

Final Report

CS3006 Parallel and Distributed Computing

Semester Project

Muhammad Ali Naveed  
Hamza Omer  
Ariyan Chaudhary

Section: J

*Submitted to: Sir. Fahad Shafique*

Department of Computer Science BS(CS)

FAST-NUCES Islamabad

Title

A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks

Introduction

We need to parallelize the construction of independent spanning trees (ISTs) in bubble sort networks so that the ISTs can be generated more efficiently. The existing algorithms make use of a recursive solution to the construction of the ISTs; however, recursion is not something that can be parallelized effectively, which is why there is a need for an iterative algorithm for the construction of ISTs. This algorithm is proposed in the Journal of Parallel and Distributed Computing (Kao et al., 2023) and this document contains the necessary information regarding its practical implementation.

Problem Questions

**Q1 – Why use Multiple Independent Spanning Trees (ISTs) for Data Broadcasting?**  
They provide different ways (trees) to send data so that if one path fails, another can still deliver the message. This increases fault tolerance (the network keeps working even if parts fail) and helps with security (messages can take different routes, making eavesdropping harder).

**Q2 – What is an Inverse Permutation?**  
A permutation is just a way to rearrange a list of things. Its “inverse” tells you how to undo that rearrangement. For example, if a permutation sends position 1 to position 3 and position 3 to position 5, the inverse will say “look at position 3, it came from position 1,” etc. Formally, if π is a permutation, then its inverse π−1 is the rule that sends each element back to its original position.

**Q3 – Importance of ISTs (Disjoint Paths and Reliability)**  
ISTs ensure multiple independent routes exist in a network. Because these routes don’t overlap, one route getting blocked or failing won’t stop other routes from working. This keeps the network up and running (high reliability).

**Q4 – High-Level Overview of the Non-Recursive “Parent Determination” Algorithm**

Goal: Each vertex quickly figures out its “parent” in every spanning tree (in constant time).

Preprocessing:

1. Compute the inverse permutation: a map that tells you how to revert to the original order.
2. Identify elements (vertices) that aren’t where they should be, so we know which positions need “fixing.”

Core Functions:

1. Swap(v, x): Swaps two elements in the permutation to gradually move them to their correct spot.
2. FindPosition(v): Looks at the current arrangement to decide the next vertex to swap or correct.
3. Parallelization: All vertices can do these checks and swaps at the same time, allowing fast computation of parents.

Example of IST for n = 3:

A diagram of a hexagon with numbers and a hexagon with numbers

AI-generated content may be incorrect.

Key Contributions

1. **Fully Parallelizable, Non‑Recursive IST Construction**  
   The paper introduces the first non‑recursive algorithm for constructing n−1 independent spanning trees (ISTs) in the bubble‑sort network Bn​. Unlike prior recursive methods, this algorithm allows every vertex to determine its parent in each spanning tree in constant time, enabling true parallel execution across all vertices ​.
2. **Asymptotically Optimal Time Complexity**  
   By processing all n! vertices in parallel, each in O(1) time, the total running time of the algorithm is O(n⋅n!). This matches the lower bound of Ω(n⋅n!), proving the approach is asymptotically optimal for Bn.
3. **Bounded Tree Height**  
   The constructed ISTs have height at most D(Bn)+(n−1) , where D(Bn)=n(n−1)/2 is the network diameter. This guarantee ensures the spanning trees remain structurally efficient, limiting the worst-case path length from any vertex to the root ​.
4. **Resolution of an Open Problem**  
   By eliminating recursion and achieving full parallelization, the algorithm settles the open question posed by Kao et al. (2019) regarding whether ISTs in bubble‑sort networks can be built in parallel ​.

Parallelization Strategy

To implement the algorithm efficiently on modern high-performance computing platforms, we propose the following three-layered parallelization approach:

**1. Inter-Node Parallelism with MPI**

* Domain Decomposition: Partition the set of all n! permutation‑vertices among MPI processes. Each process is responsible for computing the IST‑parents for its assigned subset of vertices.
* Communication Pattern: Since each vertex’s parent is determined purely by its permutation and a small number of local values (its inverse permutation and the position of the first out-of-place symbol), *no* inter-process communication is needed *during* the parent‑computation phase. After local computation, processes exchange only summary or verification messages (e.g., counts of edges) to validate global properties.
* Load Balancing: A uniform initial distribution of vertices ensures balanced computation. Minor imbalances can be further alleviated by dynamic MPI work‑stealing if needed.

**2. Intra-Node Parallelism with OpenMP / OpenCL**

* Multi-Core CPUs (OpenMP): Within each MPI rank, spawn an OpenMP team of threads. Use a simple #pragma omp parallel for loop over the local vertex list so that each thread independently executes the constant-time parent function (Algorithm 1) for its assigned vertices.
* Many‑Core Accelerators (OpenCL): For GPUs or other accelerators, express the parent‑computation as an OpenCL kernel where each work‑item handles one vertex. The kernel body implements the same Parent1(v, t, n) logic; because there are no cross‑vertex data dependencies, the GPU can execute millions of these kernels concurrently with minimal synchronization overhead.

**3. Graph Partitioning with METIS**

* Objective: Even though the core parent determination is completely local, initial graph‑partitioning can reduce memory contention and improve cache locality.
* Approach: Model the bubble‑sort network Bn as a graph in METIS, where each permutation is a vertex and edges represent adjacent swaps. Use METIS to partition this graph into PPP roughly equal parts (where PPP is the number of MPI processes), minimizing the number of edges *crossing* partitions.
* Benefit: By assigning each MPI rank a subgraph with few external edges, we improve data locality for any auxiliary operations (e.g., building local adjacency lists, performing post‑processing checks) and reduce the volume of any necessary inter‑node communication.

Summary  
This three‑tiered parallelization plan—MPI for distributing vertices, OpenMP/OpenCL for intra‑node speedup, and METIS for optimizing data locality—fully exploits modern HPC architectures. It maps directly onto the constant‑time, per‑vertex parent computation of the proposed IST algorithm, delivering scalable, efficient construction of multiple independent spanning trees in bubble‑sort networks.

# References

Kao, S., Klasing, R., Hung, L., Lee, C., & Hsieh, S. (2023). A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks. *Journal of Parallel and Distributed Computing*, *181*, 104731. https://doi.org/10.1016/j.jpdc.2023.104731

S.-S. Kao, K.-J. Pai, S.-Y. Hsieh, R.-Y. Wu, and J.-M. Chang, “Amortized efficiency of constructing multiple independent spanning trees on bubble-sort networks,” Journal of Combinatorial Optimization, vol. 38, no. 3, pp. 972–986, 2019.