Parallel vs Serial Implementation of IST Parent Finding Algorithm

Your Name

May 6, 2025

1 Introduction

This report presents an implementation of the Parent1 algorithm, which is used to compute parent vertices in the Interconnection Sorting Tree (IST) model. The algorithm is implemented in C++, with and without OpenMP parallelism. A performance comparison is performed across different values of n (the dimension of the bubble-sort network).

2 Algorithm Description

The core of the implementation follows Algorithm 1 from the source paper. It relies on three main components:

- **Parent1:** Determines the parent of a given vertex v in tree T_t^n.
- **FindPosition:** Helps locate the adjacent vertex for specific cases in Parent1.
- **Swap:** Swaps elements in a permutation.

These components work together to navigate through permutations and compute their parent vertices.

3 Implementation Details

The code is written in C++, utilizing the `std::next_permutation` function to generate all permutations of $\{1, 2, \ldots, n\}$. For each permutation, we compute its parent in each tree $T_1, T_2, \ldots, T_{n-1}$. OpenMP is used to parallelize the parent computation process.

The main steps are:

1. Generate all permutations.

2. For each permutation, compute its parents in all trees.

3. Allow user input for a specific permutation and output its parents.

OpenMP was applied using `#pragma omp parallel for` to parallelize the parent computation loop.

4 Time Comparison

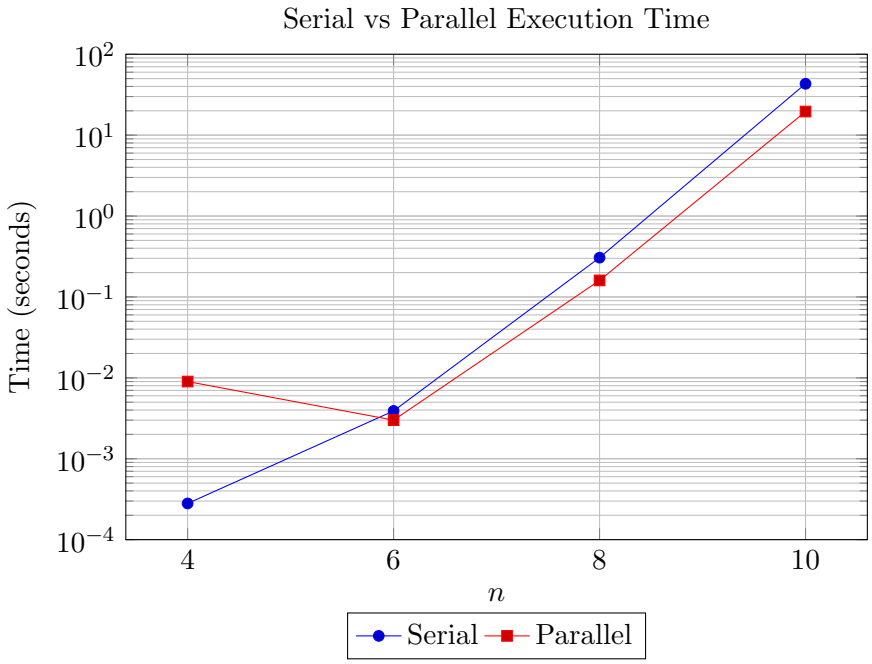
The table below shows the execution time (in seconds) for both serial and parallel implementations for different values of n.

n	**Serial Time (s)**	**Parallel Time (s)**
4	0.00028	0.009
6	0.0039	0.003
8	0.306	0.16
10	43.1526	19.58

Table 1: Execution Time Comparison: Serial vs Parallel

5 Performance Chart

The following chart visualizes the time comparison between serial and parallel execution:



6 Discussion

The results demonstrate that for small values of n (e.g., $n = 4, 6$), the overhead of parallelism outweighs its benefits, leading to longer runtimes in the parallel implementation. However, as n increases (e.g., $n = 8, 10$), parallelization yields significant improvements, nearly halving the execution time for $n = 10$.

7 Conclusion

Parallelization with OpenMP provides substantial performance gains for large input sizes, though care must be taken for small inputs where parallel overhead can negate speedup. Future work can explore more advanced parallel strategies or GPU acceleration to handle even larger n efficiently.

Github Repository: https://github.com/Salmanjaved03/PDCProject