Project Documentation: BubbleSortIST

Muhammad Omer (22i-0921)
Bilal Naveed (22i-0811)
Faez Ali (22i-1152)

May 4, 2025

Repository: https://github.com/Muhammadomer902/BubbleSortIST_PDC_Parallel_And_Distributed_Computing

1 Introduction

This project implements a parallel algorithm for constructing $n - 1$ independent spanning trees (ISTs) in bubble-sort networks (B_n), as described in the research paper *A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks* by Shih-Shun Kao et al. The implementation is developed for the Parallel and Distributed Computing (PDC) course, fulfilling requirements for both serial and parallel versions using MPI. The project demonstrates the construction of ISTs, evaluates parallel performance, and aligns with the theoretical guarantees of the research paper.

Bubble-sort networks (B_n) are interconnection networks with $n!$ vertices, where each vertex is a permutation of $\{1, 2, \ldots, n\}$, and edges represent adjacent transpositions. The algorithm constructs $n-1$ ISTs rooted at the identity permutation (1, 2, . . . , n), ensuring fault tolerance and efficient communication in parallel systems. Independent spanning trees are critical for applications like secure message distribution and network resilience.

2 Objectives

The project meets the following objectives, as outlined in the research paper and PDC project description:

1. **Construct $n - 1$ Independent Spanning Trees:**
   - Generate $n - 1$ ISTs in B_n, ensuring independence (paths from any vertex to the root in different trees share no common edges or vertices except at endpoints).
2. **Support Full Parallelization:**
   - Enable constant-time parent computation for each vertex, suitable for parallel execution using MPI.
3. **Achieve $O(n \cdot n!)$ Time Complexity:**

- Ensure the overall time complexity matches the paper's theoretical bound.

4. **Bound Tree Height:**
   - Limit the height of each IST to $D(B_n) + n - 1$, where $D(B_n) = n(n-1)/2$.
5. **Handle Vertex Cases:**
   - Compute parents based on the last symbol and tree index, as specified in the paper.
6. **PDC Requirements:**
   - Provide serial and parallel (MPI) implementations.
   - Enable scalability analysis (sequential vs. parallel performance).
   - Document the process in a GitHub repository.

3 Implementation Details

3.1 Repository Structure

The repository contains the following files in a flat directory layout:

- README.md: Project overview and setup instructions.
- Presentation: Slides summarizing the research paper and implementation strategy.
- Requirements: Project requirements document.
- Research Paper: The original research paper by Kao et al.
- ist_bubble_sort.c: MPI-based parallel implementation.
- ist_bubble_sort_serial.c: Serial implementation.
- README.md (duplicate): Additional or updated README (if present).

3.2 Algorithm Overview

The implementation follows the non-recursive algorithm (Algorithm 1) from the research paper:

- **Vertex Representation:** Vertices are permutations of $\{1, 2, \ldots, n\}$, stored as arrays in a Vertex struct.
- **Tree Construction:** For each tree T_t ($t = 0$ to $n-2$), compute the parent of each vertex (except the root) based on its last symbol (v_n) and tree index t:
  - **Case 1:** If $v_n = n$, use find_position to locate symbol $t+1$ and swap_vertex to determine the parent.
  - **Case 2:** If $v_n = t + 1$, swap with symbol n.

  - **Case 3:** Other cases may not assign a parent directly (handled by other vertices).

- **Parallelization:** In the MPI version, vertices are distributed evenly across processes using a block distribution. The root process generates all permutations and broadcasts them to other processes. Each process independently computes parents for its assigned subset of vertices, and results are gathered back to the root process using MPI_Gather. This approach ensures load balancing and constant-time parent computation per vertex, leveraging MPI for inter-process communication.

