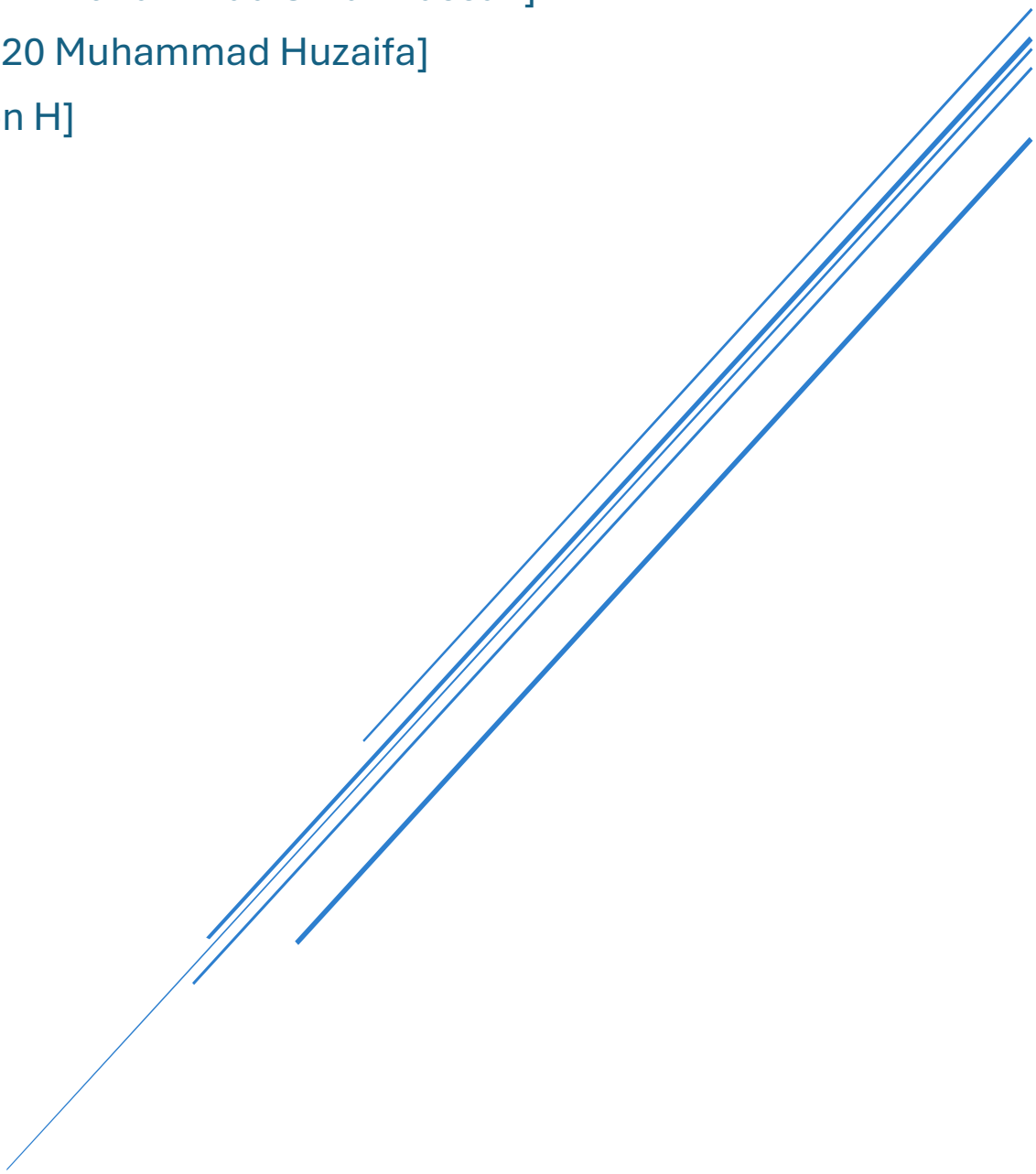
# MISTS IN BUBBLE SORT NETWORKS

[22i-0941 Muhammad Sarmad]
[22i-0942 Muhammad Umar Hassan]
[22i-1220 Muhammad Huzaifa]
[Section H]



FAST NUCES ISLAMABAD
PDC Project

# Contents

# Multiple Independent Spanning Trees (MISTs) Construction In Bubble Sort Networks


## Abstract

This report presents the design and implementation of Multiple Independent Spanning Tree (IST) construction on the bubble-sort network **Bn**, comparing a serial C++ version against a hybrid MPI+OpenMP parallel version. We describe the algorithmic structure, data layout, and communication strategy, and highlight the modifications required to scale the computation across multiple processes and threads.


