Authors of the Paper: Akimasa Morihata, Kiminori Matsuzaki

Affiliation: Tohoku University, Kochi University of Technology

Presented by: Zhili Pan, Samuel Tepoorten

Balanced Trees Inhabiting Functional Parallel Programming

INTRODUCTION

Problem

Inefficient parallelism in sequential data structures like lists and trees.

INTRODUCTION

122

Key Points

- *Traditional sumList is inherently sequential:*

$$\mathrm{sumList}\,[\,] = 0 \quad \mathrm{sumList}\,(a : x) = a + \mathrm{sumList}\,x$$

- *Divide-and-conquer cannot be applied efficiently on standard lists.*

Solution

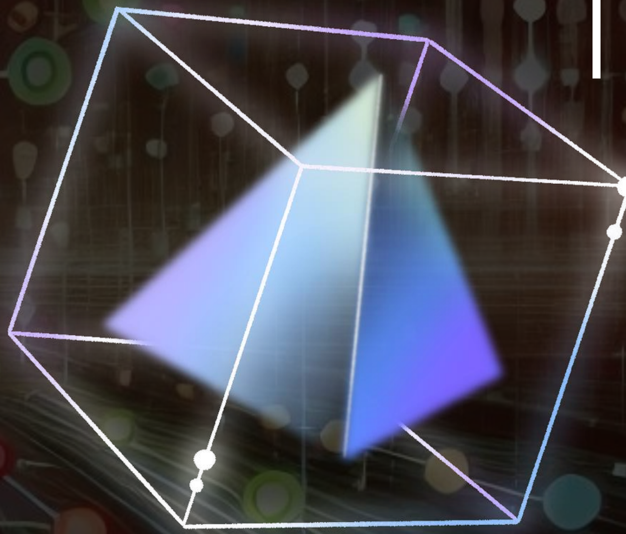
- *Use Balanced Trees to support divide-and-conquer strategies and enable parallelism.*

Goals

- *Relate parallel algorithms to* ***balanced trees.***
- *Implement recursive parallel functions on these trees.*
- *Address the lack of concrete implementations for balanced* ***Shunt Trees.***

OVERVIEW

- Preliminaries
- Join Lists
- Balancing Join Lists
- Shunt Trees
- Implementation Examples
- Main Work done
- Remaining Work
- Summary



PRELIMINARIES

Core concept:

- *Greek letters (α, β, γ) for type variables (polymorphism)*

- ***Identity Function:***

$$\mathrm{id} :: \forall \alpha . \alpha \to \alpha$$

Haskell Constructs:

- ***Single** and **Join:** Basic constructors for Join Lists*

- ***Associative operators** $\oplus$, $\otimes$ **enable parallel recombination.***

Why Join Lists?

- *Join Lists transform **sequential lists** into binary tree structures, enabling parallel processing.*

List Homomophisms

Consider list aggregating opetions, like sum, max, and etc:

- If we abstracting out the concrete computation, we obtain a skeleton as following:

$$h :: \forall\beta. (\beta \rightarrow \beta \rightarrow \beta) \rightarrow (A \rightarrow \beta) \rightarrow [\mathrm{A}] \rightarrow \beta$$

Let's say we have a function $(\oplus)$ s.t: $\boldsymbol{\beta} \rightarrow \boldsymbol{\beta} \rightarrow \boldsymbol{\beta}$
and a function $\boldsymbol{f}$ s.t $\boldsymbol{A} \rightarrow \boldsymbol{\beta}$

Then we can define :

$$h(\oplus) f [a] = f a$$

$$h(\oplus) f (x + y) = h(\oplus) f x \oplus h(\oplus) f y$$

Assuming that:

- $\oplus$ is associative: $g(g(x, y), z) = g(x, g(y, z))$

Example

- $Sum\ x = h\ (+)\ \ id\ \ x$

List Homomophisms

<u>Definition</u>: a Polymorhpic function

$$h :: \forall \beta. (\beta \to \beta \to \beta) \to (A \to \beta) \to [\mathrm{A}] \to \beta$$

Is to be a list-homomorphism scheme if it satisfies

$$h\ (+\,+)\ (\lambda a \to [a])\ x = x$$

for any list x

- This ensure that this list aggregation structure doesn't not duplicate, discard, or disorder elements

Join Lists

(1)

Definition:

$$\text{data JList}\ \alpha = \text{Single}\ \alpha \mid \text{Join}\ (\text{JList}\ \alpha)\ (\text{JList}\ \alpha)$$

(2)

Conversion:
From Join List back to sequential list:

$$\text{j2l}\ (\text{Single}\ a) = [a] \quad \text{j2l}\ (\text{Join}\ l\ r) = \text{j2l}\ l + +\text{j2l}\ r$$

(3)

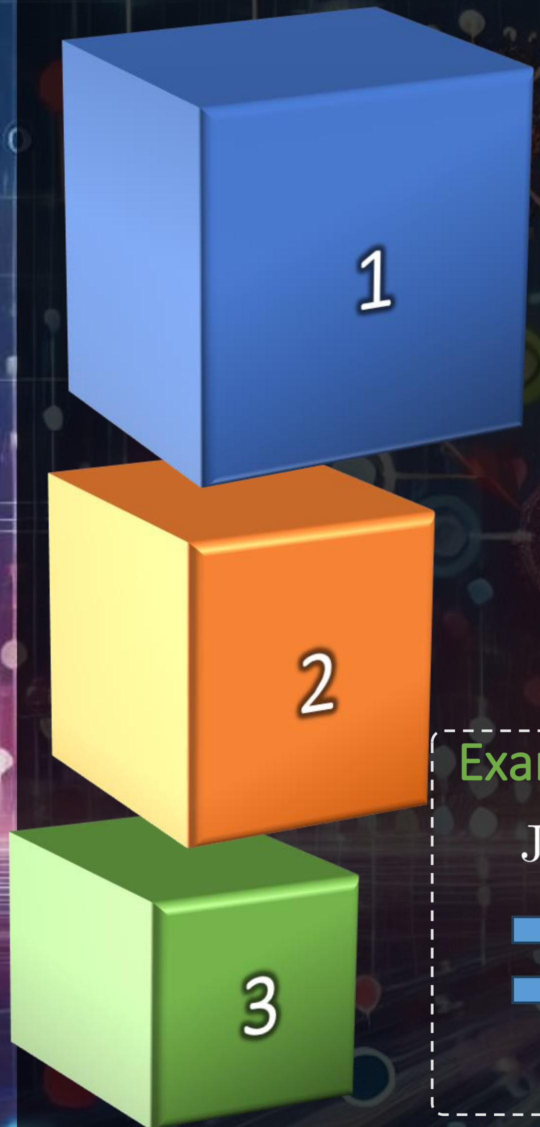
Parallel Sum Example:

- *sumJ*:

$$\text{sumJ}\ (\text{Single}\ a) = a \quad \text{sumJ}\ (\text{Join}\ l\ r) = \text{sumJ}\ l + \text{sumJ}\ r$$

- *Steps*
  - *Split list into subtrees*
  - *Compute sums in parallel*
  - *Combine results: efficient divide-and-conquer*

Balancing Join Lists



Why Balance Matters

- Balanced Join Lists ensure O(log N) height.
- Supports efficient divide-and-conquer parallel operations.

Techniques for Balancing

- Use any **self-balancing** binary tree that stays balanced after modification. i.e. ***Biased Search Trees***, etc..
- *Ropes*: Efficient for frequent concatenations.

Example

$$\text{Join (Join (Single 3) (Single 5)) (Join (Single 2) (Single 7))}$$

Step 1: Compute (3+5) and (2+7) in parallel.
Step 2: Combine results: (3+5) + (2+7)=17

Shunt Trees

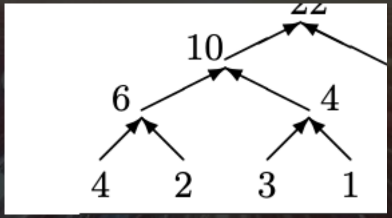
we will see how the Shunt Contraction Scheme enables efficient and parallel processing of tree structures, in order to introduce Shunt Tree Strutures.

Binary Tree Structure:
A binary tree consists of:

- *Leaves (Tip): Values A*
- *Internal nodes (Bin): Connect two subtrees and hold values*

Equation:

$$\text{data Tree}\ \alpha\,\beta = \text{Tip}\ \alpha \mid \text{Bin}\ (\text{Tree}\ \alpha\,\beta)\ \beta\ (\text{Tree}\ \alpha\,\beta)$$



