Concurrent AVL Tree Operations on GPUs

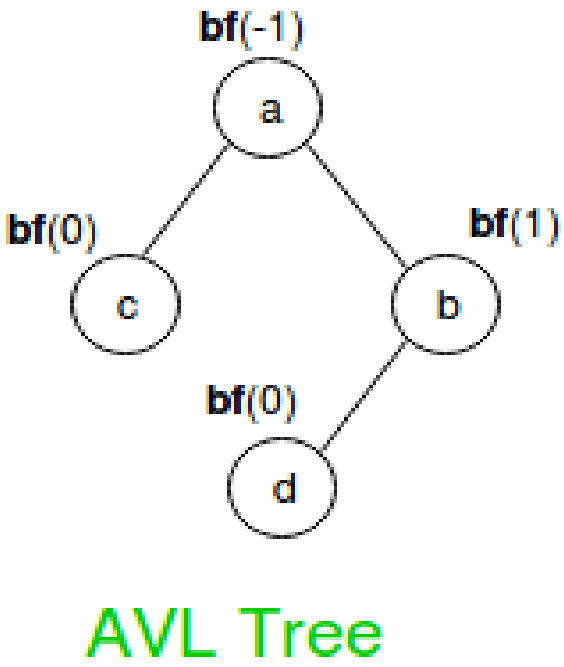
Abhishek Yadav, CS12B032
Dr. Rupesh Nasre

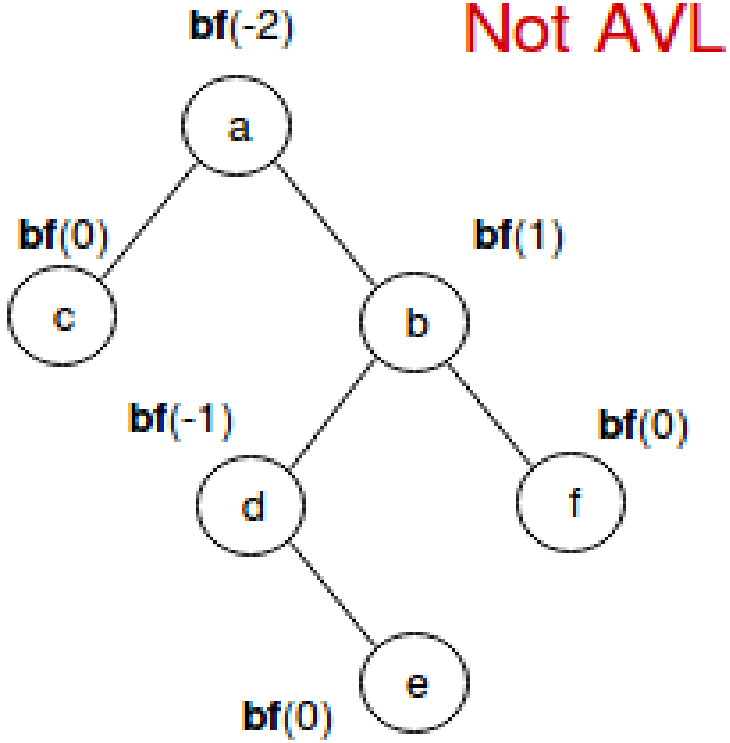
Indian Institute of Technology Madras

25th April 2017

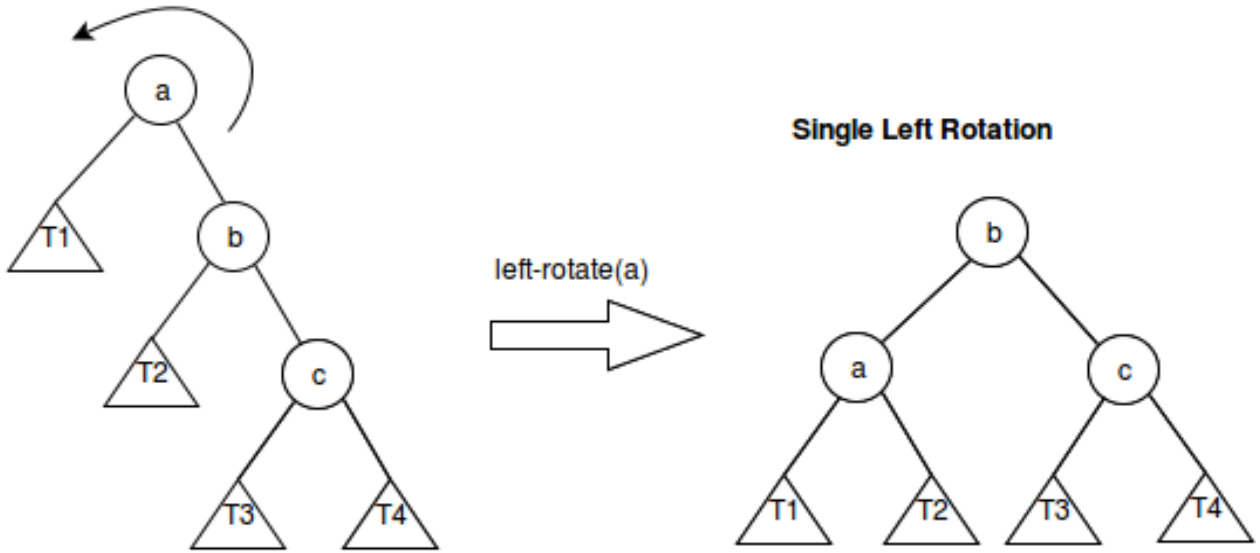
Introduction

- AVL Trees are Height Balanced Binary Search Trees
- Balance factor of all the nodes $\in \{-1,0,1\}$



