Parallel Joinable B-Tree in Fork-Join I/O Model

Michael T. Goodrich, Yan Gu, Ryuto Kitagawa, Yihan Sun

ISAAC 2025
December 9, 2025

Parallel Computational Models

Parallel Computational Models

- Binary Fork Join Model

Parallel Computational Models

- Binary Fork Join Model
    - Assume each processor shares memory

Parallel Computational Models

- Binary Fork Join Model
  - Assume each processor shares memory
  - Threads may perform a *fork* operation

Parallel Computational Models

- Binary Fork Join Model
  - Assume each processor shares memory
  - Threads may perform a *fork* operation
  - Later performs a *join* operation for sync

Work-Span Model

Work-Span Model

- Represent algorithm's computation as a DAG

Work-Span Model

- Represent algorithm's computation as a DAG
  - Each node represents tasks, each edge is a dependency

Work-Span Model

- Represent algorithm's computation as a DAG
  - Each node represents tasks, each edge is a dependency
- Work: Total number of nodes in the graph

Work-Span Model

- Represent algorithm's computation as a DAG
    - Each node represents tasks, each edge is a dependency
- Work: Total number of nodes in the graph
- Span: Longest path within the graph

Cache Efficient Memory Accesses

Cache Efficient Memory Accesses

- Modern computers typically bottlenecked by memory accesses

Cache Efficient Memory Accesses

- Modern computers typically bottlenecked by memory accesses
  - Captured sequentially by the *External Memory Model*

Cache Efficient Memory Accesses

- Modern computers typically bottlenecked by memory accesses
  - Captured sequentially by the ***External Memory Model***
  - PRAM has the parallel equivalent, ***PEM***

Cache Efficient Memory Accesses

- Modern computers typically bottlenecked by memory accesses
  - Captured sequentially by the *External Memory Model*
  - PRAM has the parallel equivalent, *PEM*
  - Binary Fork Join model does not

Fork-Join I/O Model

Fork-Join I/O Model

- Natural extension of Binary Fork-Join Model

Fork-Join I/O Model

- Natural extension of Binary Fork-Join Model
    - All threads share external memory

Fork-Join I/O Model

- Natural extension of Binary Fork-Join Model
  - All threads share external memory
  - Each thread may access a block of data B in a single memory access

Fork-Join I/O Model

- Natural extension of Binary Fork-Join Model
  - All threads share external memory
  - Each thread may access a block of data B in a single memory access
  - Each fork and join spawns and syncs at most B threads

Join Based Balanced BST

Join Based Balanced BST

- Join-based framework

Join Based Balanced BST

- Join-based framework
  - Supports efficient set operations on search trees

Join Based Balanced BST

- Join-based framework
  - Supports efficient set operations on search trees
  - Core operations are the Join and Split operations

Join Based Balanced BST

- Join-based framework
  - Supports efficient set operations on search trees
  - Core operations are the Join and Split operations

B-Trees

B-Trees

- Multi-way cache efficient self-balancing tree

B-Trees

- Multi-way cache efficient self-balancing tree
- Each node in the tree contains b keys and $b + 1$ subtrees

B-Trees

- Multi-way cache efficient self-balancing tree
- Each node in the tree contains $\boldsymbol{b}$ keys and $\boldsymbol{b+1}$ subtrees
  - $\frac{B}{2} \leq b \leq B$

B-Trees

- Multi-way cache efficient self-balancing tree
- Each node in the tree contains $\boldsymbol{b}$ keys and $\boldsymbol{b+1}$ subtrees
  - $\frac{B}{2} \leq b \leq B$
- We seek to extend *Join* to *B-way Join*

B-Trees

- Multi-way cache efficient self-balancing tree
- Each node in the tree contains $\boldsymbol{b}$ keys and $\boldsymbol{b+1}$ subtrees
  - $\frac{B}{2} \leq b \leq B$
- We seek to extend *Join* to *B-way Join*
  - Eventually develop *Multi-way Join*

B-way Join

B-way Join

- General steps for the algorithm

B-way Join

- General steps for the algorithm
  - *Grouping*

B-way Join

- General steps for the algorithm
    - *Grouping*
    - *Recurse and Subtree Replace*