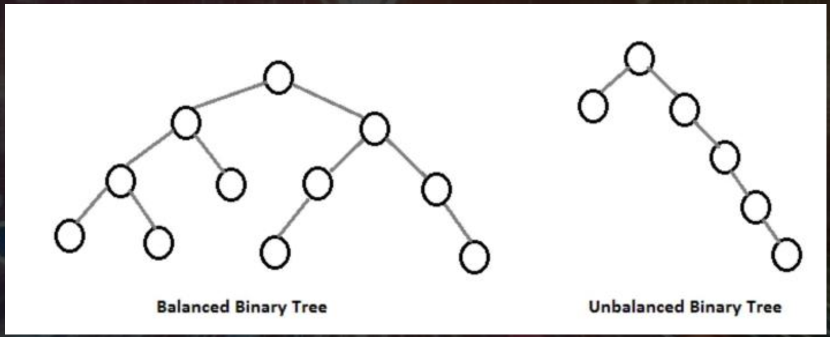
Shunt Trees

The Challenge

The Challenge arise when we are talking about parallelization

- *Unbalanced (list-like) trees require O(h) steps, where h is the height.*
- *Parallelizing operations on trees requires reducing the tree depth.*

How to address the issue?
Here is where Shunt and Tree Contraction Concepts comes into play



Shunt Trees

The Shunt Operation

The Shunt operation perform the following:

1. *Removes a leaf and its parent node.*
2. *Connects the subtree directly to its grandparent node.*

Formal equation:

$$\text{shunt } \forall \alpha, \beta. \text{Tree } \alpha\, \beta \rightarrow \text{Tree } \alpha\, \beta$$

$$\text{shunt } (\text{Bin } t\_ \ (\text{Tip\_})) = t \quad \text{and} \quad \text{shunt } (\text{Bin } (\text{Tip\_})\ t\_) = t$$

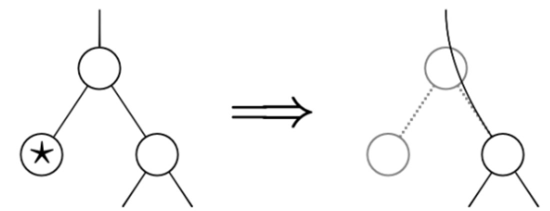


Figure 2. A Shunt operation applied to the star-marked leaf.

Shunt Trees

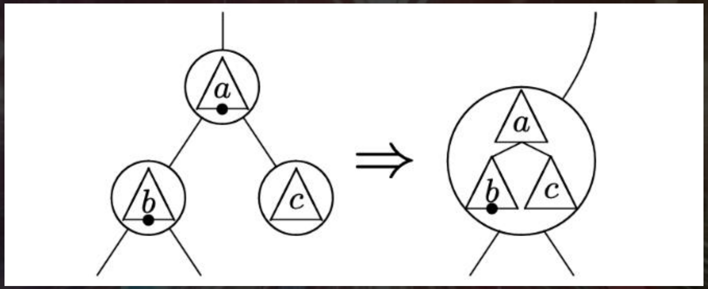
Shunt Contraction Scheme

The scs function generalizes the Shunt operation, allowing us to:

1. *Process leaves using A→**α**.*
2. *Process internal nodes using B→**β**.*
3. *Combine intermediate results with Shunt **α β***

Equation:

$$\text{scs} :: \forall \alpha, \beta. (A \to \alpha) \to (B \to \beta) \to \text{Shunt } \alpha\beta \to \text{Tree } AB \to \alpha$$



Shunt Trees

Example

Let's use scs to compute the sum of values in a tree

1. *Leaves: Processed with identity (id)*
2. *Nodes: Aggregated using the function add3*

Code:

$$\text{sumTree} = \text{scs id id (add3, add3, add3)}$$

$$\text{where add3}\, b\, a\, c = b + a + c$$

Shunt Trees

Hole and Connect

To ensure that the tree is reconstructed correctly during the Shunt contraction, we use two key concepts

1. *Hole: Creates a context with a "hole" that represents a missing subtree.*
2. *Connect: Combines a context with subtrees to reconstruct the tree.*

Definitions:

- *Hole:*

$$\text{hole} :: \forall \alpha, \beta . \beta \to \text{Context } \alpha \, \beta$$
$$\text{hole } a = \lambda(l, r) \to \text{Bin l a r}$$

- *Connect:*

$$\text{connect} :: \forall \alpha, \beta . \text{Shunt } (\text{Tree } \alpha \, \beta) \, (\text{Context } \alpha \, \beta)$$
$$\text{connect} = (\psi_L, \psi_R, \psi_N)$$

