

A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks

Team Members: Abeerah Aamir &
Ahmed Bin Asim

Course: Parallel and Distributed
Computing

Date: April 20, 2025



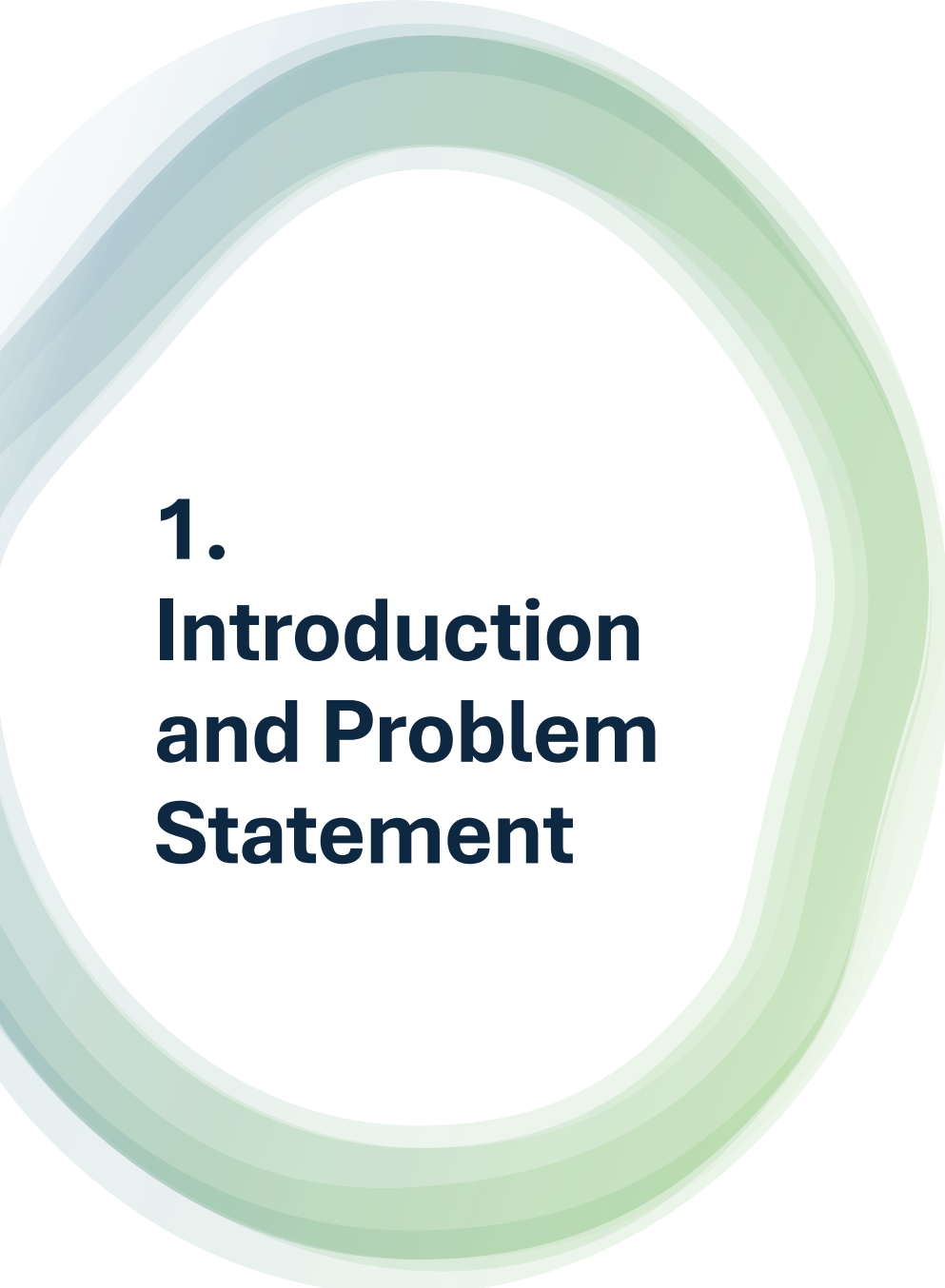
Outline

- 1. Introduction & Motivation
- 2. Background & Definitions
- 3. Problem Statement
- 4. Algorithm Description
- 5. Correctness & Performance
- 6. Parallelization Strategy
 - - MPI Implementation
 - - OpenMP Integration
 - - METIS for Graph Partitioning
- 7. Implementation Plan
- 8. Expected Results & Evaluation
- 9. Summary & Next Steps

1. Introduction and Problem Statement

- **Goal**
- • Construct $n-1$ independent spanning trees (ISTs) in a bubble-sort network B_n rooted at the same vertex
- Previous Work (Kao et al. 2019)
 - • Constructed $n-1$ ISTs on B_n
 - • **Used a recursive algorithm**
 - • Difficult to parallelize
 - • Open Problem: Develop a parallel construction algorithm
- Our Focus
 - • **Implementing and analyzing the non-recursive, parallelizable algorithm proposed in this paper**

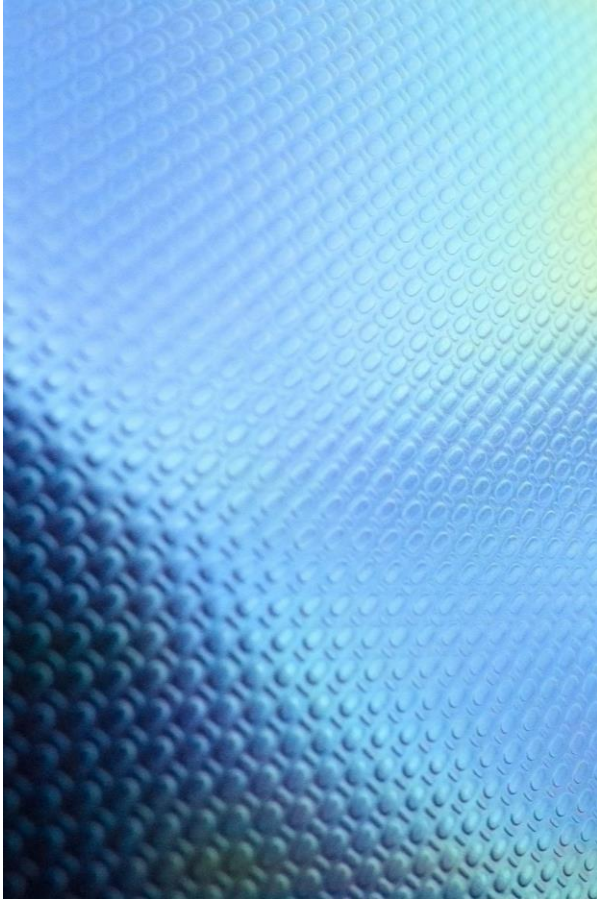




1. Introduction and Problem Statement

- **Independent Spanning Trees (ISTs)**
- Multiple spanning trees rooted at the same vertex
- Share no common edges or vertices (except root and endpoints)
- Applications:
 - - Fault-tolerant communication networks
 - - Secure message distribution
 - - Reliable broadcasting

Real-world Applications of ISTs



Network Reliability

If one path fails, alternative paths exist

Secure Messaging

Split message into multiple parts, send through independent paths

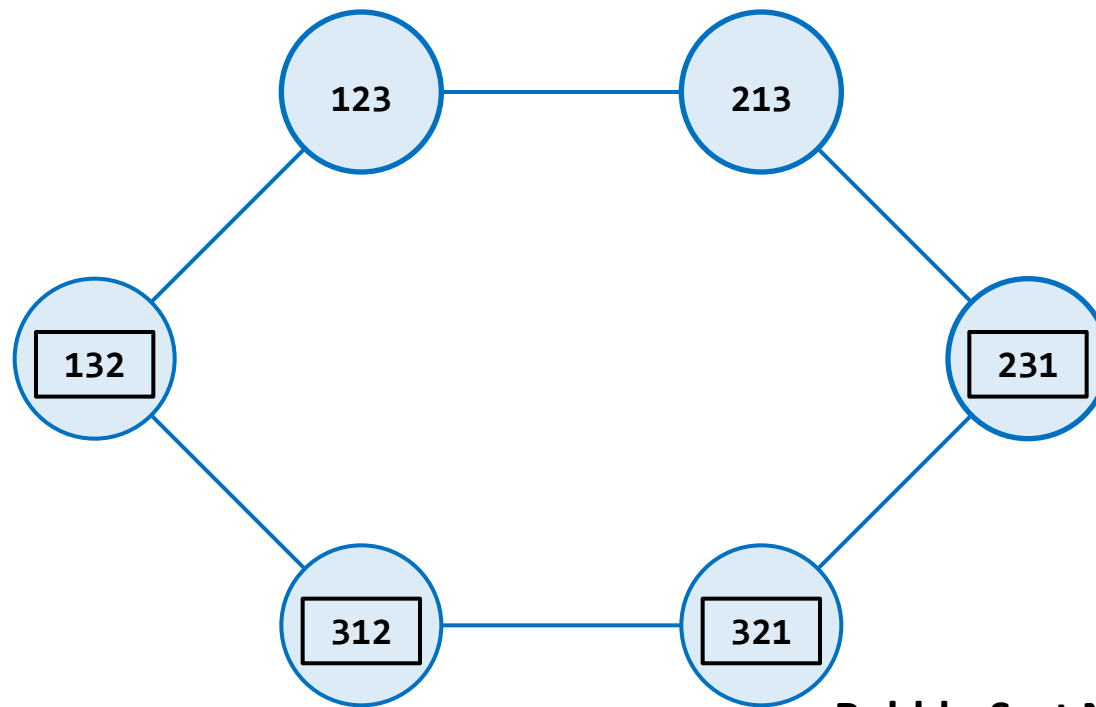
Multicast Routing

Efficient distribution of data

2. Background: Bubble-Sort Networks

- **Definition**
 - Bubble-sort network B_n is a Cayley graph based on permutation groups
 - Vertices: All permutations of $\{1, 2, \dots, n\}$ ($n!$ vertices)
 - Edges: Connect permutations that differ by a single adjacent swap
- **Properties**
 - Regular graph with degree $n-1$
 - Vertex-transitive but not edge-transitive for $n = 4$
 - Connectivity: $n-1$
 - Diameter: $n(n-1)/2$

Visualization: B3 Bubble-Sort Network

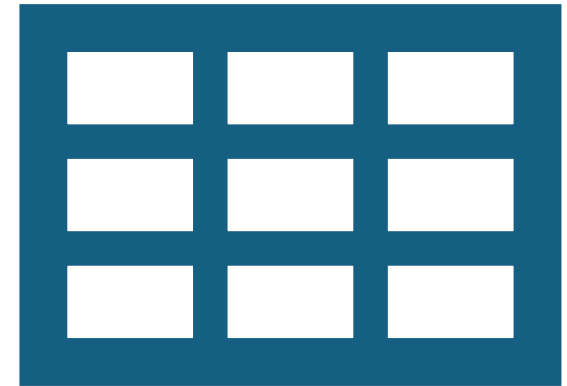


Bubble-Sort Network B3:

- 6 vertices ($3!$ permutations)
- Each edge connects permutations that differ by a single adjacent swap

4. Algorithm Description

- **Key Insight**
- • Each vertex can independently determine its parent in each spanning tree
- • No recursive calls or global state tracking needed
- **Algorithm Overview**
- 1. Choose identity permutation (12...n) as root
- 2. For each vertex v and tree T_t ($t \in \{1, 2, \dots, n-1\}$):
 - - Determine parent based on local rules
 - - Rules depend on the last symbol of v and tree index t



Algorithm Steps

Parent1(v, t, n):

Input: vertex $v = v_1 \dots v_n$, tree T_t , dimension n

Output: parent of v in T_t

if $v_n = n$ then

 if $t = n-1$ then $p = \text{FindPosition}(v)$

 else $p = \text{Swap}(v, v_{n-1})$

else

 if $v_n = n-1$ and $v_{n-1} = n$ and $\text{Swap}(v, n) \neq 1n$ then

 if $t = 1$ then $p = \text{Swap}(v, n)$

 else $p = \text{Swap}(v, t-1)$

 else

 if $v_n = t$ then $p = \text{Swap}(v, n)$

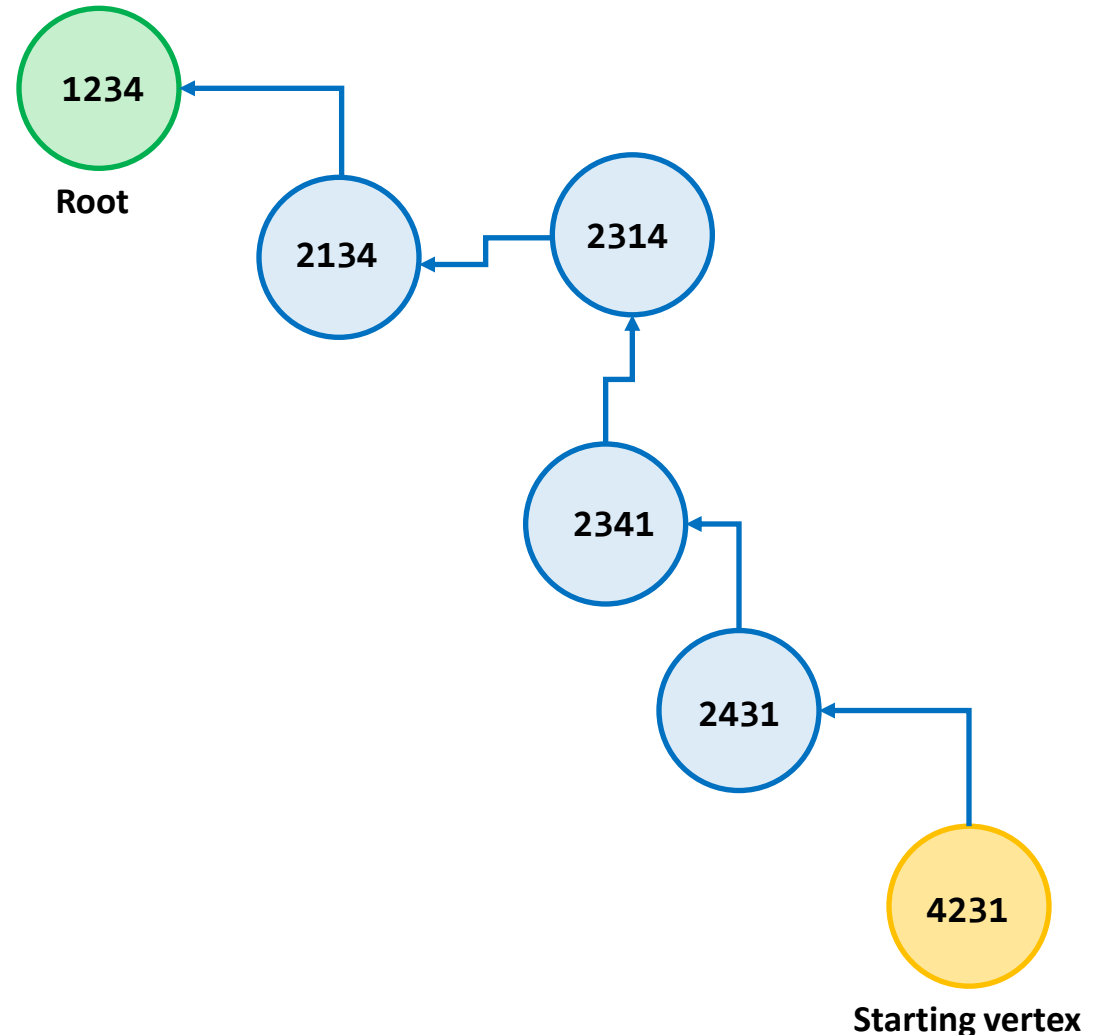
 else $p = \text{Swap}(v, t)$

return p

Algorithm Walkthrough Example

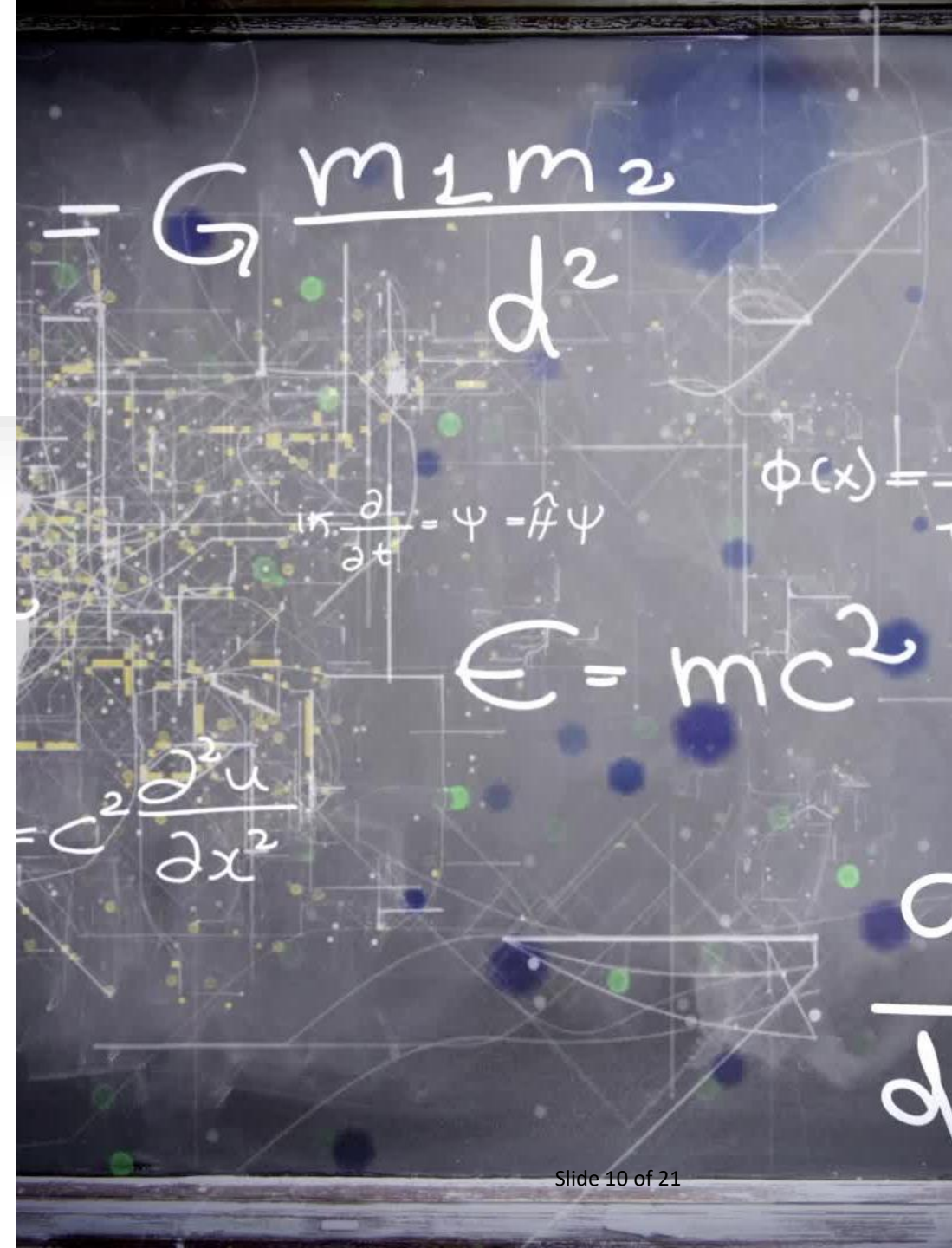
Let's trace vertex $v = 4231$ in tree T_1 :

1. $v_4 = 1$ (last symbol)
2. Since $v_4 = 1 = t$, rule (5) applies:
if $v_n = t$ then $p = \text{Swap}(v, n)$
3. $p = \text{Swap}(4231, 4) = 2431$
4. Continue applying rules:
 $2431 ? 2341 ? 2314 ? 2134 ? 1234$



5. Example Walkthrough

- **B4 Example**
- Let's trace the algorithm for vertex $v = 4231$ in tree $T1$:
 - 1. $v4 = 1$ (last symbol)
 - 2. Since $v4 = 1 = t$, rule (5) applies
 - 3. $p = \text{Swap}(v, n) = \text{Swap}(4231, 4) = 2431$
- • Continue applying rules until reaching root
- • Different trees provide different paths to root
- • Paths in different trees share no vertices except endpoints



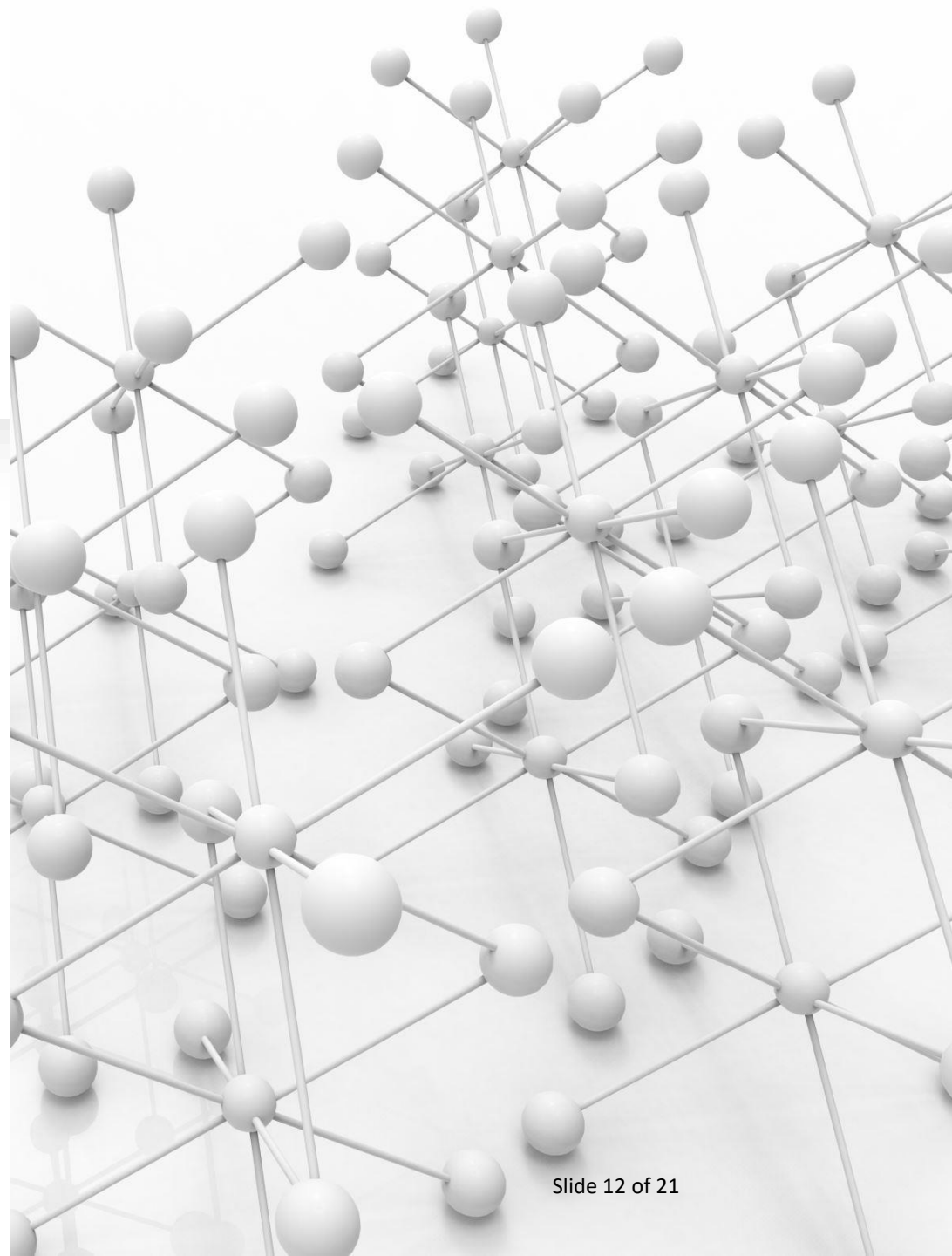


6. Correctness & Performance

- **Correctness**
 - Paper proves that for each vertex v :
 1. Algorithm constructs a unique path from v to root
 2. Paths in different trees are independent
- Time Complexity
 - **Overall complexity: $O(n \cdot n!)$**
 - Per vertex/tree: $O(1)$ - constant time parent determination
 - Proven to be asymptotically optimal
- Tree Height
 - Height = $D(B_n) + n - 1$
 - **Where $D(B_n)$ is the diameter of B_n ($n(n-1)/2$)**

7. Parallelization Strategy: MPI

- **Inter-Node Implementation**
 - Distribute vertices across MPI processes
 - Each process handles a subset of vertices
 - No communication needed during tree construction
- **Vertex Distribution**
 - Static approach: Divide $n!$ vertices evenly
 - Alternative: Use METIS to partition considering tree structure



```
// Pseudocode for MPI process
process_rank = MPI_Comm_rank()
total_processes = MPI_Comm_size()

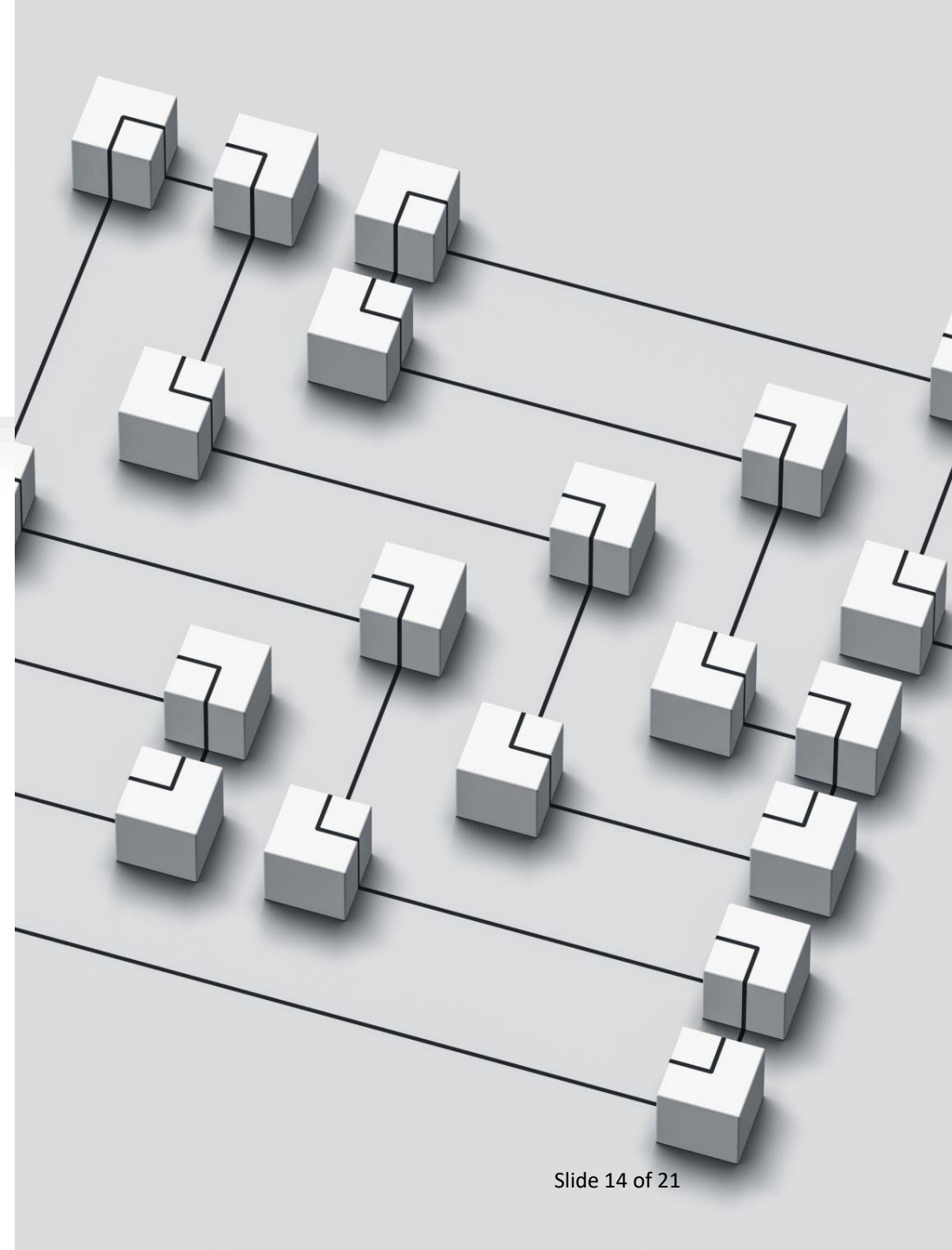
// Determine local vertex set
local_vertices = assign_vertices(process_rank, total_processes)

// Process each local vertex
for each vertex v in local_vertices:
    for t = 1 to n-1:
        parent[v][t] = Parent1(v, t, n)
```