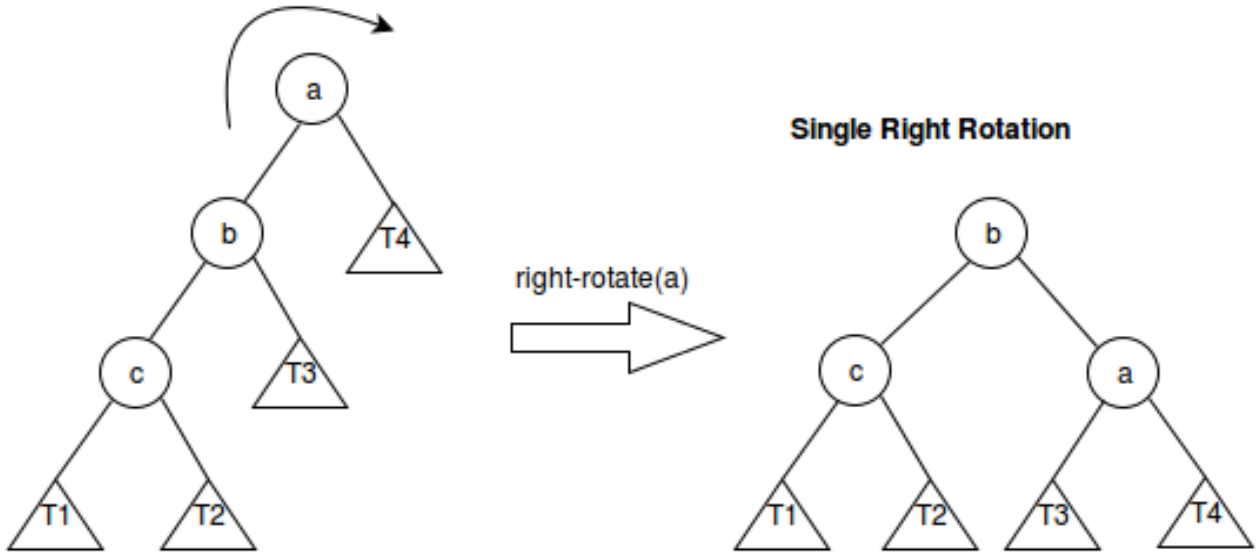


- Provides efficient *lookup* operation
- Insert and delete operations may result in imbalanced nodes
- Rotations are performed to balance the nodes

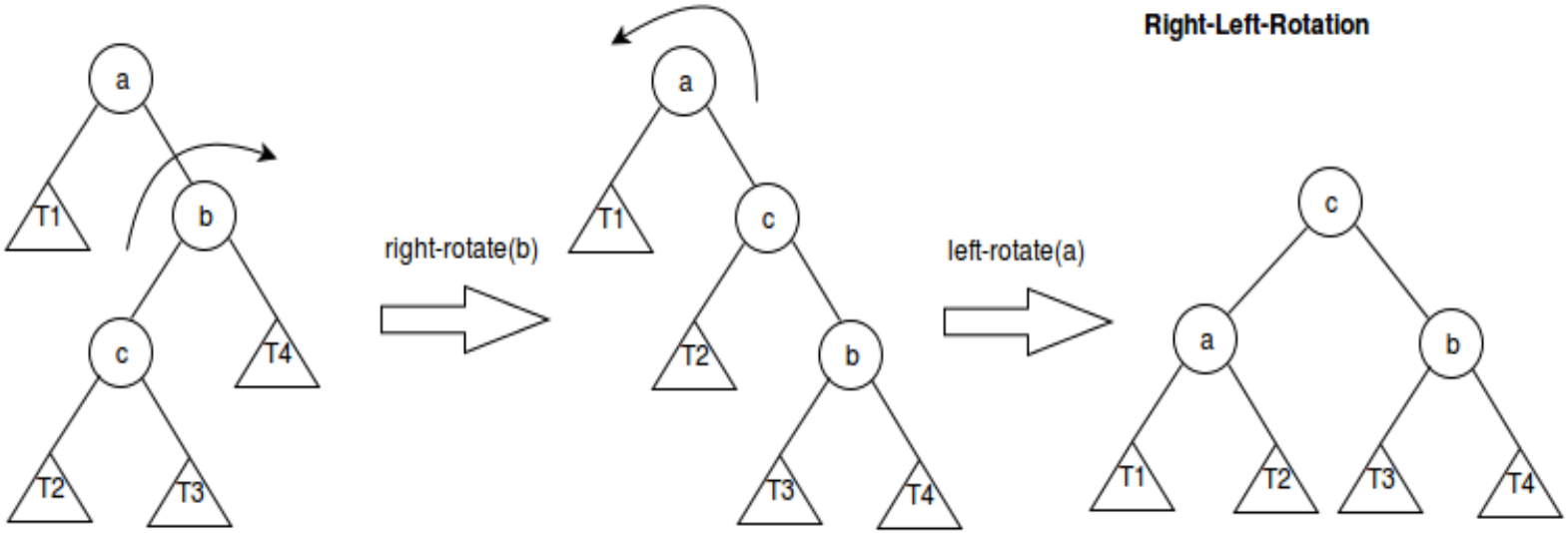


- **Single Left** rotation takes place if **bf** of the critical node(a) $\leq$ -2 and **bf** of its right child is ≤ 0

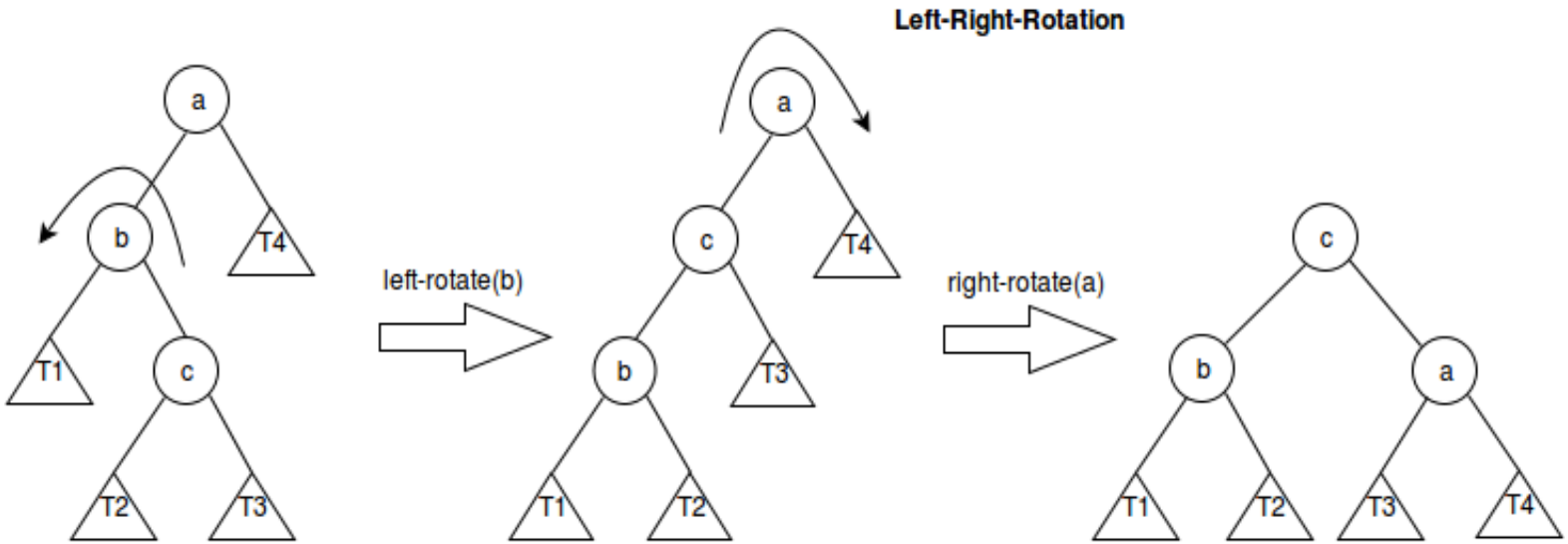
- **Single Right** rotation is performed if **bf** of the unbalanced node is $\geq$ 2 and **bf** of its left child is ≥ 0



- **Right-Left** double rotation takes place if **bf** of the unbalanced node is $\leq$ -2 and **bf** of its right child is $\geq$ 0
- A single right rotation followed by a single left rotation

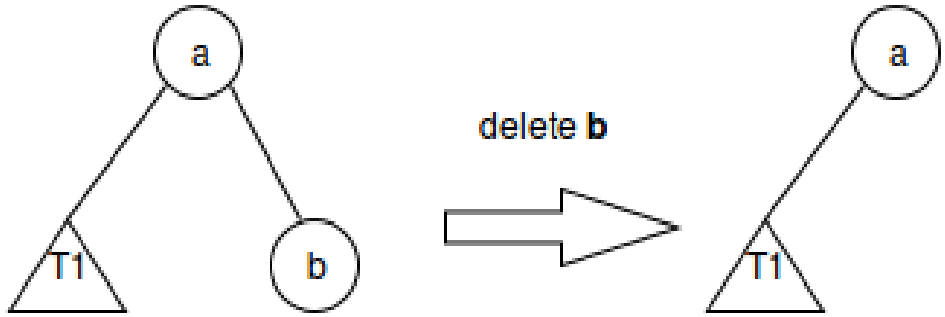


- **Left-Right** double rotation balances the critical node if **bf** of the critical node ≥ 2 and **bf** of its left child is ≤ 0
- A single Left rotation at the left child is followed by a single right rotation at the imbalanced node

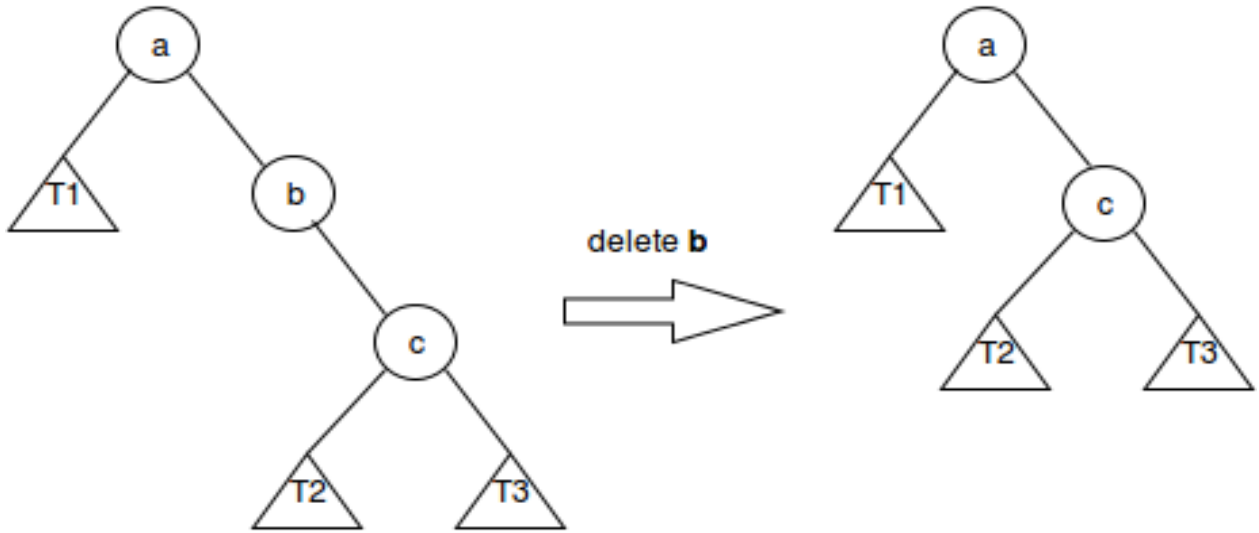


- **Insertions** and **deletions** may result in traversal upto the root

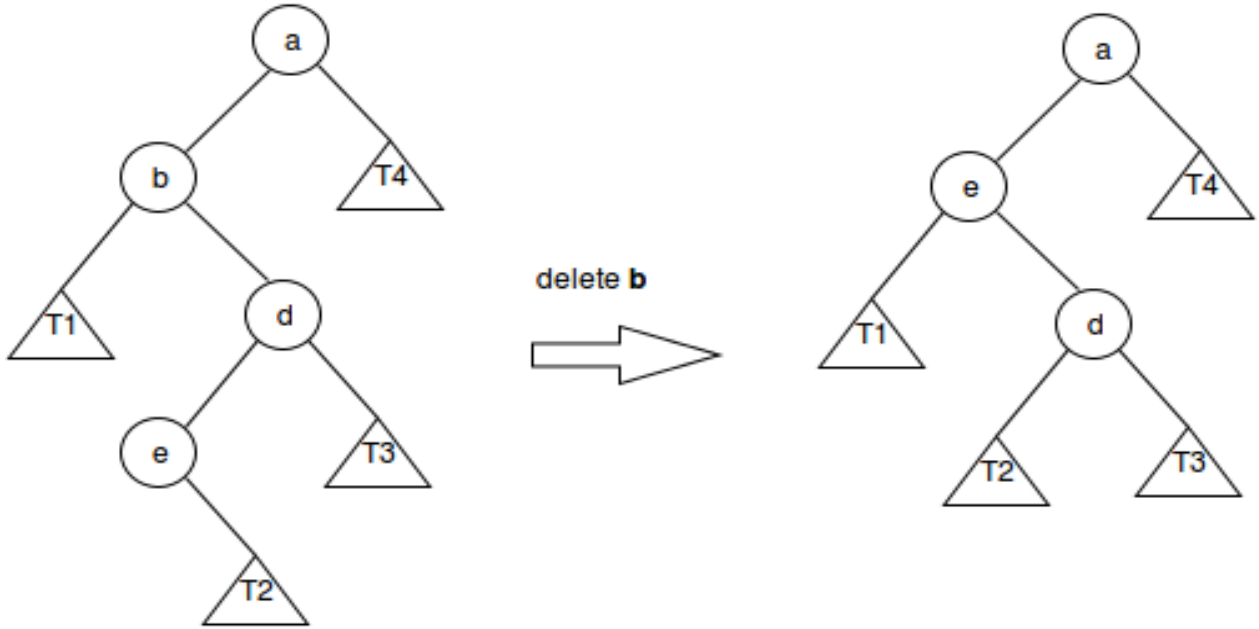
- Three cases while deleting a node
  - Node being deleted is a **leaf** - delete it and set its parent to point to null accordingly



  - Node with a **single child** it deleted - connect its parent and child and remove it physically

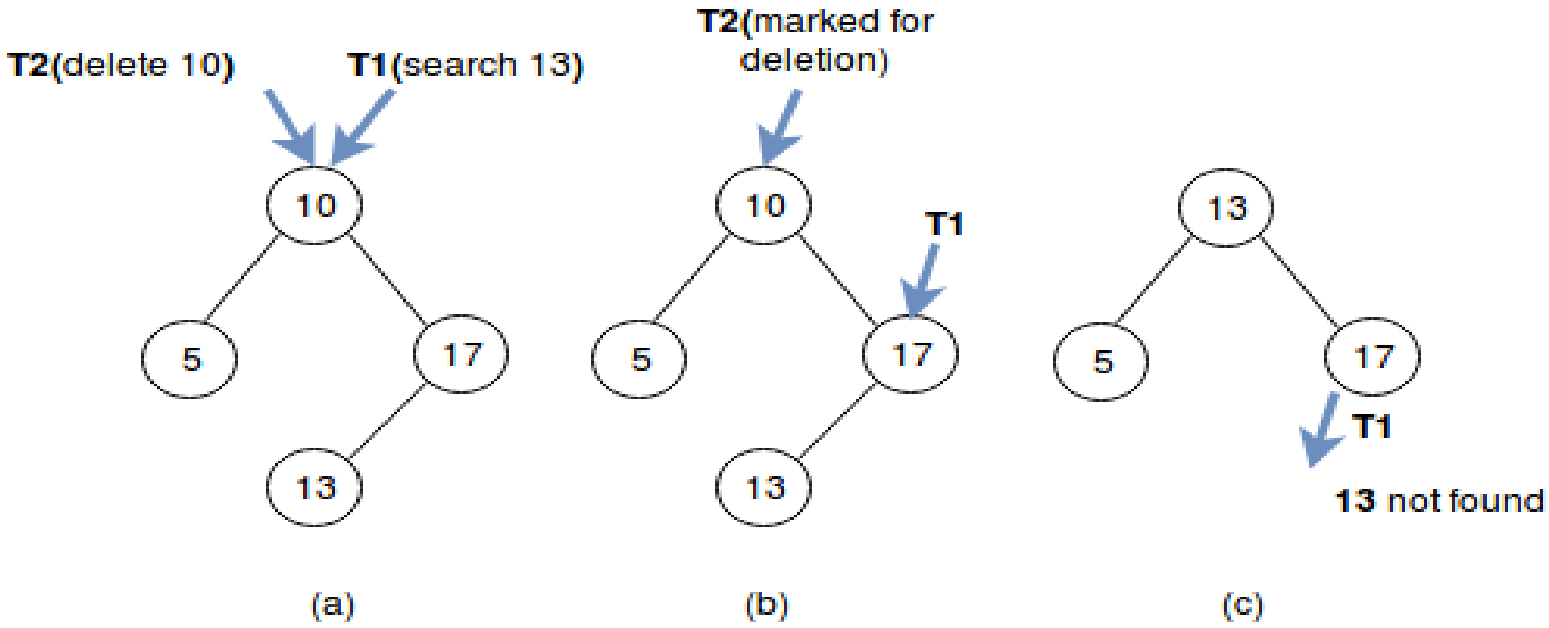


- Node with two children is deleted - Replace the key of node with key of its successor
- Delete the successor physically; connect its child to its parent



Motivation

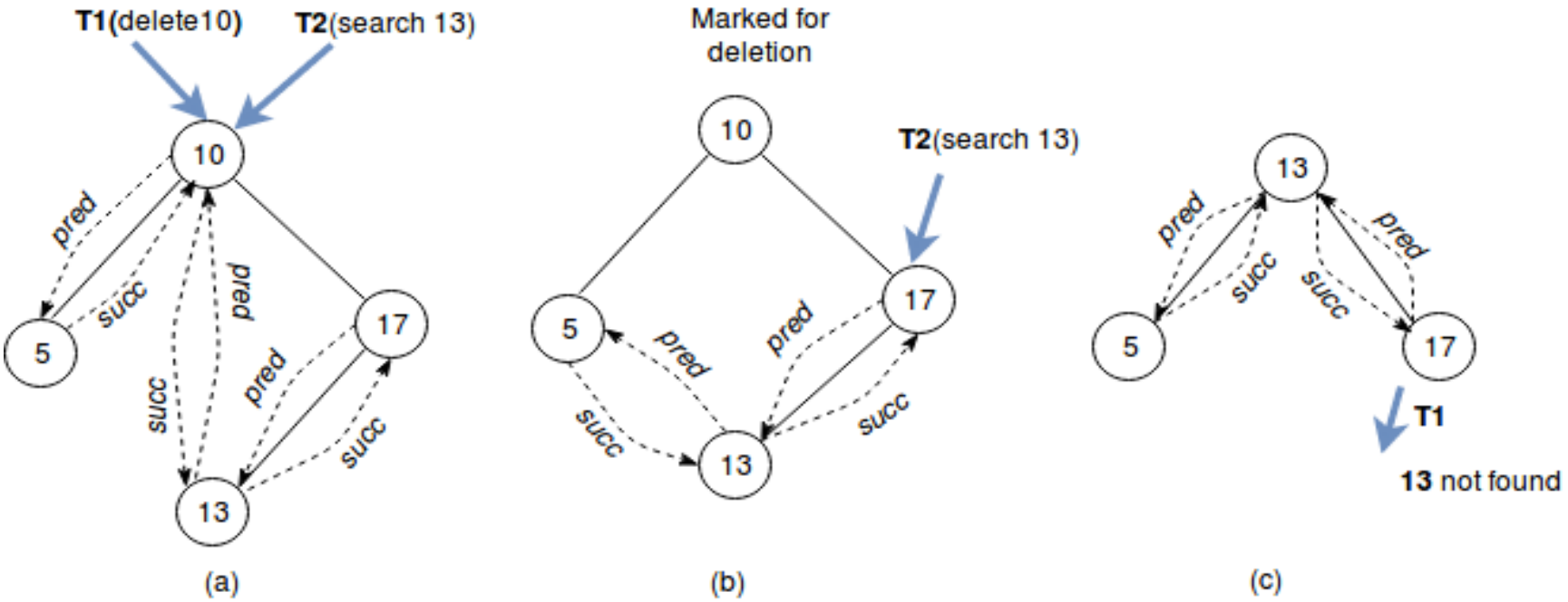
- Balanced BSTs are used for efficient lookup operations
- Update operations may cause mutations leading to change in physical layout of the tree
- In concurrent environment, lookup operations may return wrong results



- Insert and delete operations also require lookup before inserting/deleting an element

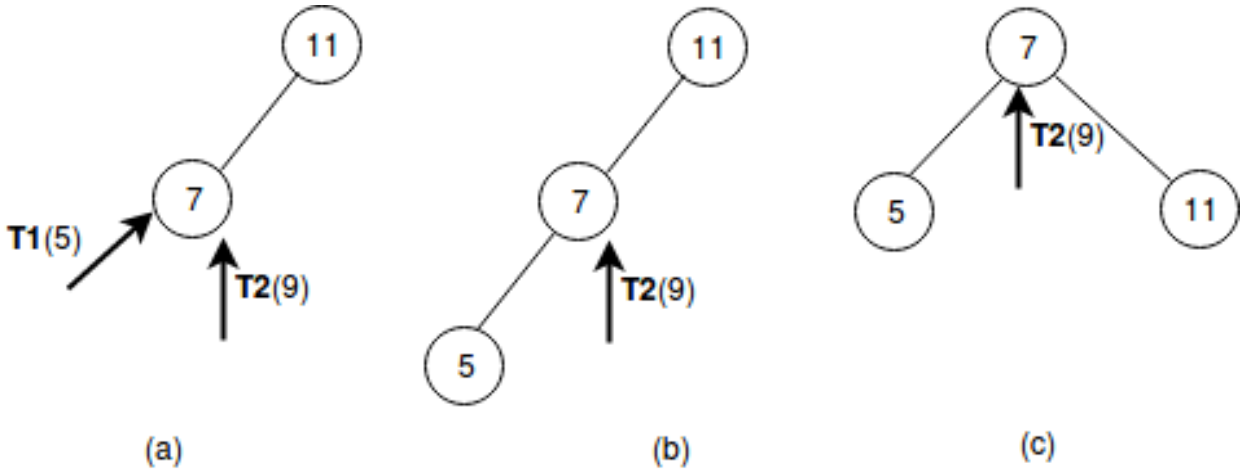
Concurrent Algorithms I

- **Logical Ordering** of tree elements can be used to perform lock-free search operations
- Independent of physical layout and stable under layout manipulations
- Requires additional fields to be incorporated per node - *succ*, *pred* and *succ_lock*



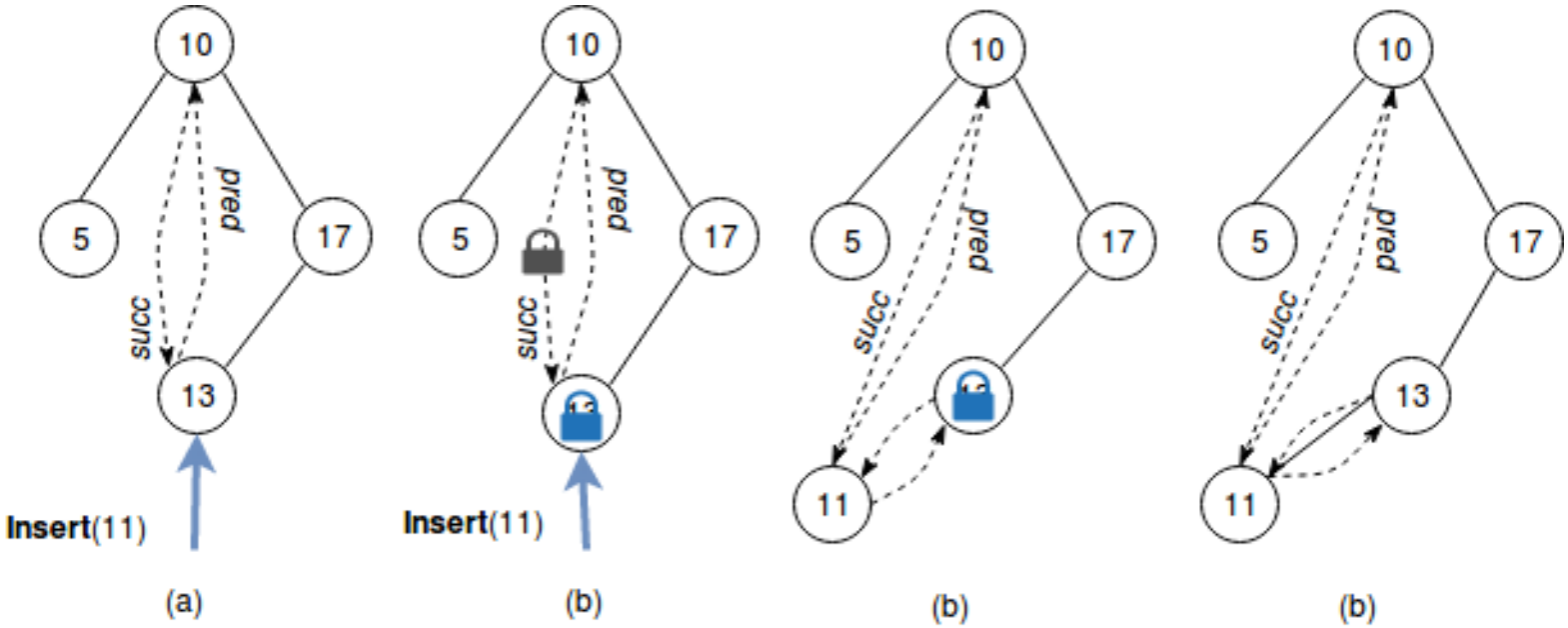
Concurrent Algorithms II

- If search fails, *succ* and *pred* pointers are traversed
- **Insert** requires choosing the parent correctly
- Rotations may alter the place of insertion



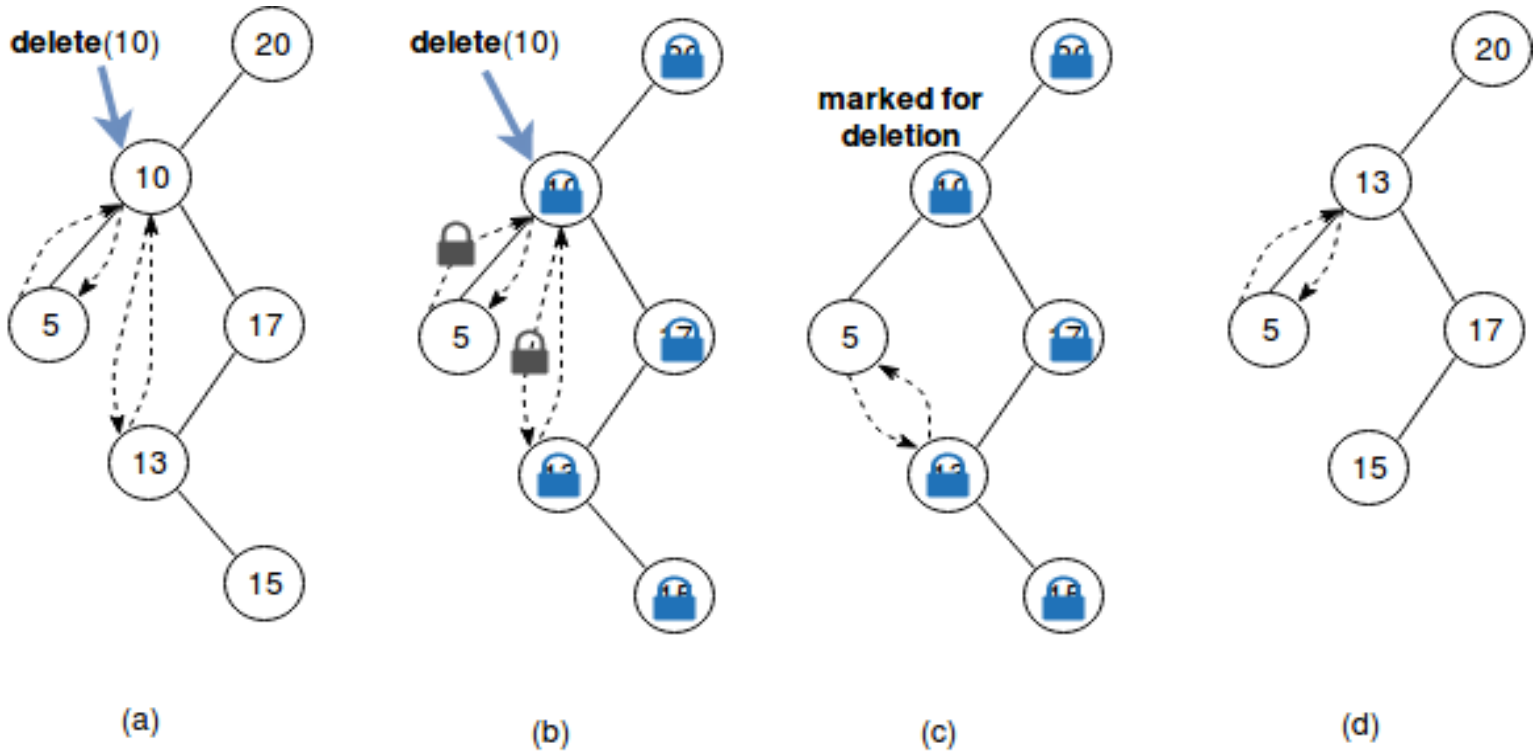
- Logical Ordering is updated by acquiring *succ_lock* on the predecessor
- Physical layout is updated by acquiring *tree_lock* on the parent followed by rebalancing operation if required

Concurrent Algorithms III



- **Delete** operation starts with searching for the node to be deleted
- If found, *succ_lock*s are acquired on the node and its predecessor
- Attempt is made to acquire *tree_lock*s on-
  - node and its parent
  - its child(if it exists), when it has less than two children
  - its successor(s), parent and right child(if exists) of s

Concurrent Algorithms IV

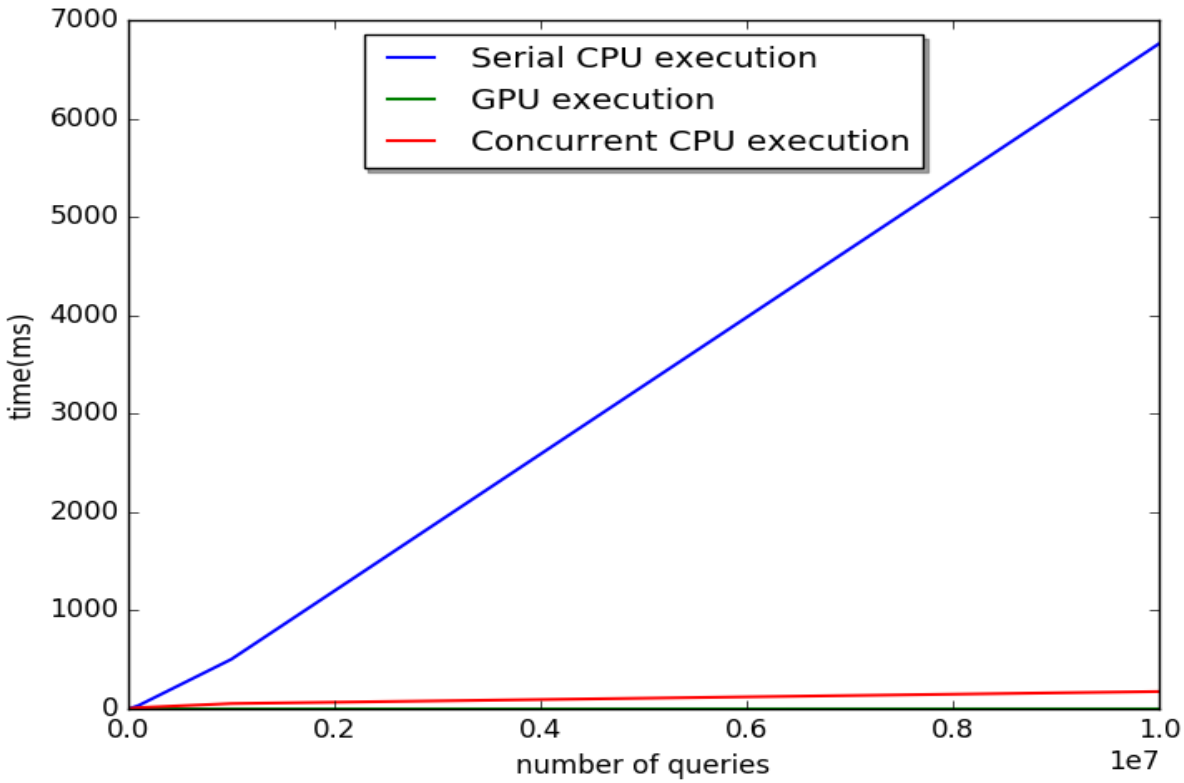


Contribution

- Cuda Implementation using locks fails to give any output for large datasets under heavy update operations
- Leveraged the idea of Logical Ordering to perform lock-free lookups concurrently
- Used Relaxed balance to allow only few update operations concurrently
- Outperforms concurrent cpu implementation for all sizes of the dataset