B-way Join

- General steps for the algorithm
  - *Grouping*
  - *Recurse and Subtree Replace*
  - *Concatenate and Rebalance*

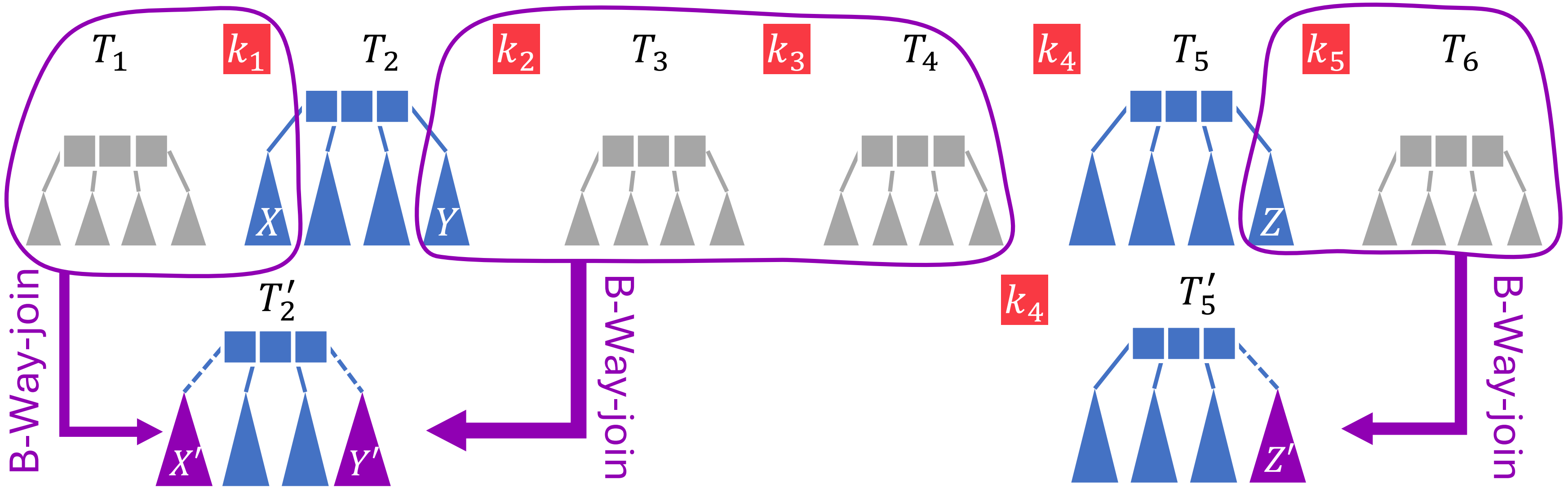
B-way Join

B-Way-join on five keys and six B-trees. $h(T_2) = h(T_5) = h^* = \max_i h(T_i)$, $B = 6$.

Input

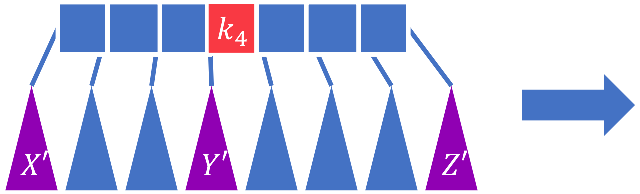
Step 1
Group inputs by the tall trees

Step 2
Recursively call Multi-join. Replace the left/right-most child of the tall trees



The purple trees have height $h^* - 1$ or height h^* (with one key at root). Note that T_2' and T_5' may not be strictly balanced at this point (X', Y' and Z' may be 1 level taller than other subtrees).

Step 3
Directly concatenate all (new) tall trees to the resulting tree & rebalance, based on Lemma 1



Resulting in a valid B-tree T with height h^* or $h^* + 1$; if T has height $h^* + 1$, then the root has at most $|C|$ keys, where $C \subseteq \{X', Y', Z'\}$ contains all trees with height h^*.

B-way Join

Theorem 1

Let $T_1, T_2, \ldots, T_{b+1}$ be a set of B-trees, with the largest tree height $h_{\max}$ and the shortest tree height $h_{\min}$, and $k_1, k_2, \ldots, k_b$ be a set of separator keys, where $b \leq B$. The *join* operation can be performed in $O(h_{\max} - h_{\min})$ I/O span and $O(b \cdot (h_{\max} - h_{\min}))$ I/O work.

B-way Join

Theorem 1

Let $T_1, T_2, \ldots, T_{b+1}$ be a set of B-trees, with the largest tree height $h_{\max}$ and the shortest tree height $h_{\min}$, and $k_1, k_2, \ldots, k_b$ be a set of separator keys, where $b \leq B$. The *join* operation can be performed in $O(h_{\max} - h_{\min})$ I/O span and $O(b \cdot (h_{\max} - h_{\min}))$ I/O work.

- Leads to a span-inefficient solution

B-way Join

Theorem 1

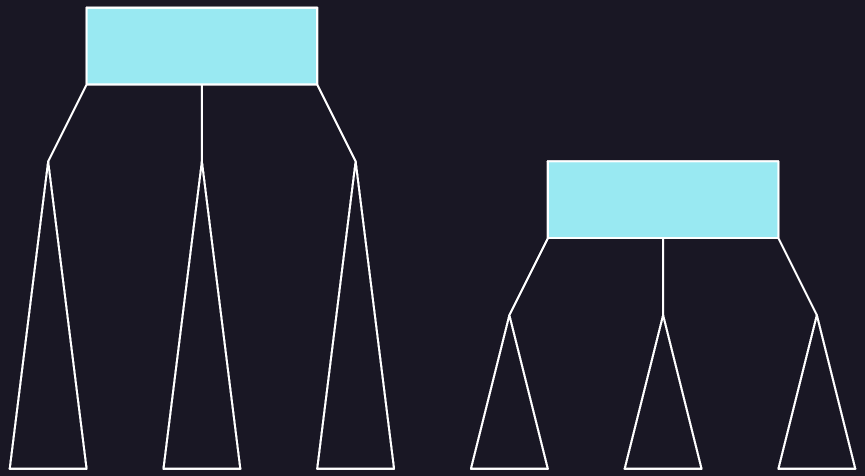
Let $T_1, T_2, \ldots, T_{b+1}$ be a set of B-trees, with the largest tree height $h_{\max}$ and the shortest tree height $h_{\min}$, and $k_1, k_2, \ldots, k_b$ be a set of separator keys, where $b \leq B$. The *join* operation can be performed in $O(h_{\max} - h_{\min})$ I/O span and $O(b \cdot (h_{\max} - h_{\min}))$ I/O work.

- Leads to a span-inefficient solution
- Union with above primitive leads to I/O Span $O(\log_B^2 n)$

B-way Join Improved

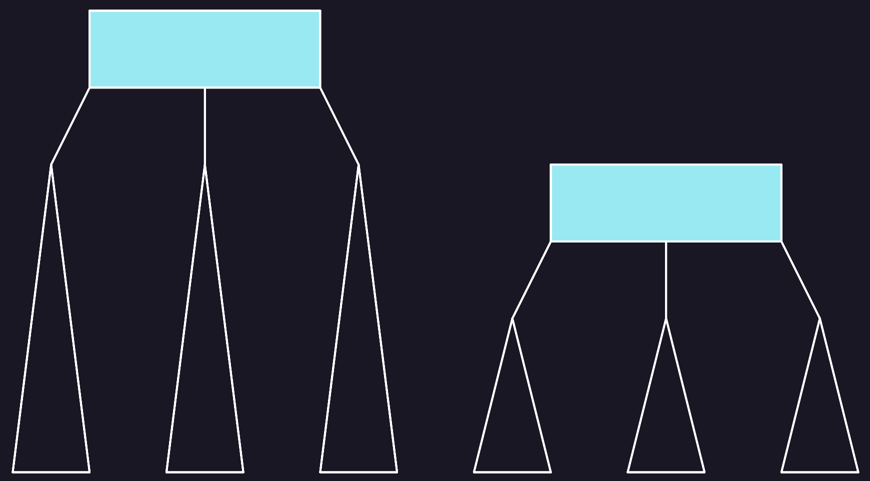
- *Grouping*
- Fuse within Groups
- Divide Large Nodes
- Divide Smaller Nodes
- *Concatenate and Rebalance*

