

Core Ideas and Major Modifications

The core idea of our implementation is Cache Sensitive B+-Tree (CSB+-Tree). We made use of the similar code architecture that we were given. However we implemented almost all methods by ourselves. Main part of our implementation is under the core/csbtrees folder. Instead of leaves and internal nodes, we have leaves in leaf groups and nodes in node groups. But nodes and leaves are not physically there, namely they are only parts of the arrays. They are used to find the position of the key in the child node or leaf with the help of simple arithmetic. Additionally our split reasons are different. Whenever a node or a leaf is full, it will split into two (simply array is expanded), at that time if the parent node is also full, the whole node/leaf group will also split.

Our root is a leaf group at the beginning, then it will become a node group, but with only a single node inside. This constraint is because we want to make binary search cache efficient in the root. Without this (single node) restriction it causes many cache misses to binary search the whole node group. Since we do not have any pointers between the keys in a node, we fit more keys in a node as compared to the b tree. Hence we have more fan-out. This means height of CSB tree is less. Therefore our searches are faster. While creating the tree, we try to keep utilization of the leaves low, that is, splitting a leaf group depends on the fullness of the parent node. So leaf groups may split even if the leaves inside them are not full. This makes insertion faster while executing the workload. We wrote our own test classes for the Array Maps of nodes and leaves. We fully documented everything except them.

Comments and Results

With the given workload (for 10 million operations), the following were the times recorded for B+ tree and our implementation:

Time for B+ tree:	25.789040 sec
Time for our implementation:	19.553026 sec
Percentage improvement:	24.18 % better

Further Possible Optimization: Since the workload has nearly 42% range queries, we could improve it by avoiding sequential scan once the lower bound of range is located. Thereafter we can compare the upper bound of range with the leaf group edges and perform binary search if it lies within. The idea is to return full leaf groups in case they are subsumed by the range query intervals.

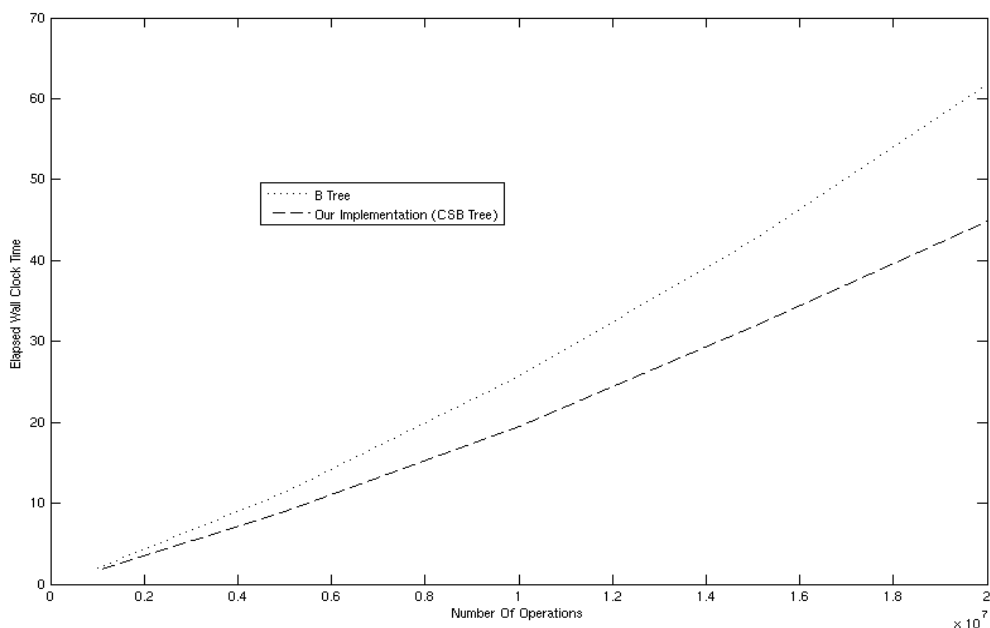


Figure1: B+ tree Vs Our Implementation (CSB+ tree)