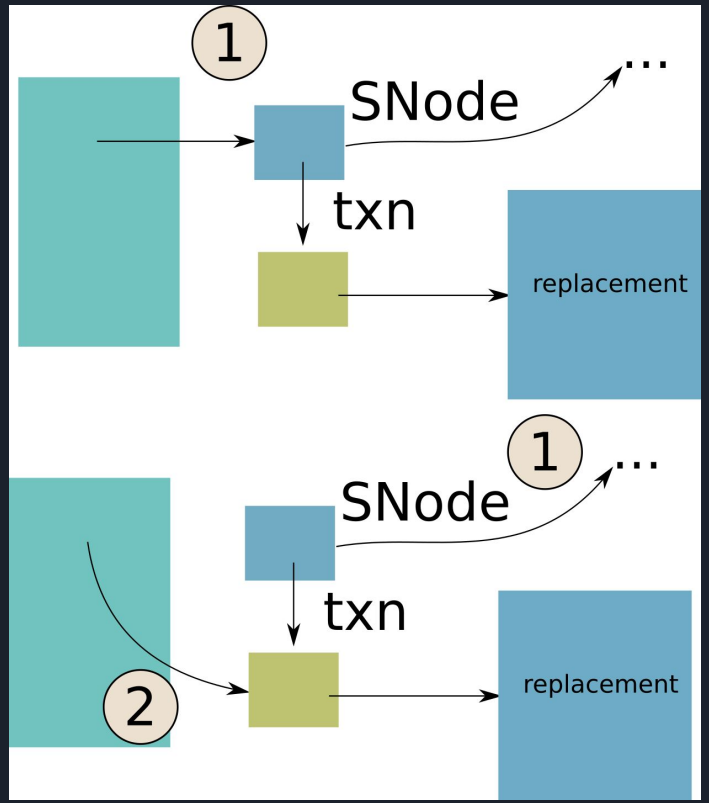
- Design a memory layout
  - Static Memory layout
    - Clustering
- Dynamic Memory layout
  - Packing the data in the order of "First Appearance"
  - Group packing
  - Reference: Chen Ding, Ken Kennedy: Improving Cache Performance in Dynamic Application through Data and Computation Reorganization at Run time

Lock-free HAMT

Nodes are immutable

Each node has an associated transaction field that indicates there is a concurrent modification request.

When another thread sees this, it helps complete the transaction.



Lock-free HAMT: Cache and Memory Layout

Cache Trie

We also cache the largest level of the trie. After a certain number of misses, periodically, we change this level.

Layout

We implement consecutive packing for Node elements, in the order of inserts.

Results - Lockfree

Tested against Rust's Concurrent HashMap.

Though we are beaten by chashmap in most cases, our performance scales well, particularly when the number of threads exceeds the number of hardware contexts.

Conclusion

In this project, we implemented concurrent cache-conscious hash array mapped trie. We first implement the memory layout for cache-consciousness, which gives the hash trie a speed up of 2500% in best case with 10,000,000 nodes. For the blocking version of hash trie, we can see a good scalability when node number goes larger. For lock-free version, we implemented a concurrent HAMT with a cache-trie for O(1) lookup in the best-case. Our implementations scale well with the number of threads.

Multi-thread programming: