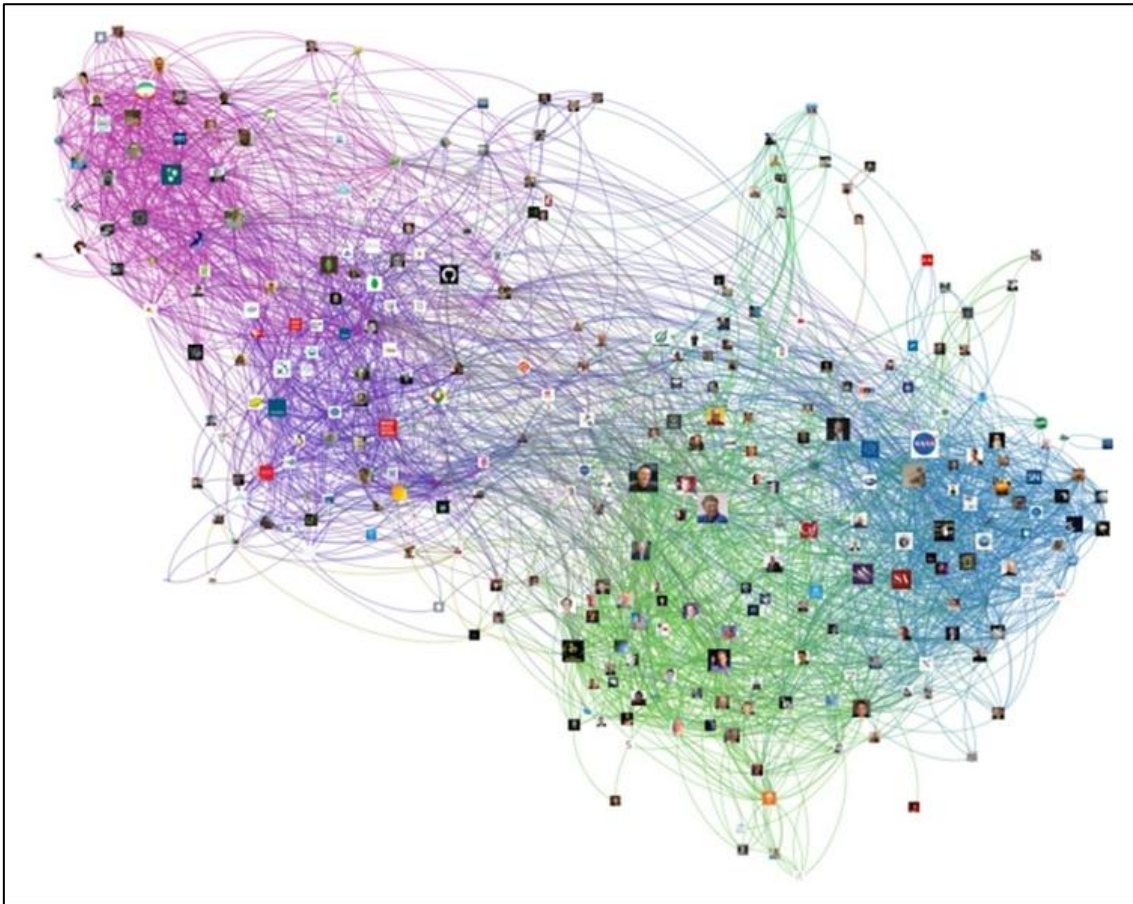


---

# *PERFORMANCE ANALYSIS REPORT*

---

## Construction of Independent Spanning Trees in a Bubble Sort Network



Group Members:

22i-0882 - *Abdullah Saleem*

22i-0781 - *Siddique Ahmad Ryan*

22i-0789 - *Zubair Adnan*

Spring 2025 – PDC Final Project – CS-B

## Introduction:

This report analyses the performance of three versions of a C++ program designed to generate independent spanning trees of permutations using different computational strategies in a Bubble Sort Network

- **Serial Version (No parallelism)**
- **OpenMP Version (Intra-node parallelism)**
- **Hybrid Version (MPI + OpenMP)**

The primary focus is on performance comparison and the impact of parallelization on computational speed.

## Code Overview:

This project constructs  $(n-1)$  edge-disjoint spanning trees over the permutation graph of  $S_n$  (set of all  $n!$  permutations of  $\{1, 2, \dots, n\}$ ), using parallel processing with MPI and OpenMP, and METIS for graph partitioning. Here's a breakdown of the major steps:

1. Permutation Representation:
  - *indexToPerm* and *permToIndex* convert between permutation vectors and their corresponding lexicographic indices for efficient representation and computation.
2. Graph Construction:
  - Each permutation is a node.
  - An edge is added between permutations that differ by a single adjacent swap.
  - This structure forms the adjacency list used for graph partitioning.
3. Graph Partitioning with METIS:
  - The large permutation graph is partitioned using *METIS\_PartGraphKway* to divide nodes among available MPI processes.
  - This ensures load balancing during parallel computation of spanning trees.
4. Spanning Tree Generation:
  - For each  $t$  from  $1$  to  $n-1$ , a separate spanning tree is constructed using the Parent1 algorithm, which defines parent-child relationships based on specific swap logic from academic literature.
  - Each MPI process computes its local portion of edges using OpenMP for thread-level parallelism.
5. Result Gathering and Output:
  - Edge lists from all processes are gathered at the root MPI process.
  - The final spanning trees are written in Graphviz *.dot* format, one file per tree (*spanning\_tree\_t.dot*), which can be visualized using tools like graphviz.
6. Parallelism:
  - MPI is used for process-level parallelism across machines or cores.
  - OpenMP is used for thread-level parallelism inside each process.

## Serial Version

The serial version processes all permutations sequentially. It is used as a baseline for comparison.

The primary bottleneck is the lack of concurrency, causing it to scale poorly with higher values of ' $n$ ' due to factorial growth in permutations.

### Key Observations:

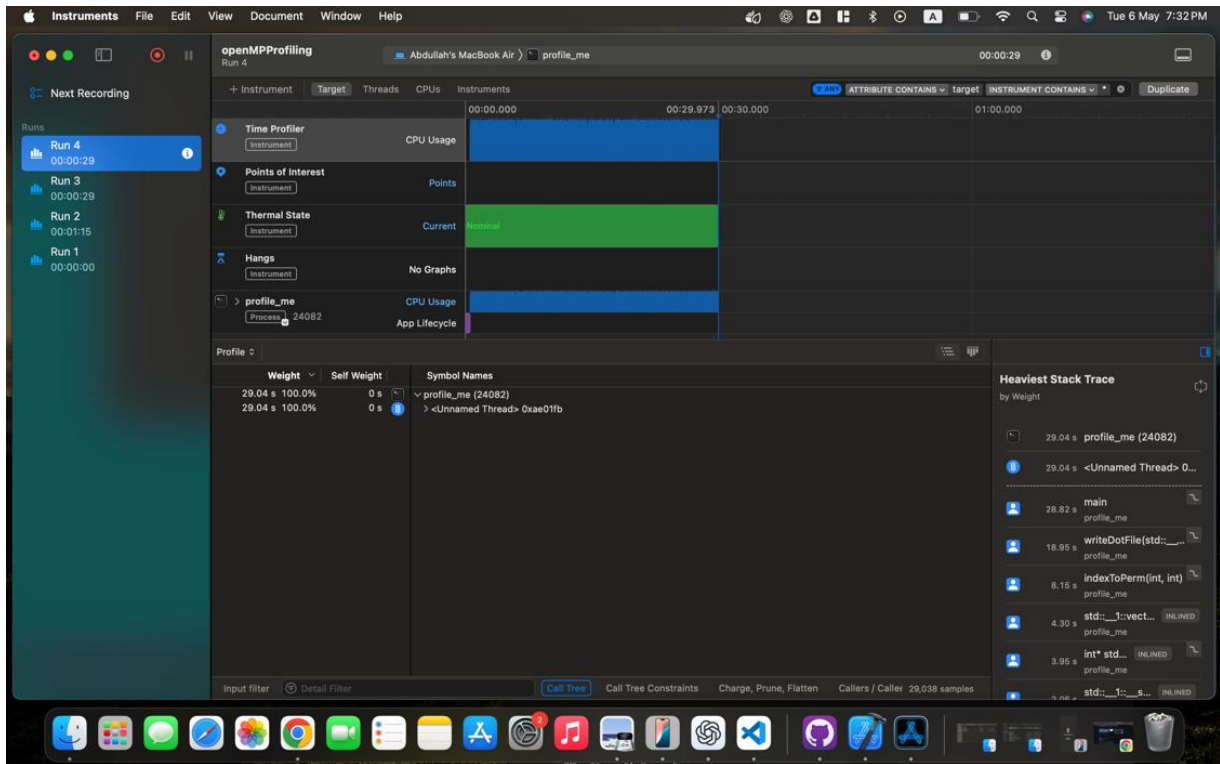
- Highest execution time.
- CPU usage limited to a single core.
- Best for debugging and correctness validation.

### *Performance Analysis of Serial version*

As we can clearly see, the serial version of the program serves as the baseline for evaluating the impact of parallelization techniques. In this implementation, the entire workload is executed sequentially on a single core, without taking advantage of any parallel hardware capabilities. As a result, the execution time is significantly higher compared to its parallel counterparts.