Fuse within Groups



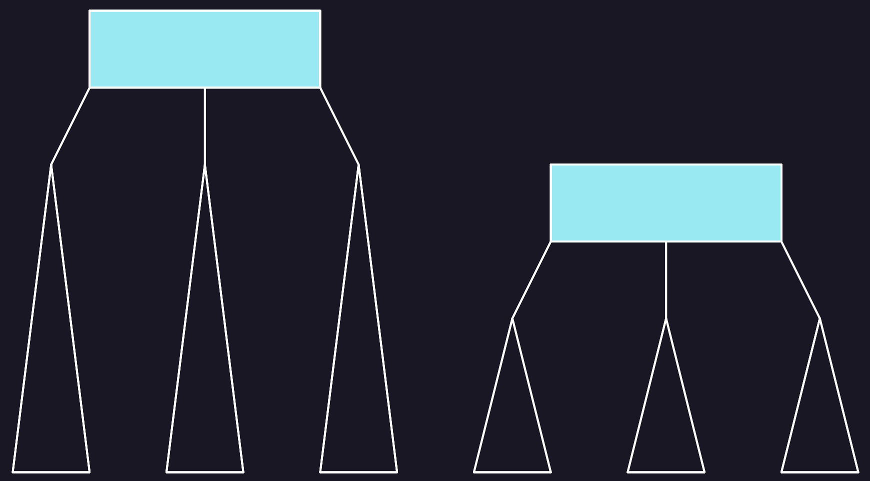
Fuse within Groups

- Do not recurse into subgroups of tall trees



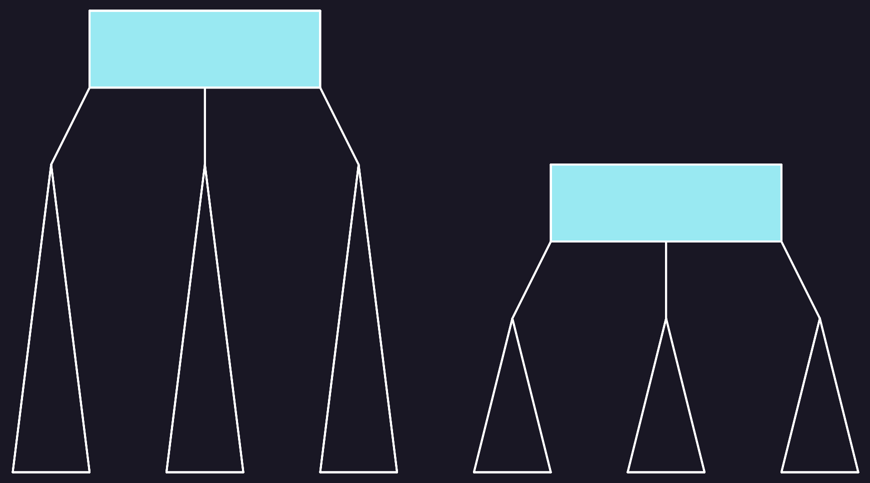
Fuse within Groups

- Do not recurse into subgroups of tall trees
    - *Fuse* keys and nodes at their heights directly