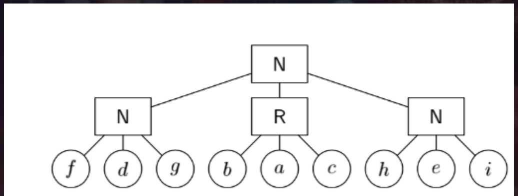
Shunt Trees

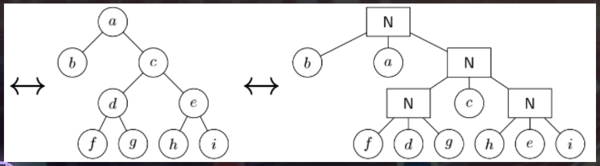
Building Shunt Trees

Shunt Trees are derived structures that

1. *Reduce the tree depth.*
2. *Support parallelization of operations.*

Definition:





$$\begin{aligned}\textbf{data ST}\ \alpha\,\beta &= \mathrm{T}\ \alpha\\ &\mid \mathrm{N}\ (\mathrm{ST}\ \alpha\,\beta)\ (\mathrm{ST}_{\bullet}\ \alpha\,\beta)\ (\mathrm{ST}\ \alpha\,\beta)\end{aligned}$$

$$\begin{aligned}\textbf{data ST}_{\bullet}\ \alpha\,\beta &= \mathrm{H}_{\bullet}\ \beta\\ &\mid \mathrm{L}_{\bullet}\ (\mathrm{ST}_{\bullet}\ \alpha\,\beta)\ (\mathrm{ST}_{\bullet}\ \alpha\,\beta)\ (\mathrm{ST}\ \alpha\,\beta)\\ &\mid \mathrm{R}_{\bullet}\ (\mathrm{ST}\ \alpha\,\beta)\ (\mathrm{ST}_{\bullet}\ \alpha\,\beta)\ (\mathrm{ST}_{\bullet}\ \alpha\,\beta)\end{aligned}$$

With $ST\cdot$ for Context, ST for binary tree

Shunt Trees

Balancing Trees

Definition:

Balancing is necessary to reduce tree height and fully exploit parallelism.
The contraction algorithm reduces tree height to $O(\log N)$

Theorem:

Execution time: $O(\frac{N}{P} + \log P)$

Implementation Examples

(1)

Sum on Shunt Trees:

Definition:

$$\mathrm{sumT}\,(\mathrm{T}\,a) = a \quad \mathrm{sumT}\,(\mathrm{N}\,b\,a\,c) = \mathrm{sumT}\,b + a + \mathrm{sumT}\,c$$

(2)

Parallel Execution:

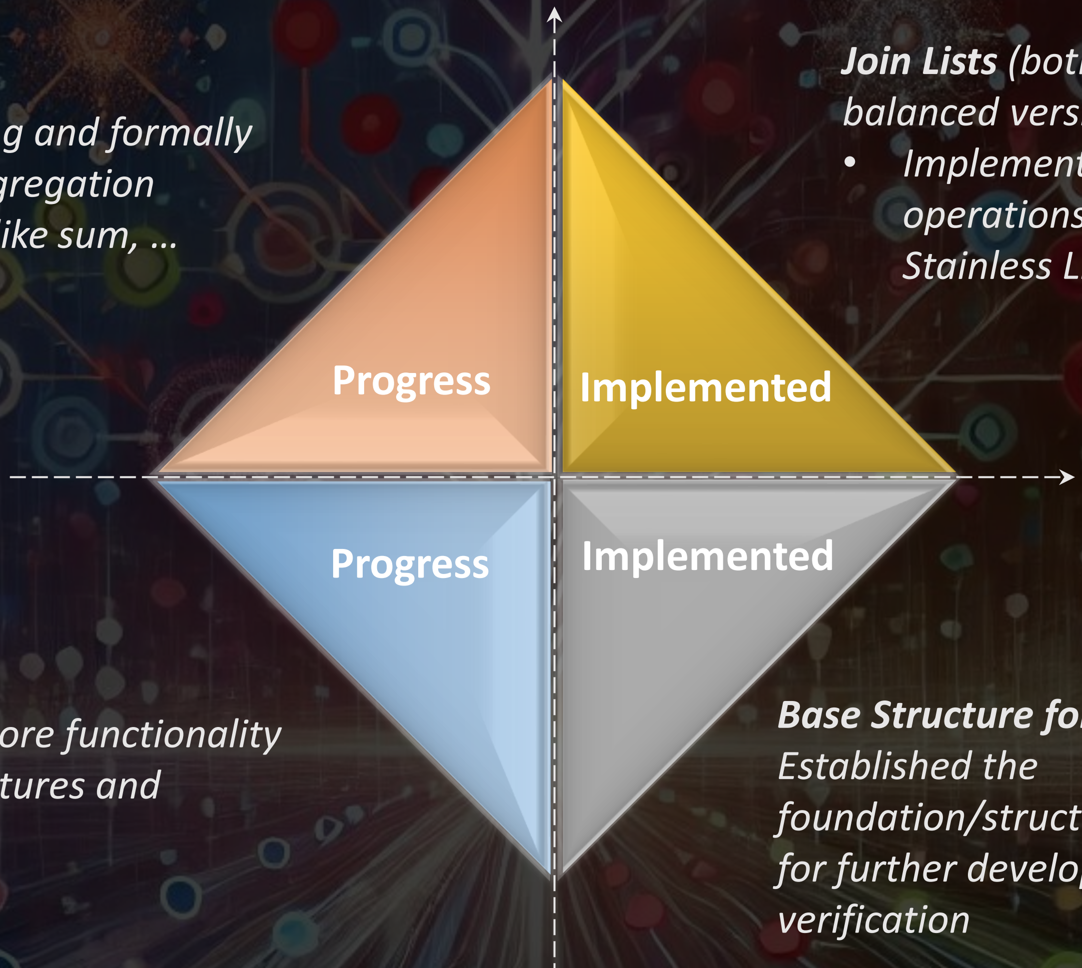
- Divide computation across subtrees.
- Combine results efficiently in a ***bottom-up*** fashion.