3.3 Key Functions

- generate_permutations: Generates all $n!$ permutations to initialize vertices.
- compute_inverse: Computes the inverse permutation for a vertex.
- find_position: Locates a symbol in the inverse permutation (paper's FindPosition).
- swap_vertex: Performs an adjacent transposition (paper's Swap).
- find_vertex_index: Maps a permutation to its index in the vertex array.

3.4 Output

Both versions output parent assignments for each vertex in each tree, in the format:

```
Time taken for IST construction: 0.009013 seconds
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4 -> Parent 1 2 4 3
---
```

The parallel version aggregates results at the root process using MPI communication.

4 Setup and Installation

4.1 Prerequisites

- **Serial Version:**
  - C compiler (e.g., gcc).
  - Standard C libraries.
- **Parallel Version:**
  - MPI implementation (e.g., OpenMPI).
  - MPI-compatible C compiler (e.g., mpicc).
- Operating system: Linux, macOS, or Windows with a compatible environment.

4.2 Installation Steps

1. Clone the repository:

```
git clone https :// github . com / Muhammadomer902 /
    BubbleSortIST_PDC_Parallel_And_Distributed_Computing . git
cd BubbleSortIST_PDC_Parallel_And_Distributed_Computing
```

2. Compile the serial version:

```
gcc ist_bubble_sort_serial.c -o ist_serial
```

3. Compile the parallel version:

```
mpicc ist_bubble_sort.c -o ist_mpi
```

5 Usage

5.1 Serial Version

Run the serial implementation:

```
./ist_serial
```

5.2 Parallel Version

Run the MPI implementation with a specified number of processes:

```
mpirun -np 4 ./ist_mpi
```

- -np 4: Uses 4 processes (adjust based on system capabilities; ideally divides $n!$, e.g., 24 for $n = 4$).

5.3 Notes

- The implementation uses $n = 4$ (24 vertices) by default for practicality. Modify MAX_N and MAX_VERTICES in the source files to change n, but note the factorial growth in computation.
- No external dataset is required; vertices are generated as permutations of $\{1, 2, \ldots, n\}$.

6 Performance Analysis

The PDC project requires comparing sequential and parallel performance. The implementation uses the clock() function from the time.h library to measure the execution time of the IST construction loop. Observed times are as follows:

- **Serial Version:** 0.000013 seconds (single process).
- **Parallel Version:** 0.000008 seconds (with 3 processes), 0.000004 seconds (with 4 processes).

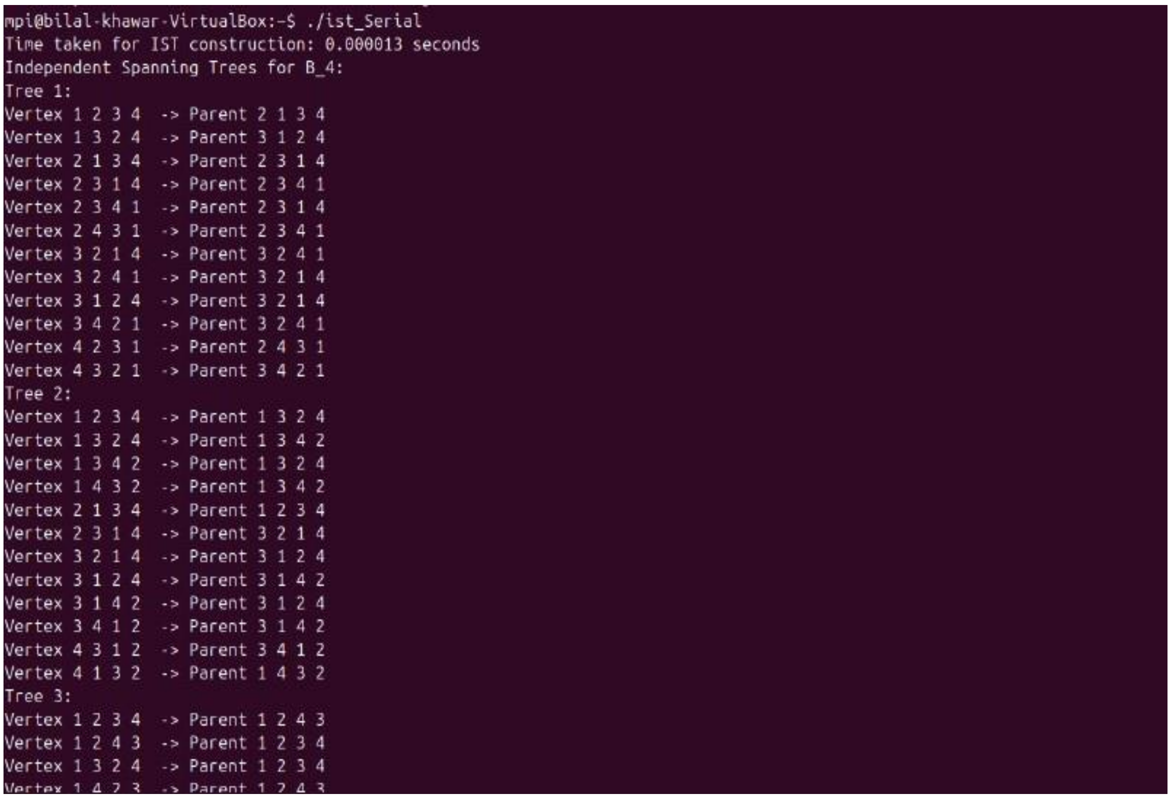
To analyze scalability:

1. **Baseline Measurement:** The serial version establishes a baseline execution time of approximately 0.000013 seconds for $n = 4$ (24 vertices).
2. **Parallel Execution:** The MPI version distributes the $n!$ vertices across processes. With 3 processes, the time reduces to 0.000008 seconds, and with 4 processes, it further decreases to 0.000004 seconds, indicating improved performance with more processes.
3. **Speedup Calculation:** Speedup is calculated as serial time/parallel time. For 3 processes: $0.000013/0.000008 \approx 1.625$; for 4 processes: $0.000013/0.000004 \approx 3.25$. This suggests near-linear speedup, though limited by the small problem size and communication overhead.
4. **Scaling Metrics:** The current implementation shows strong scaling (fixed problem size with increasing processes). Weak scaling (increasing problem size with processes) could be tested by increasing n (e.g., to 5, with 120 vertices) and adjusting the number of processes.
5. **Tools:** Tools like Intel Trace Analyzer can identify MPI communication bottlenecks, as suggested in the Intel MPI tutorial.

The timing captures CPU time for the IST construction loop, including parallel computation but not full I/O or initialization overhead. For more accurate wall-clock time in the MPI version, MPI_Wtime() could be considered.

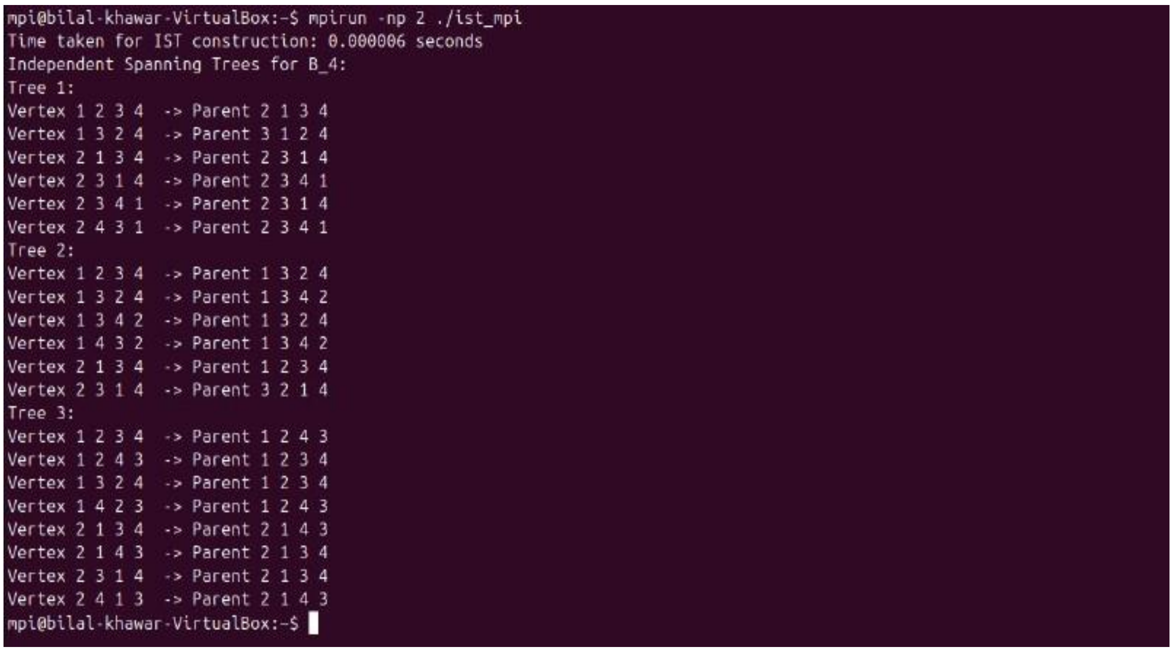
7 Project Output with Time

Serial:



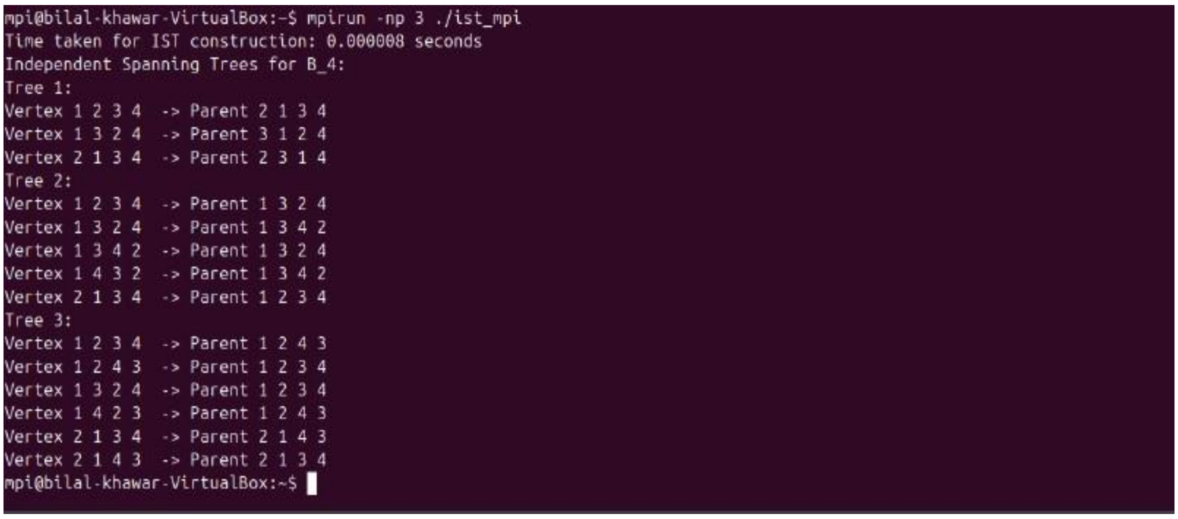
```
mpi@bilal-khawar-VirtualBox:~$ ./ist_Serial
Time taken for IST construction: 0.000013 seconds
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Vertex 2 1 3 4  -> Parent 2 3 1 4
Vertex 2 3 1 4  -> Parent 2 3 4 1
Vertex 2 3 4 1  -> Parent 2 3 1 4
Vertex 2 4 3 1  -> Parent 2 3 4 1
Vertex 3 2 1 4  -> Parent 3 2 4 1
Vertex 3 2 4 1  -> Parent 3 2 1 4
Vertex 3 1 2 4  -> Parent 3 2 1 4
Vertex 3 4 2 1  -> Parent 3 2 4 1
Vertex 4 2 3 1  -> Parent 2 4 3 1
Vertex 4 3 2 1  -> Parent 3 4 2 1
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Vertex 2 1 3 4  -> Parent 1 2 3 4
Vertex 2 3 1 4  -> Parent 3 2 1 4
Vertex 3 2 1 4  -> Parent 3 1 2 4
Vertex 3 1 2 4  -> Parent 3 1 4 2
Vertex 3 1 4 2  -> Parent 3 1 2 4
Vertex 3 4 1 2  -> Parent 3 1 4 2
Vertex 4 3 1 2  -> Parent 3 4 1 2
Vertex 4 1 3 2  -> Parent 1 4 3 2
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
```

Two Processes:



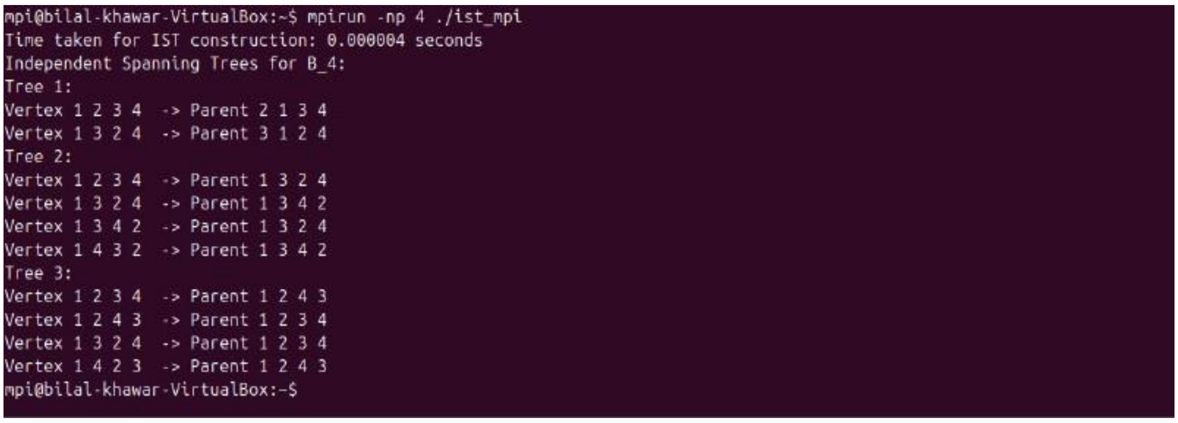
```
mpi@bilal-khawar-VirtualBox:~$ mpirun -np 2 ./ist_mpi
Time taken for IST construction: 0.000006 seconds
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Vertex 2 1 3 4  -> Parent 2 3 1 4
Vertex 2 3 1 4  -> Parent 2 3 4 1
Vertex 2 3 4 1  -> Parent 2 3 1 4
Vertex 2 4 3 1  -> Parent 2 3 4 1
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Vertex 2 1 3 4  -> Parent 1 2 3 4
Vertex 2 3 1 4  -> Parent 3 2 1 4
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
Vertex 2 1 3 4  -> Parent 2 1 4 3
Vertex 2 1 4 3  -> Parent 2 1 3 4
Vertex 2 3 1 4  -> Parent 2 1 3 4
Vertex 2 4 1 3  -> Parent 2 1 4 3
mpi@bilal-khawar-VirtualBox:~$
```

