Lock-free Skip List

https://supertaunt.github.io/CMU_15618_project.github.io/

Chen He, Yida Wu

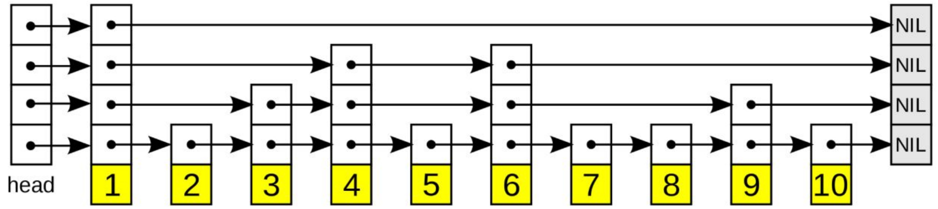
Background & Goal

Goal: Implement coarse-grained, fine-grained and lock-free versions of skip list using C++

Target Machine : GHC machines with 8-cores and shared memory

Interface, inputs and outputs:

```
class Skiplist {
    public:
        virtual bool insert(int e) = 0;
        virtual bool remove(int e) = 0;
        virtual bool contain(int e) = 0;
};
```



Lock-based Skip List

- Coarse-Grained Lock-based Skip List

Implementation: A global mutex lock to protect the whole data structure

- Fine-Grained Lock-based Skip List

Key ideas: locks every connected node for write.

Fully Linked List: The whole list is fully linked.

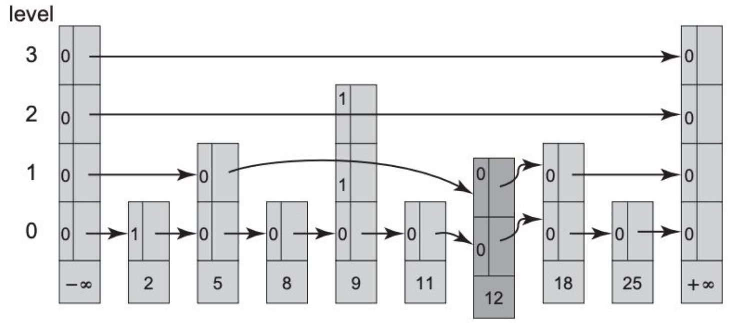
Logical Deletion: Mark the node under deletion.

Non-blocking Read: Read can perform as if in a single thread view, no lock required.

Pros: Search very fast with non-blocking read; Fine performance when contentions are low

Cons: The retry cost is very high in high contention especially in sequential scenarios.

Lock-free Skip List



Key ideas: CAS: compare and swap

Bottom Level List: only elements in bottom level matter.

Logical Deletion: mark pointer to indicate deletion.

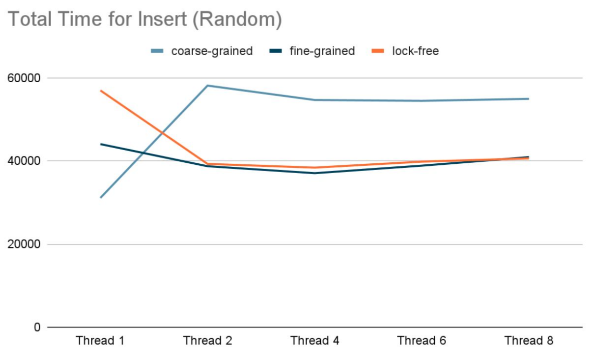
Markable Reference: store pointer and mark together to CAS in one time.

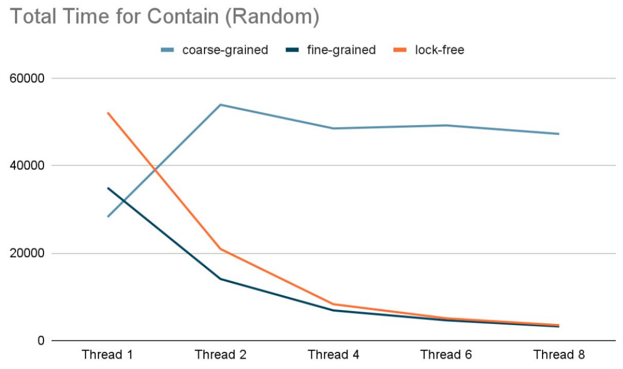
Pros: No need to lock, low overhead when threads number is relative large and collision is low.

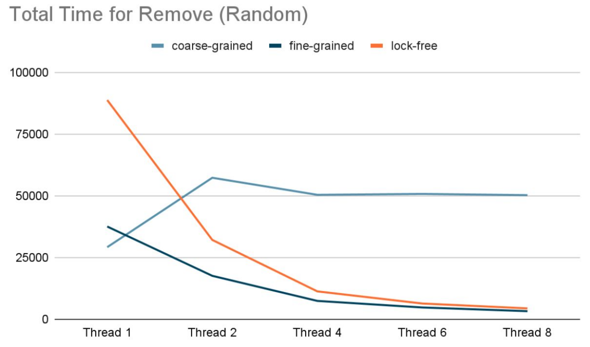
Cons: High overhead during searching due to implementation of “Markable Reference.” More obvious in low thread number. High retry cost for high collision rate. Not obvious enough when threads number reach up to 8.