Main Work Done

Implementing and formally verifying aggregation operations: like sum, …

Join Lists *(both unbalanced and balanced version)*

- *Implemented and verified operations as supported in Stainless List*



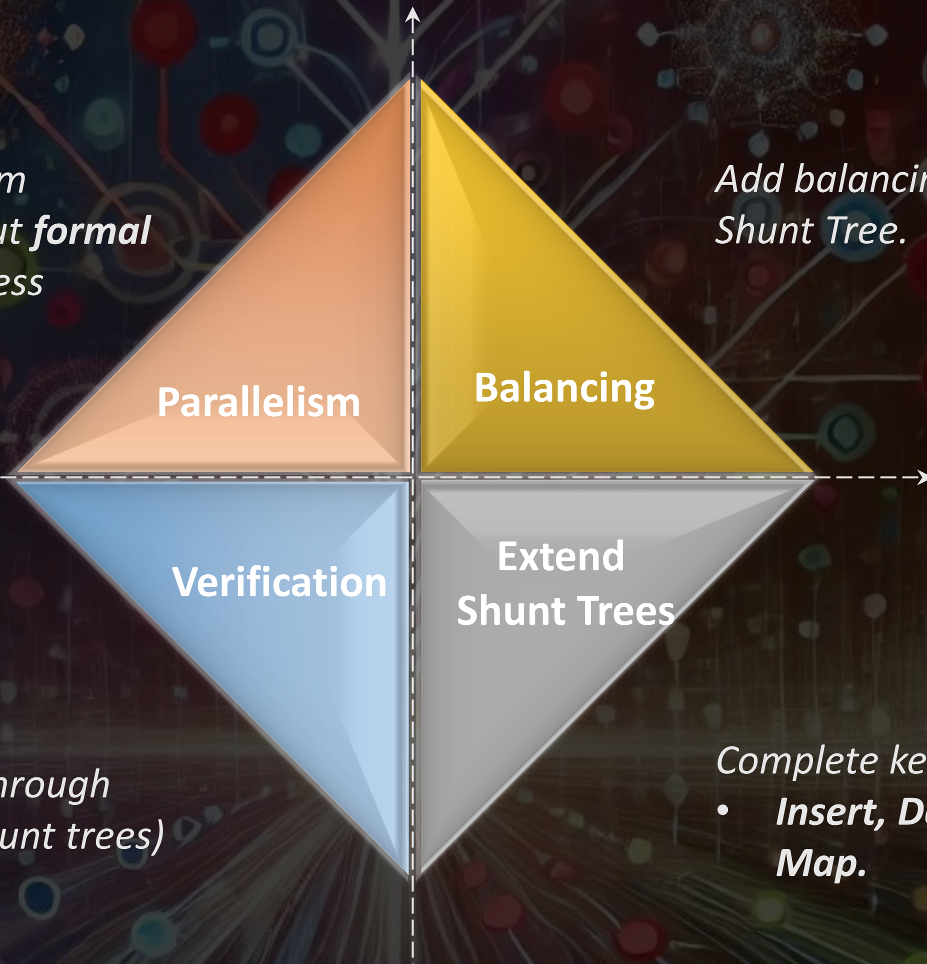
Building the core functionality for both structures and balancing

Base Structure for Shunt Trees: *Established the foundation/structure and core for further development and verification*

Remaining Work

Implement parallelism computations without ***formal*** *proving the correctness*

Add balancing mechanisms to Shunt Tree.