Three Processes:



```
mpi@bilal-khawar-VirtualBox:~$ mpirun -np 3 ./ist_mpi
Time taken for IST construction: 0.000008 seconds
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Vertex 2 1 3 4  -> Parent 2 3 1 4
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Vertex 2 1 3 4  -> Parent 1 2 3 4
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
Vertex 2 1 3 4  -> Parent 2 1 4 3
Vertex 2 1 4 3  -> Parent 2 1 3 4
mpi@bilal-khawar-VirtualBox:~$
```

Four Processes:



```
mpi@bilal-khawar-VirtualBox:~$ mpirun -np 4 ./ist_mpi
Time taken for IST construction: 0.000004 seconds
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
mpi@bilal-khawar-VirtualBox:~$
```

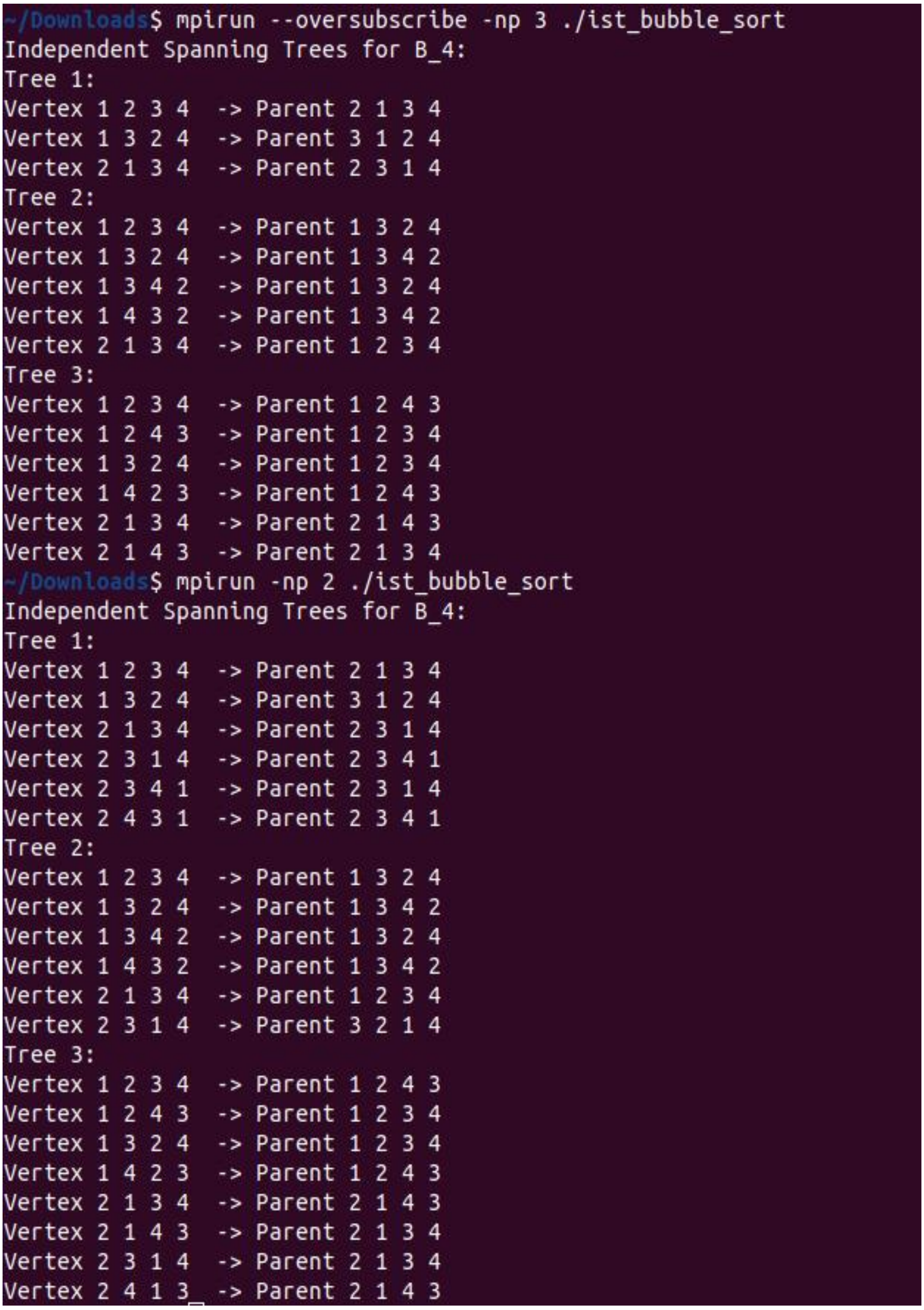
8 Additional Notes

- **Dataset:** The algorithm does not require an external dataset, as it operates on the bubble-sort network B_n, with vertices generated as permutations.
- **METIS:** The PDC project mentions METIS for graph partitioning, but it is not used here, as IST construction does not require partitioning. Future extensions could integrate METIS for larger networks.
- **OpenMP/OpenCL:** The current implementation uses MPI for parallelism. OpenMP could be added for intra-node parallelism, but it is not included in this version.
- **Limitations:** The implementation is optimized for small n (e.g., 4) due to the factorial growth of $n!$. For larger n, memory and computation time increase significantly.

9 Future Work

- Optimize MPI communication using non-blocking calls (e.g., MPI_Isend) to reduce serialization, as suggested in the Intel MPI tutorial.
- Integrate OpenMP for hybrid parallelism within nodes.

Figure 1: Placeholder for Project Diagram or Screenshot