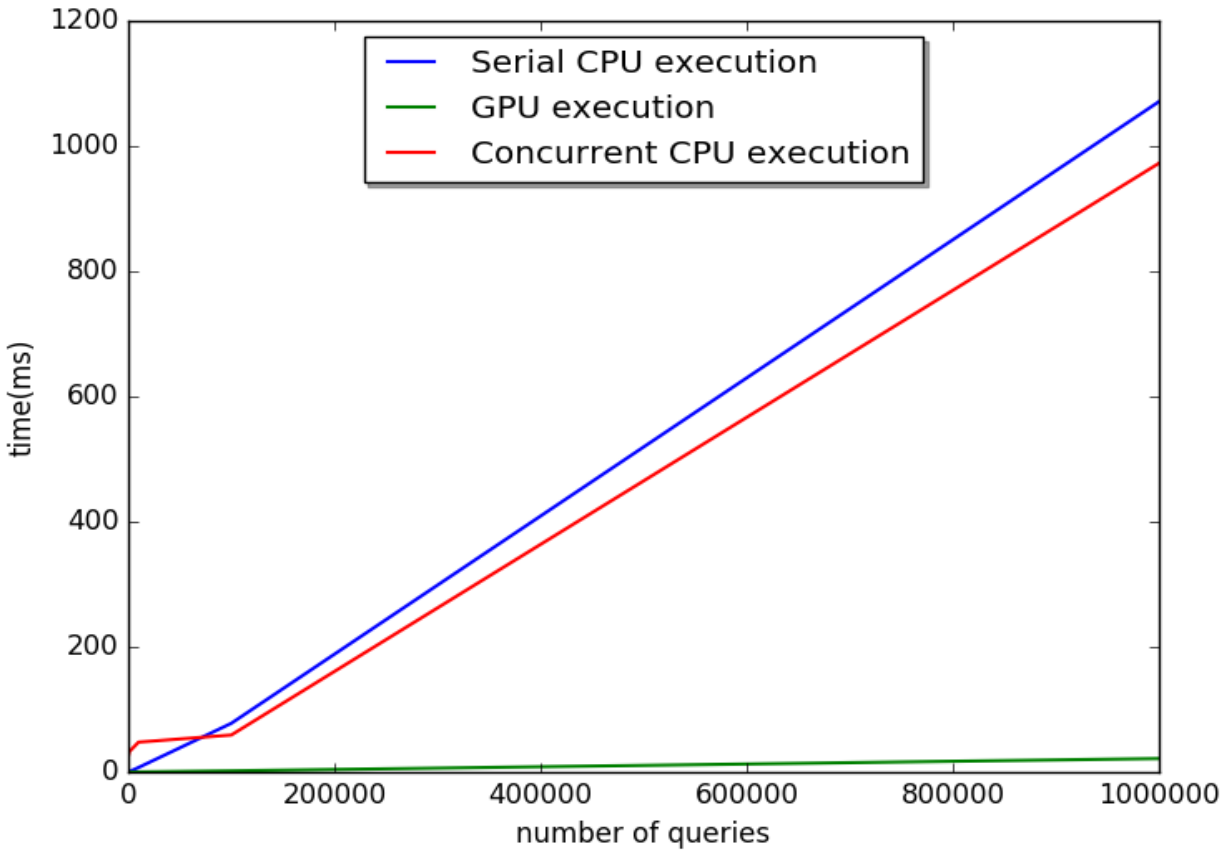
Results I

- **100% search:** GPU implementation performs $>$ 1000 times better
- Uses as many number of threads as the size of dataset as compared to just 128 threads used by cpu counterpart