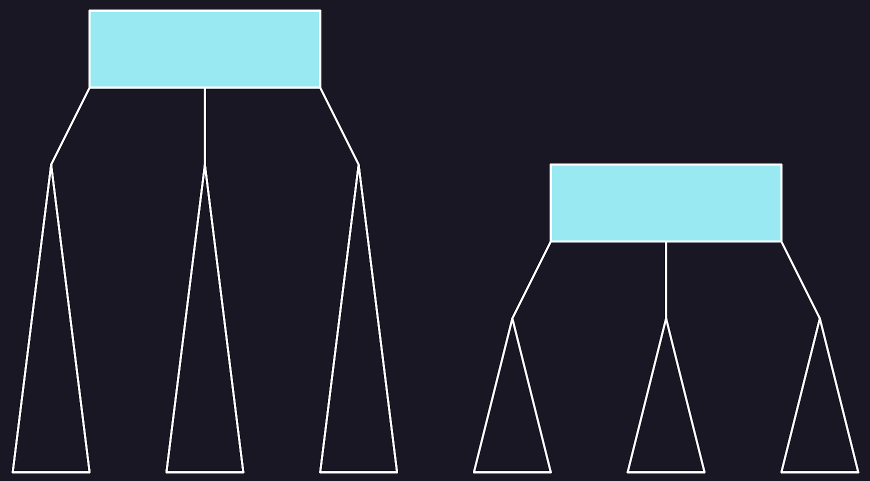
Fuse within Groups

- Do not recurse into subgroups of tall trees
  - *Fuse* keys and nodes at their heights directly
- Maintain list of pointers along the left and right spines



Fuse within Groups

- Do not recurse into subgroups of tall trees
  - *Fuse* keys and nodes at their heights directly
- Maintain list of pointers along the left and right spines
- Finding the node a root merges with takes $O(\log_B \log_B n)$ I/O span