---

## 1. Introduction

The bubble-sort network **Bn** is a well-known interconnection topology whose vertices correspond to all **n!** permutations of **{1,…,n}** and edges connect permutations that differ by a single adjacent swap. Constructing **n−1** independent spanning trees (ISTs) on **Bn** has applications in fault-tolerant routing and multipath communication. We first implement a straightforward serial algorithm in C++, and then extend it into a hybrid parallel version using MPI for inter-process distribution of work and OpenMP for intra-node threading.

## 2. Serial Implementation

**Complexity :** O( n * n!), where n is the dimension of the bubble sort network

i. **Permutation Generation**
   - All **n!** permutations are generated **via std::next_permutation**.

ii. **Preprocessing**
   - For each vertex **v**, compute:
     - pos[v][symbol]: the position of each symbol in the permutation.
     - firstWrong[v]: rightmost index where the permutation differs from the identity.

iii. **Parent Selection**
   - For each tree index **t∈[1..n−1]**] and each vertex v≠root, compute its parent parent1(v, t, n) by analyzing the last two symbols and applying swap rules ensuring independence across trees.

iv. **Tree Construction**

- Build adjacency lists children[t][p] by mapping each non-root vertex to its parent in tree T.

v. **Output**

- Optionally export each tree to a DOT file for visualization.

The serial program measures wall-clock time via clock() and reports total construction time.