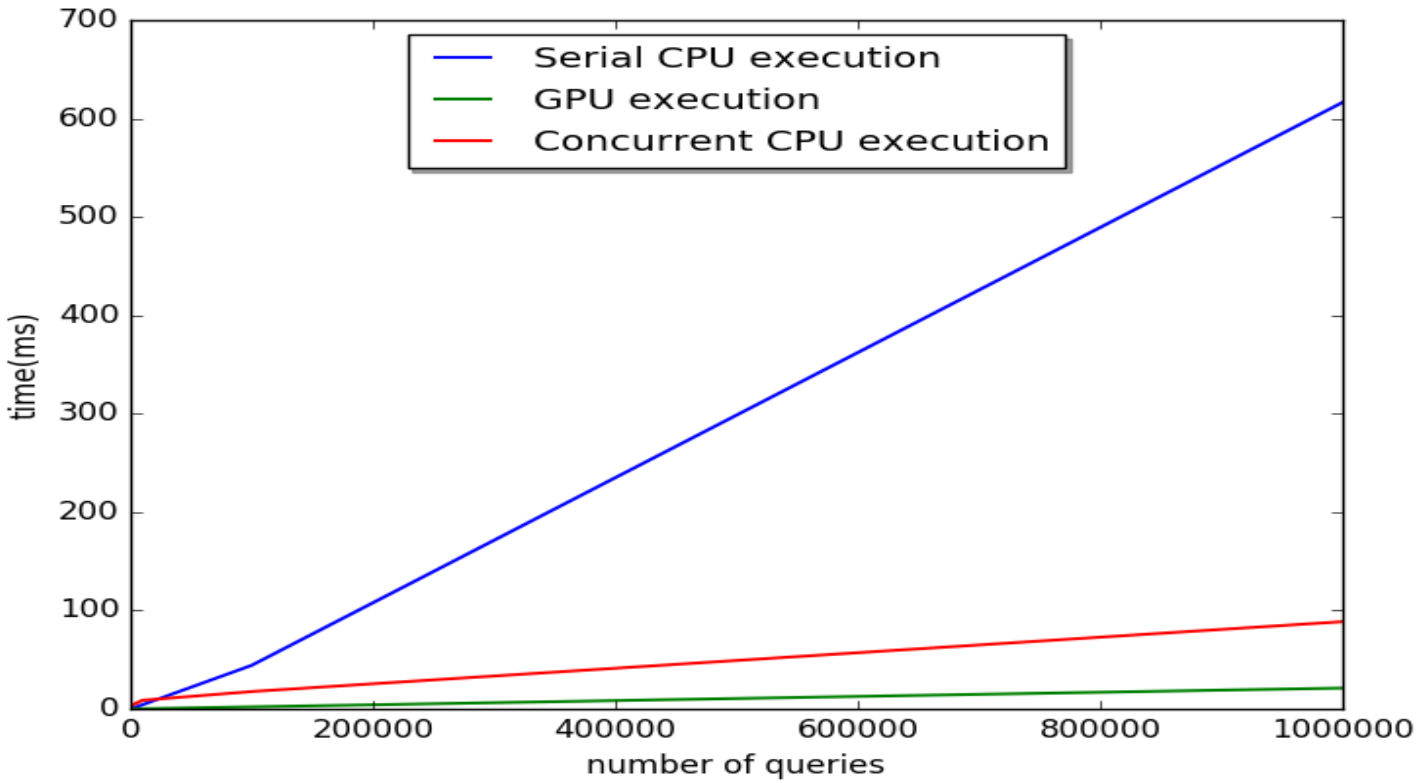
Results II

- **50% search,50% insert**



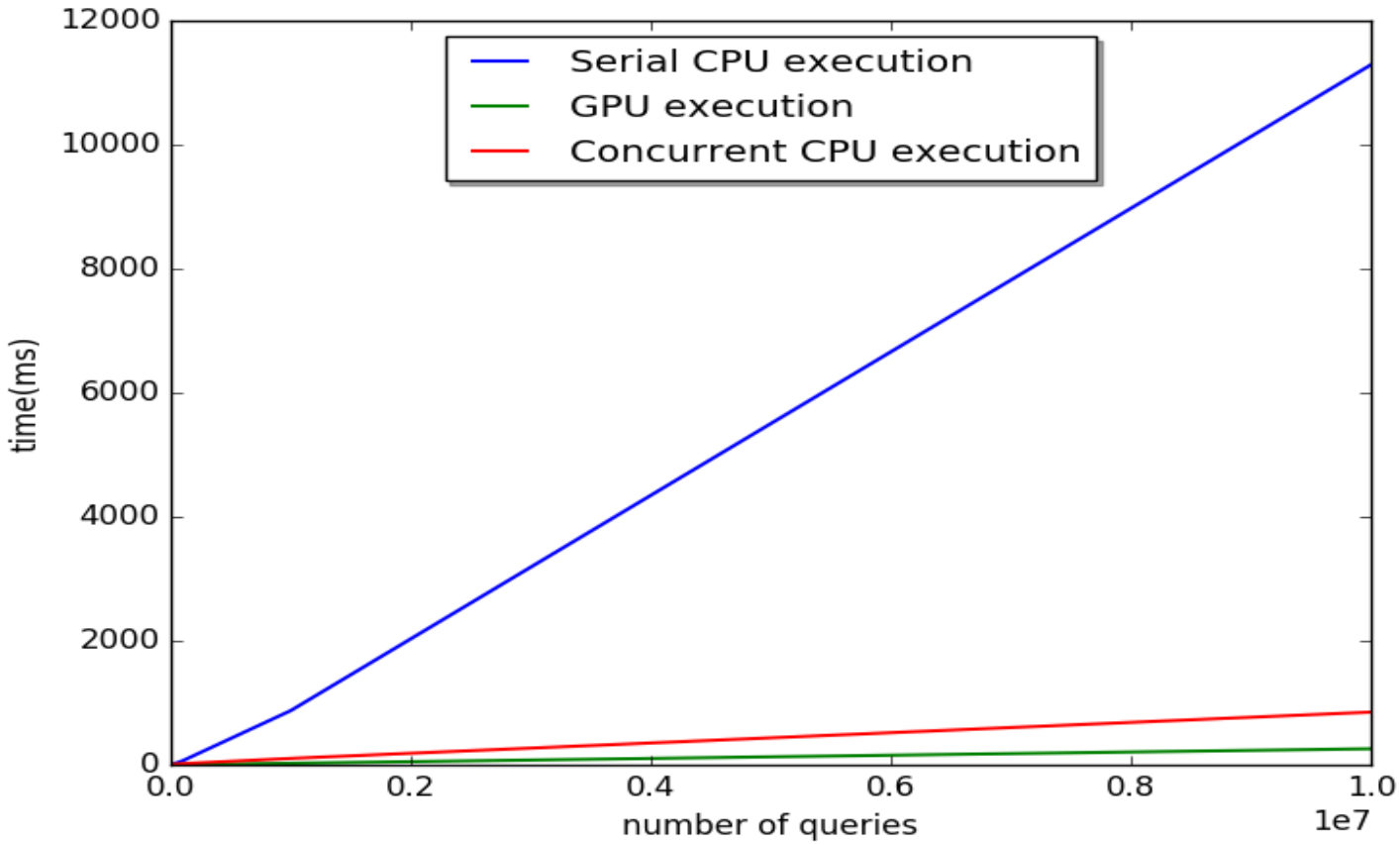
Results III

- **50% search,50% delete**



Results IV

- **50% search,25% insert, 25% delete**



Conclusion

- Use of Logical Ordering Layout helps in scaling the performance of lookup operations in concurrent environment
- Excessive use of locks degrades the performance of operations under heavy load conditions
- Concurrent operations perform worse than serialized operations under high contention
- GPUs are not suited for lock-based operations, perform poorly
- Mutating operations restrict the amount of concurrency in Balanced BSTs
- Concurrent CPU implementation fails to give any output within 30 mins for dataset of size 1 million

References

- Dana Drachsler, Martin Vechev. Eran Yahav; *Practical Concurrent Binary Search Trees via Logical Ordering*. PPoPP '14 Proceedings of the 19^{th} ACM SIGPLAN symposium on Principles and practice of parallel programming
- Nathan G. Bronson, Jared Casper. Hassan Chafi, Kunle Olukotun*A Practical Concurrent Binary Search Tree*. *PPoPP '10* Proceedings of the $15^{t}h$ ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming
- Zhang Yin and Xu Zhuoqun. *Concurrent Manipulation of Expanded AVL Trees*[J]. Journal of Computer Science and Technology, 1998,V13(4): 325-336
- Carla Schlatter Ellis, *Concurrent Search and Insertion in AVL Trees*. IEEE Transactions on Computers, September 1980

Thank You