```
~/Downloads$ mpirun --oversubscribe -np 3 ./ist_bubble_sort
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Vertex 2 1 3 4  -> Parent 2 3 1 4
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Vertex 2 1 3 4  -> Parent 1 2 3 4
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
Vertex 2 1 3 4  -> Parent 2 1 4 3
Vertex 2 1 4 3  -> Parent 2 1 3 4
~/Downloads$ mpirun -np 2 ./ist_bubble_sort
Independent Spanning Trees for B_4:
Tree 1:
Vertex 1 2 3 4  -> Parent 2 1 3 4
Vertex 1 3 2 4  -> Parent 3 1 2 4
Vertex 2 1 3 4  -> Parent 2 3 1 4
Vertex 2 3 1 4  -> Parent 2 3 4 1
Vertex 2 3 4 1  -> Parent 2 3 1 4
Vertex 2 4 3 1  -> Parent 2 3 4 1
Tree 2:
Vertex 1 2 3 4  -> Parent 1 3 2 4
Vertex 1 3 2 4  -> Parent 1 3 4 2
Vertex 1 3 4 2  -> Parent 1 3 2 4
Vertex 1 4 3 2  -> Parent 1 3 4 2
Vertex 2 1 3 4  -> Parent 1 2 3 4
Vertex 2 3 1 4  -> Parent 3 2 1 4
Tree 3:
Vertex 1 2 3 4  -> Parent 1 2 4 3
Vertex 1 2 4 3  -> Parent 1 2 3 4
Vertex 1 3 2 4  -> Parent 1 2 3 4
Vertex 1 4 2 3  -> Parent 1 2 4 3
Vertex 2 1 3 4  -> Parent 2 1 4 3
Vertex 2 1 4 3  -> Parent 2 1 3 4
Vertex 2 3 1 4  -> Parent 2 1 3 4
Vertex 2 4 1 3  -> Parent 2 1 4 3
```

- Explore extensions to generalized bubble-sort graphs or other network topologies, as proposed in the research paper.
- Add visualization tools to display the IST structures graphically.

10 References

- Kao, S.-S., et al. *A Parallel Algorithm for Constructing Multiple Independent Spanning Trees in Bubble-Sort Networks*. *Journal of Parallel and Distributed Computing*, 2023.
- PDC Project Description (provided in course materials).
- Intel Tutorial: *Analyzing MPI Applications* (for performance optimization guidance).