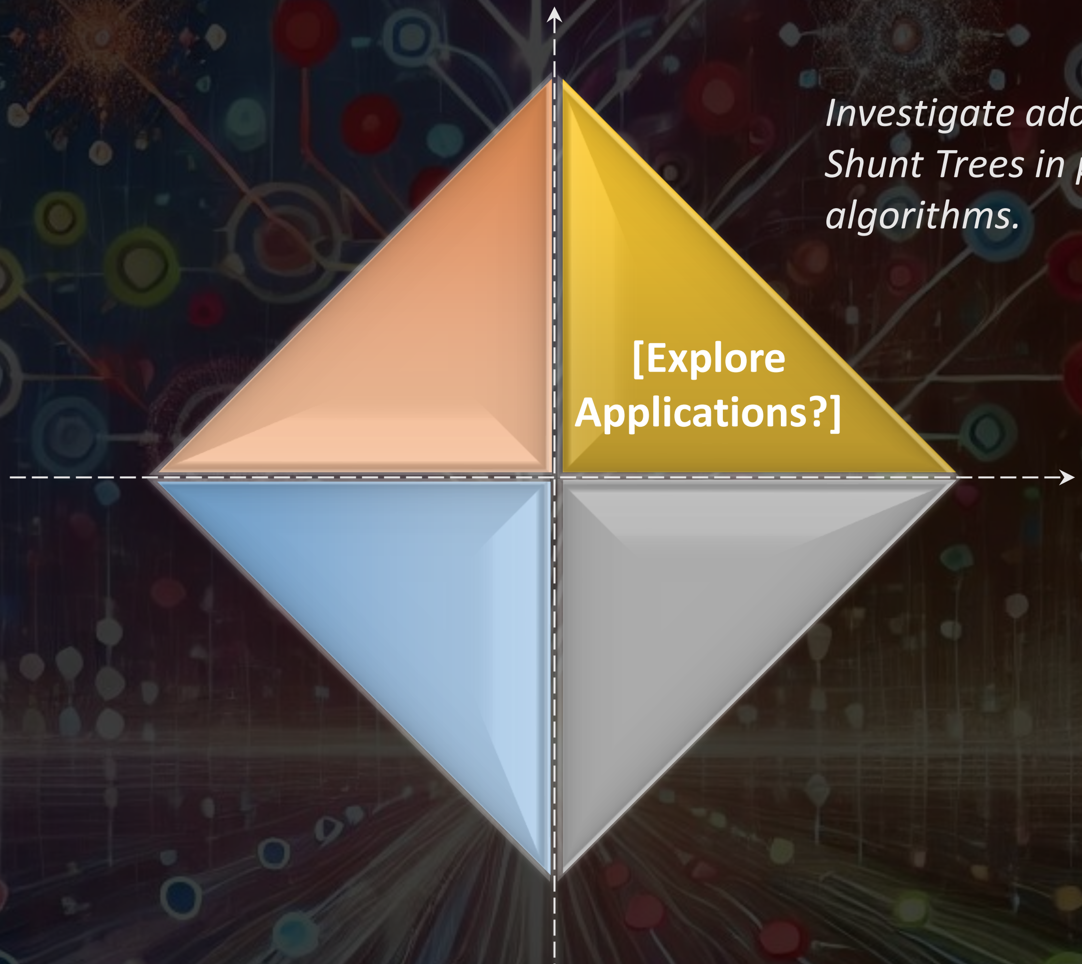


Ensure correctness through formal proofs (for Shunt trees)

Complete key operations:

- ***Insert, Delete, Zip, and Map.***

Remaining Work



Investigate additional uses of Shunt Trees in parallel algorithms.

Summary

<u>Goal:</u>
Implement balanced Shunt Trees and verify operations.

<u>Achievements:</u>

- *Basic implementation of Shunt Trees.*
- *Basic implementation of Join Lists*
- *Formal verification of Join Lists*

<u>Next Steps:</u>
Complete balancing, add operations, and finish formal verification and Add parallel operations.