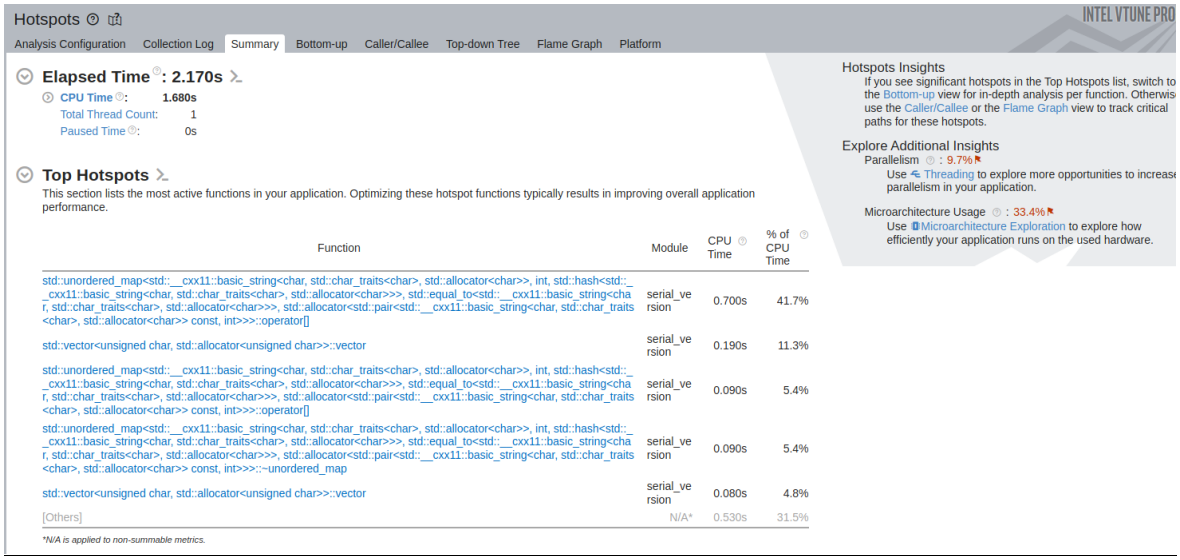
## 3. Parallel Implementation

**Complexity :** O( n * n!), where n is the dimension of the bubble sort network

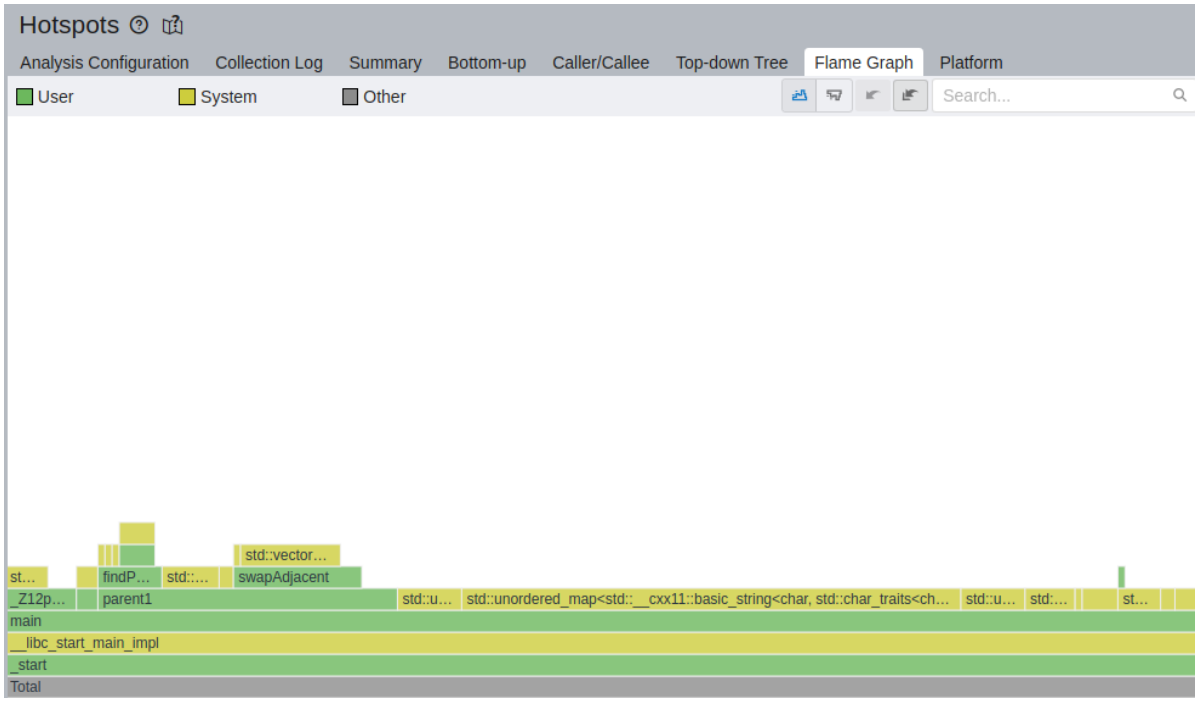
1. **MPI Initialization**
   - Master process gets the dimension **n** from the user and communicates to each slave process.
2. **Work Distribution**
   - The **n−1 trees** are partitioned evenly among size ranks (with remainder handled by first rem ranks).
   - Each MPI process works on its assigned subset of tree indices assigned_t.
3. **Local Computation with OpenMP**
   - Within each MPI rank, **OpenMP** is used to to parallelize the contruction of spanning trees. For loop iterates over all assigned (t,v) pairs, computes each vertex's parent, and accumulates edges into a thread-local buffer, then merges via a critical section.
4. **Communication**
   - Non-root ranks send, per tree t, the count and list of (**parent,child) pairs** to rank 0 using MPI_Send.
   - Rank 0 receives from each rank and assembles the global children_global adjacency lists.
5. **Finalization**
   - Rank 0 writes DOT files for all n-1 trees.

- All ranks synchronize with MPI_Barrier; the maximum elapsed time across ranks is reduced to rank 0 and reported.
- Finally, **convert.sh** script is used to convert Dot files to PDF format.

## 4. Performance Analysis:




**(Hotspot Analysis of Serial Version)**

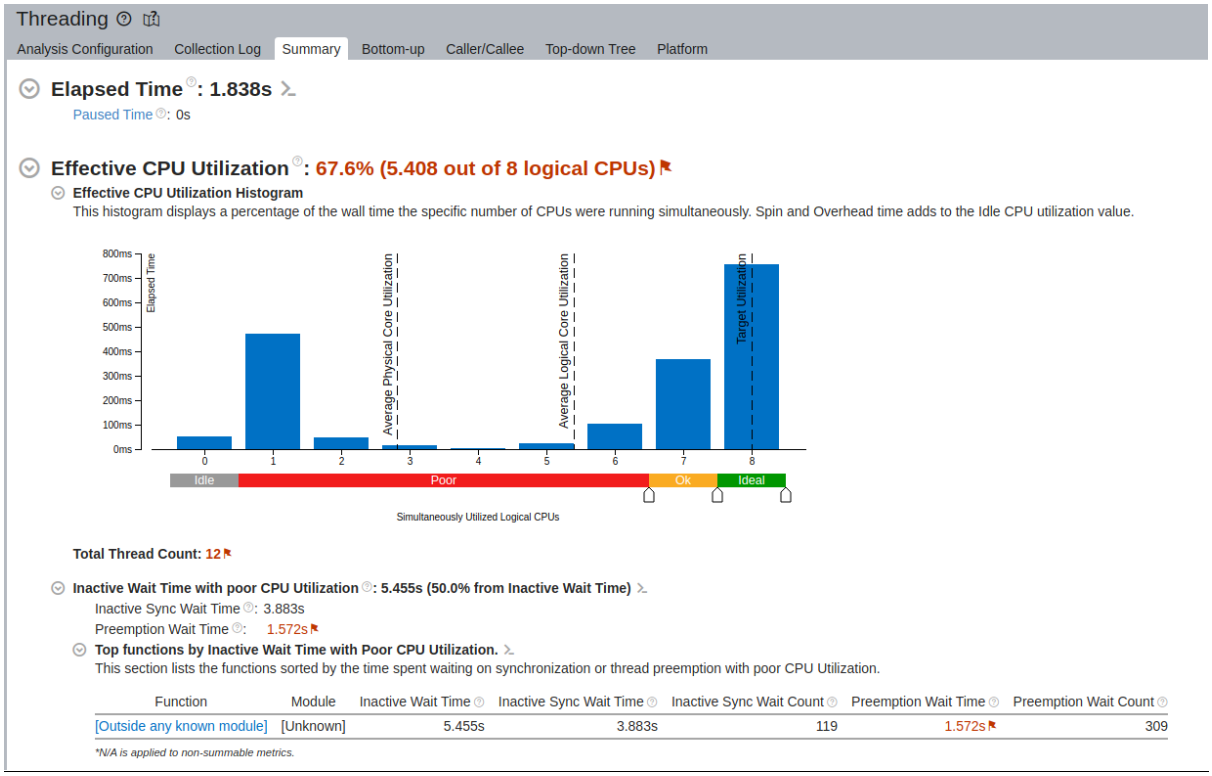



**(Flame Graph of Hotspots)**

Hotspot Analysis shows following **hotspots** in the serial version:

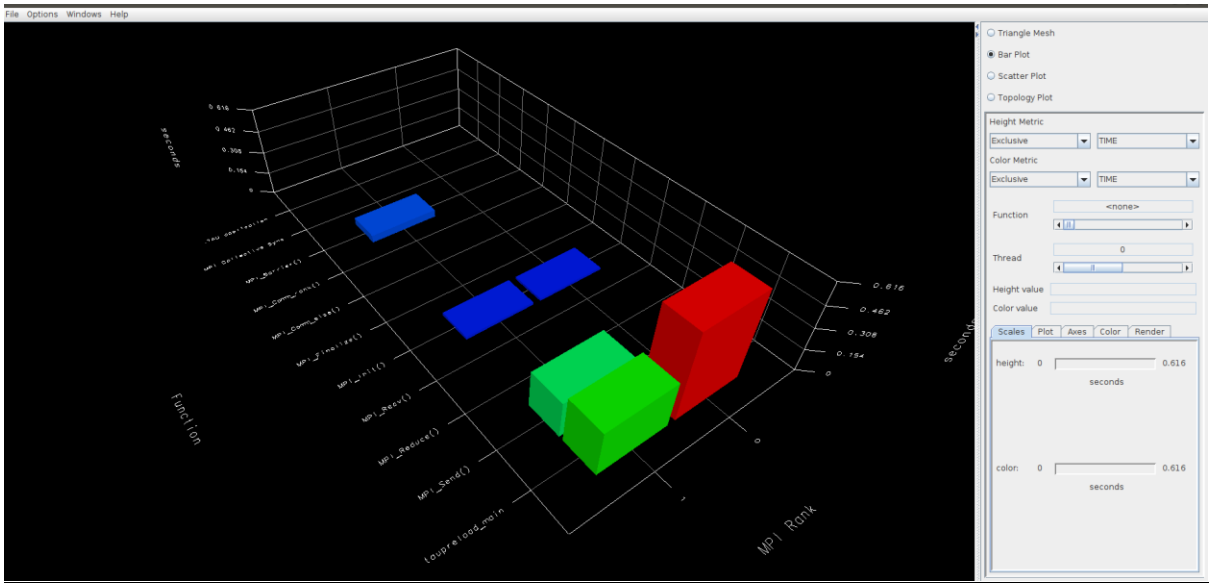
- Parent() function
- swapAdjacent() function
- hash table lookup

These operations are parallelized in the parallel version for intranode performance improvement using OpenMP by parallelizing the parent finding loop for the vertices




**(Performance analysis of the parallel version)**

Parallel version takes full advantage of inter-node **(MPICH)** and intra-node **(OpenMP)** parallelism to compute the spanning trees.




**(MPICH Analysis using TAU profiler)**

We have tested both versions using different dimensions of the bubble sort network **Bn.** Following table compares the performance of the serial and parallel versions **(using 2 MPI processes having 2 threads per process)**:

| Dimension(n) | Serial Version | Parallel Version |
|---|---|---|
| **8** | 0.128s | 0.075s |
| **9** | 1.398s | 0.712s |
| **10** | 16.667s | 8.165s |

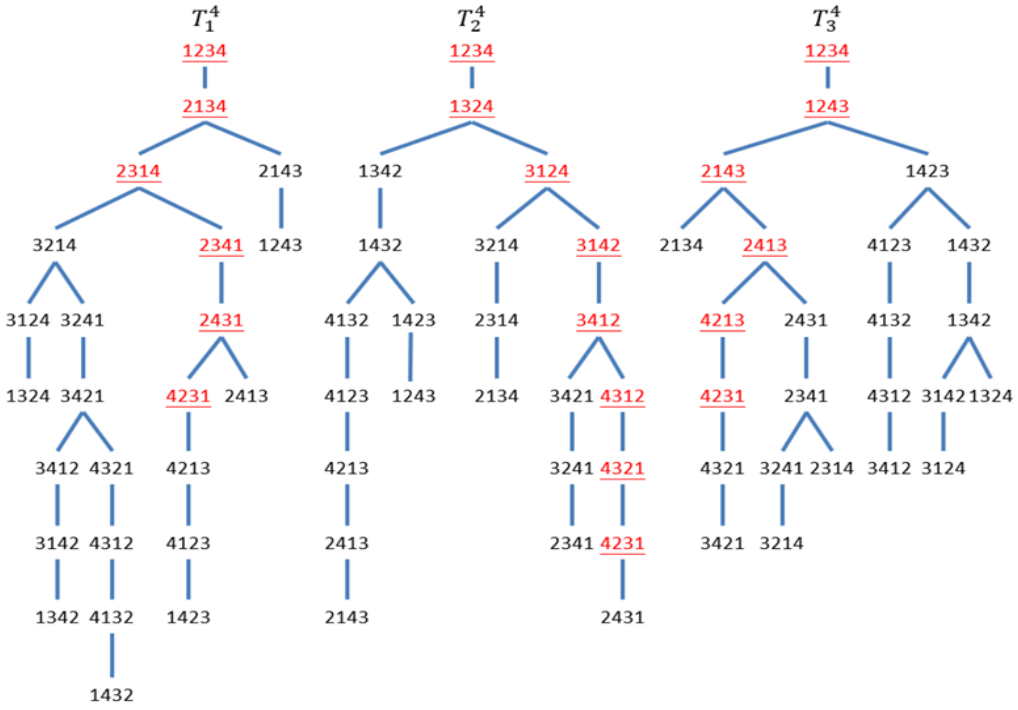
The parallel version gives **2x speed boost** by using **2 MPI Processes** and **2** threads per process. The less speedup as compared to threads is due to the communication time which happens between the master and other processes for trees sharing.