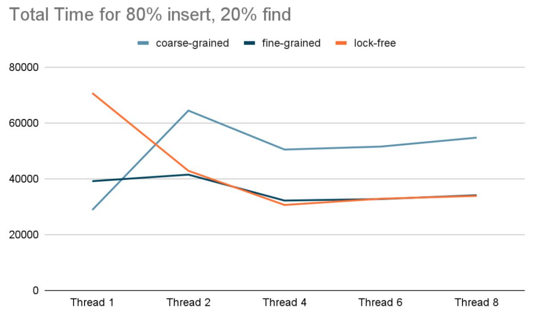
Performance

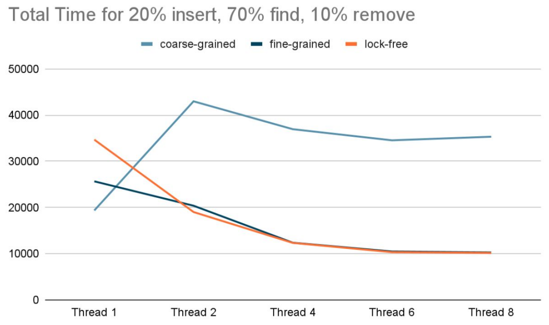
Good performance in contain and remove
Relative low in insert

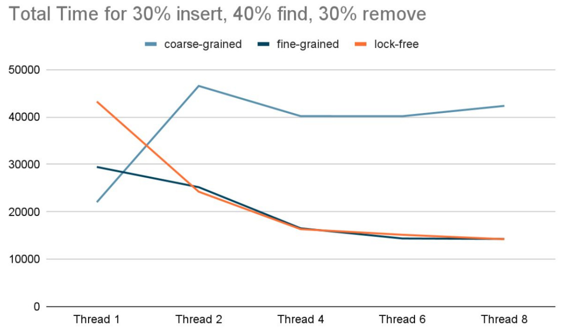






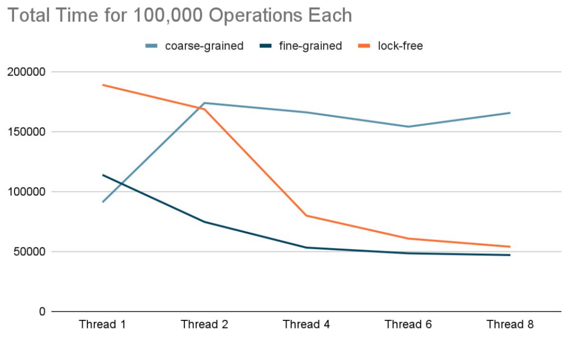


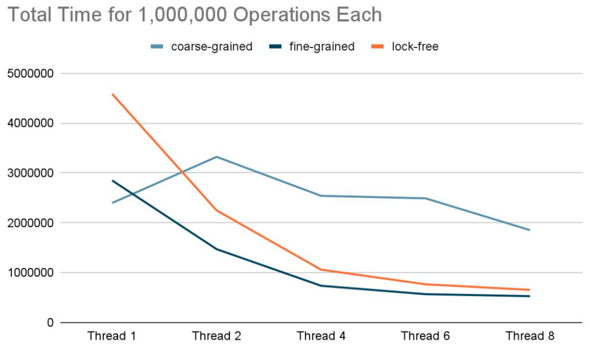


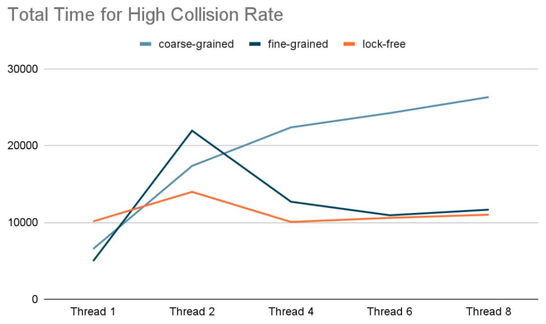


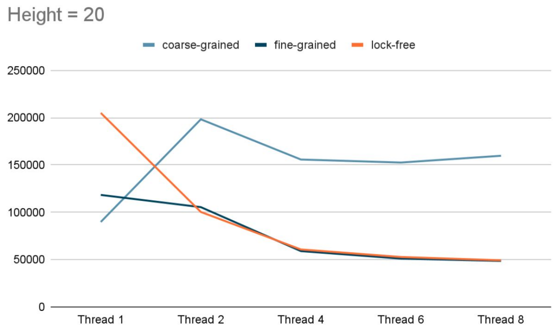
Performance

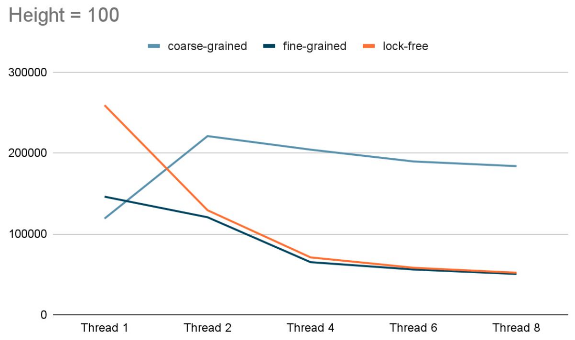
Better speedup when operation number is large
Poor speedup when high collision rate











Little decrease on the performance when max height increase. It takes super long time when height is too small or too large. (not recorded)

Conclusion

Summary:

- Both fine-grained and lock-free achieve good performance on multithreading
- Better performance for contain and delete compared to insert
- Better speedup for large number of operations
- Lock-free have higher overhead in low thread number in current implementation

Future Work

- Find other solutions for Markable Reference in lock-free to reduce overhead
- Perform more tests on other machines with more than 8 cores
- Optimize insert speed (e.g. batch insertion)

A Special Observation on Fine-Grained Skip List

Observation:

- Instead of using atomic pointer and bool variables to record connections and the state of a node, using normal pointer and volatile bool variables still lead to the same correct result.

Thinking:

- The compiler automatically aligns variables well and the underlying architecture automatically executes them in an atomic manner even they are not atomic variables.
- The instruction execution order of these variables does not affect the correctness of the program, since these variables will finally reach the desired states and precision of the non-blocking search is not extremely important.