Dividing Large Nodes

Dividing Large Nodes

- Divide node if contains $> B$ keys

Dividing Large Nodes

- Divide node if contains $> B$ keys
  - Any node contains at most B^2 keys

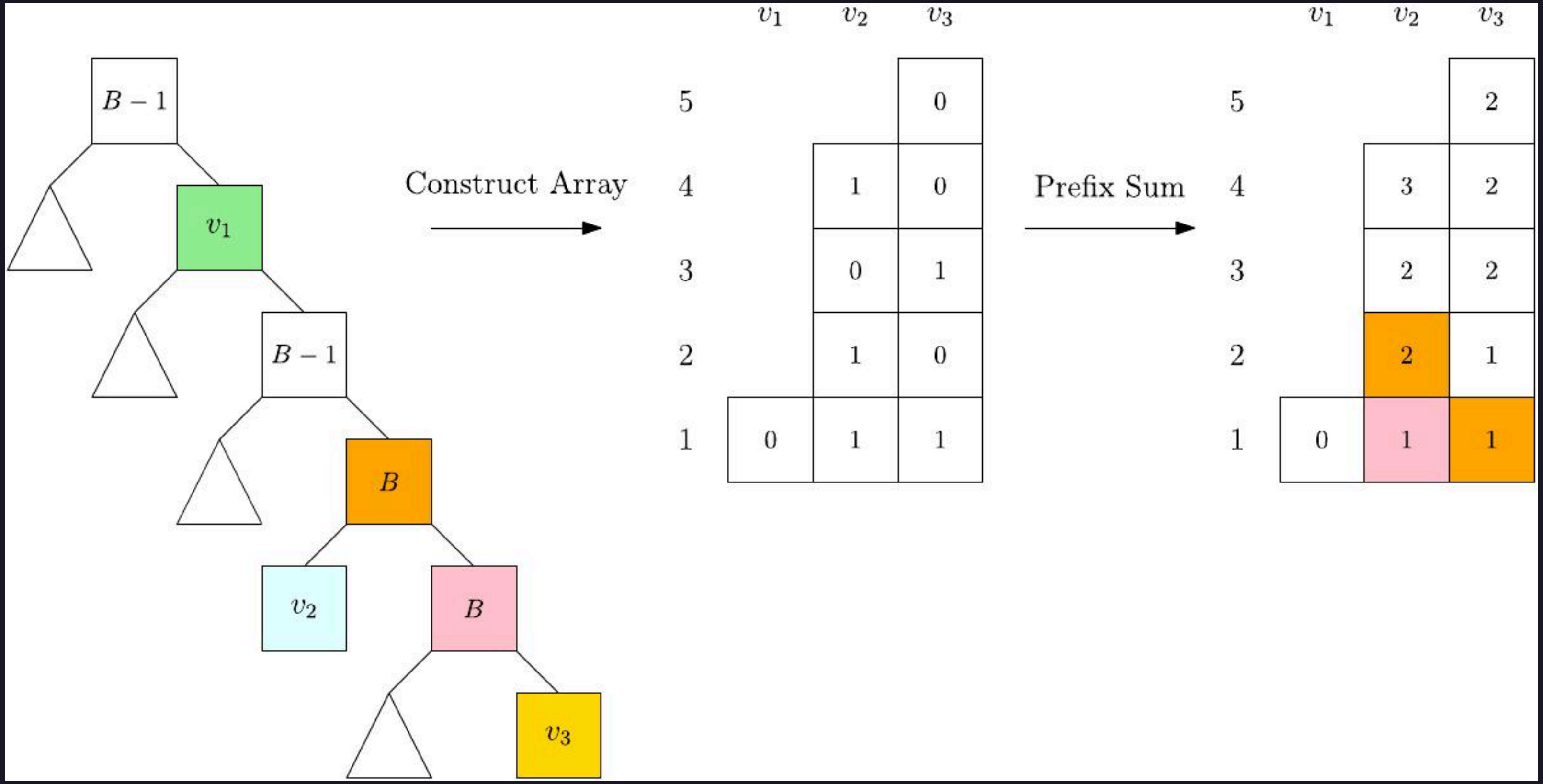
Dividing Large Nodes

- Divide node if contains $> B$ keys
  - Any node contains at most B^2 keys
- After, each node contains at most $2B$ keys

Dividing Large Nodes

- Divide node if contains $> B$ keys
  - Any node contains at most B^2 keys
- After, each node contains at most $2B$ keys
  - Each subsequent dividing of nodes will push up at most 1 key

Dividing Smaller Nodes



Multi-way Join I/O Span and Work

Theorem 2

We can join $T_1, \ldots, T_d$ B-trees and $k_1, \ldots, k_d$ keys together in parallel with $O(\log_B d \cdot \log_2 \log_B n + \log_B n)$ I/O span and $O(d \log_B n)$ I/O work, where n is $\sum_{i=1}^{d} |T_i|$, in the Fork-Join I/O model.

Multi-way Split I/O Span and Work

Theorem 3

We can split a B-tree T by $k_1, \ldots, k_d$ keys in parallel with $O(\log_B d + \log_B n)$ I/O span and $O(d \log_B n)$ I/O work, where n is $\sum_{i=1}^{d} |T_i|$, in the Fork-Join I/O Model.

Union Operation

Union Operation

- Divide each tree into $\sqrt{n+m}$ sections

Union Operation

- Divide each tree into $\sqrt{n+m}$ sections
    - Use the Multi-way Split operation

Union Operation

- Divide each tree into $\sqrt{n+m}$ sections
  - Use the Multi-way Split operation
- Perform a union on each pair of subtrees

Union Operation

- Divide each tree into $\sqrt{n+m}$ sections
  - Use the Multi-way Split operation
- Perform a union on each pair of subtrees
- Use Multi-way Join to combine all elements in the tree

Union Operation

- Divide each tree into $\sqrt{n+m}$ sections
    - Use the Multi-way Split operation
- Perform a union on each pair of subtrees
- Use Multi-way Join to combine all elements in the tree
    - Minor modifications may be made to Multi-way Join to get Intersection and Difference operations

Union Operation

Theorem 4

Given two B-trees with sizes m and $n \geq m$, there exists a parallel algorithm that returns a new B-tree containing the union, intersection, and set difference of the two input trees in and $O\left(m \log_B \left(\frac{n}{m}\right)\right)$ I/O work, $O(\log_B m \cdot \log_2 \log_B n + \log_B n)$ I/O span, where B is the block size.

Future Work

Future Work

- Turning $\log_2$ to $\log_B$

Future Work

- Turning $\log_2$ to $\log_B$
- Removing $\log\log$ factor entirely

Future Work

- Turning $\log_2$ to $\log_B$
- Removing $\log\log$ factor entirely
- Apply algorithm to adaptive sorting settings

Future Work

- Turning $\log_2$ to $\log_B$
- Removing $\log\log$ factor entirely
- Apply algorithm to adaptive sorting settings
- Use Fork-Join I/O model on other algorithms

Thank you!