This communication couldn't be overlapped with computation as it added to the memory usage which is already very high because of the huge number of vertices and trees that we have to keep in memory.

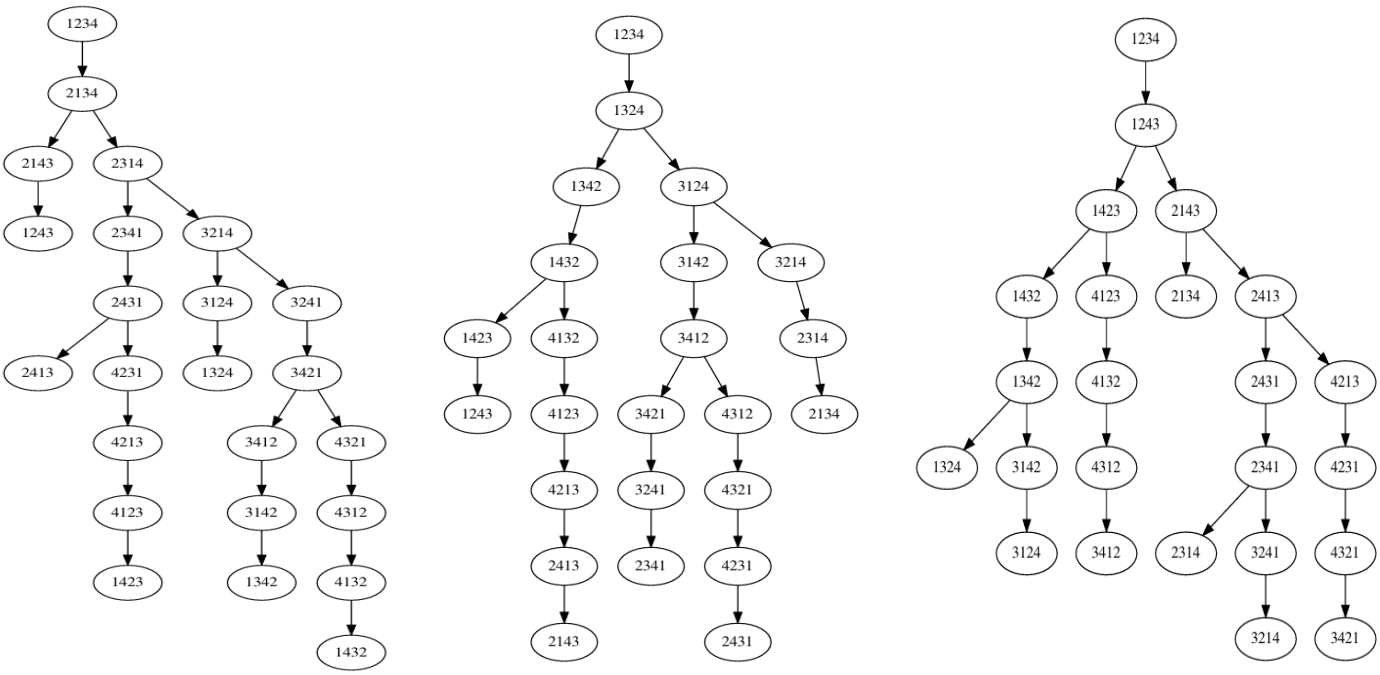
**NOTE :** We have added the non-blocking communication code alongside the submission.

## 5. Correctness:

Given implementations have been tested and compared with the results of the research paper for the bubble sort networks of dimension 3 and 4. These results perfectly align with the results fond in the research paper:




**(MISTs for n=4 from the research paper)**




**(MISTs for n=4 by the given algorithm)**

## 6. Conclusion

We have demonstrated that a straightforward serial algorithm for constructing ISTs on **Bn** can be effectively parallelized with **MPI+OpenMP**. The hybrid approach distributes the tree indices across processes and exploits thread-level parallelism for inner loops, leading to substantial speedups on multi-node, multi-core platforms.

## References:

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