MPI Process Structure

8. Parallelization Strategy: OpenMP

- **Intra-Node Parallelism**
 - Use OpenMP to parallelize tree construction within each MPI process
 - Each thread handles multiple trees for a subset of vertices
- **Work Distribution**
 - Optimal thread count depends on n and vertex count
 - Balance between parallelism and overhead



```
// Within each MPI process
#pragma omp parallel for collapse(2)
for (int i = 0; i < local_vertex_count; i++) {
    for (int t = 1; t < n; t++) {
        Vertex v = local_vertices[i];
        parent[v][t] = Parent1(v, t, n);
    }
}
```

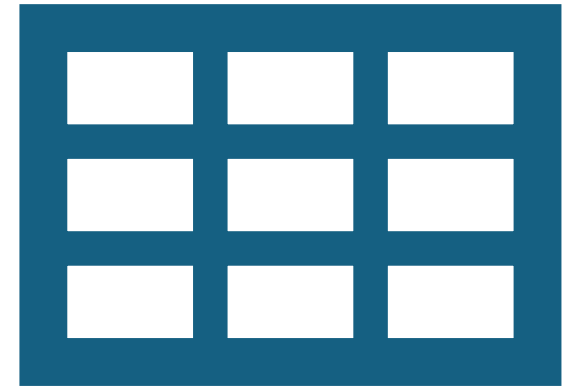
OpenMP Implementation

9. Graph Partitioning with METIS

- **METIS Integration**
 - Use METIS to partition the bubble-sort network
 - Goal: Minimize communication, balance load
- Partitioning Approach
 - **1. Construct adjacency graph for B_n**
 - 2. Apply METIS k-way partitioning
 - 3. Assign partitions to MPI processes
- Benefits
 - **Improved locality**
 - Reduced communication overhead
 - Better load balancing

10. Implementation Plan

- **Data Structures**
 - Vertex representation: Integer arrays for permutations
 - Tree storage: Adjacency lists or parent pointers
 - Graph partitioning: CSR format for METIS
- **Implementation Phases**
 1. Sequential implementation & verification
 2. MPI parallelization
 3. OpenMP integration
 4. METIS-based partitioning
 5. Performance optimization



11. Expected Results & Evaluation

- **Performance Metrics**
 - Strong scaling: Speedup with fixed problem size, increasing processors
 - Weak scaling: Efficiency with increasing problem and processor count
 - Efficiency: Comparison to theoretical optimal performance
- Experimental Setup
 - **Test on different dimensions: B4, B5, B6, B7**
 - Compare: Sequential vs. MPI vs. MPI+OpenMP
 - Measure impact of METIS partitioning
- Evaluation Goals
 1. Demonstrate algorithmic correctness
 2. **Show near-linear scaling with processors**
 3. Quantify benefit of hybrid parallelization





12. Summary & Next Steps

- **Key Contributions**

- • Implementing a fully parallelizable algorithm for IST construction
- • Demonstrating efficient hybrid MPI+OpenMP implementation
- • Analyzing scaling behavior with different network sizes

- **Next Steps**

- • Complete implementation (Phase 2)
- • Conduct performance experiments
- • Analyze results and optimize
- • Prepare final demonstration

References

1. Kao, S. S., Klasing, R., Hung, L. J., Lee, C. W., & Hsieh, S. Y. (2023). A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks.

2. Kao, S. S., Pai, K. J., Hsieh, S. Y., Wu, R. Y., & Chang, J. M. (2019). Amortized efficiency of constructing multiple independent spanning trees on bubble-sort networks. *Journal of Combinatorial Optimization*, 38(3), 972-986.

3. Akers, S. B., & Krishnamurty, B. (1989). A group theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4), 555-566.



Thank You!

Questions?