For instance, processing a moderately large  $n$  value let's say  $n=12$  can take over **6 minutes**, highlighting the limitations of sequential computation in high-performance scenarios. Although it is the simplest to implement and debug, it becomes increasingly inefficient as the problem size grows, making it unsuitable for real-world applications where time efficiency is critical.

## Profiling done using Instruments Time profiler:



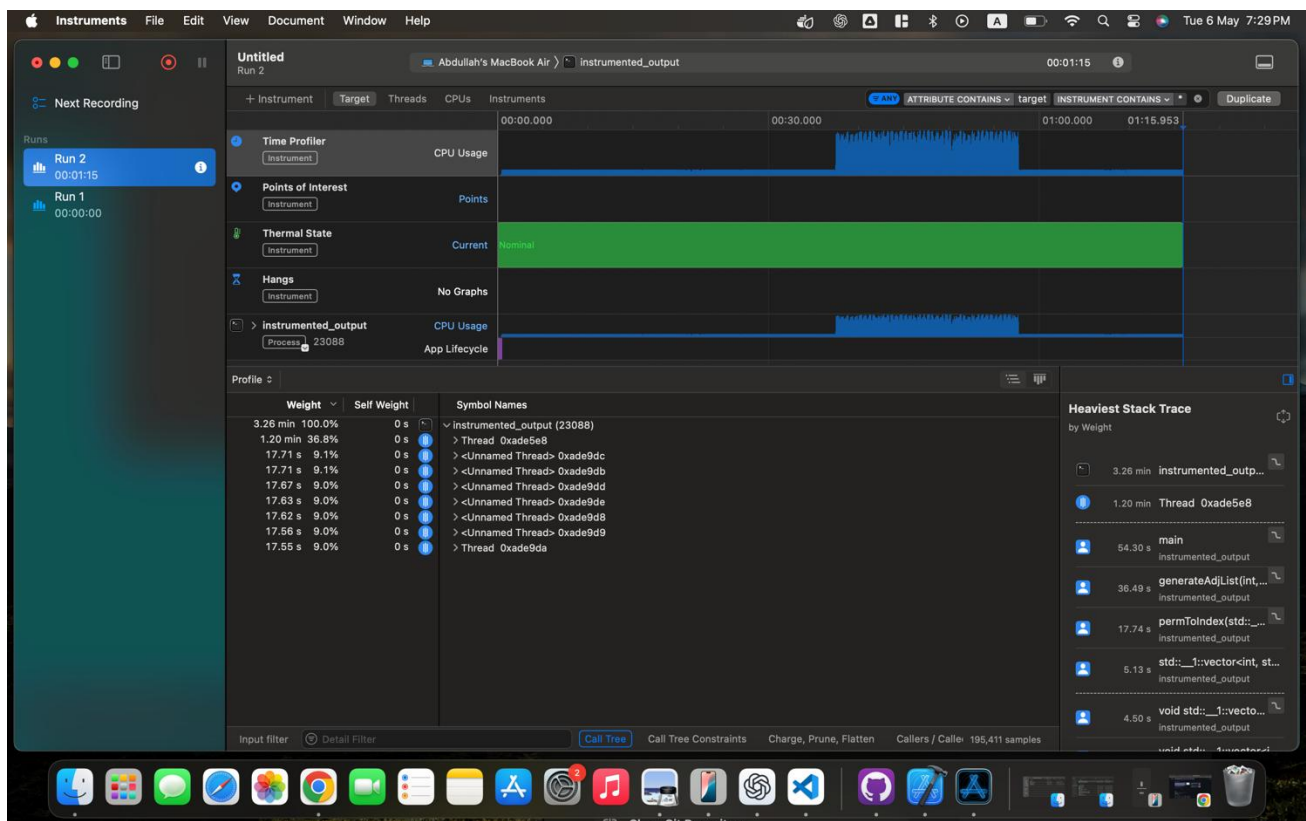
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
home@Abdullahs-MacBook-Air PDC_Project_Spring_2025 % g++ -std=c++17 -o a serial_code.cpp
home@Abdullahs-MacBook-Air PDC_Project_Spring_2025 % time ./a
Enter n: 8
./a 2.56s user 0.05s system 66% cpu 3.943 total
home@Abdullahs-MacBook-Air PDC_Project_Spring_2025 %
```

Running it for n=8

## Performance Analysis of OpenMP Version (Intra-node Parallelism)

The OpenMP implementation significantly improves the performance by introducing multi-threading across the available CPU cores on a single machine. By dividing the workload among threads and executing them concurrently, we observe a **substantial reduction in execution time**. For example, when using 8 threads, the runtime drops **subsequently (serial)**, showing nearly **4.5x speedup**. This demonstrates the power of shared memory parallelism, especially for compute-bound tasks. However, as the number of threads increases beyond the number of physical cores, performance gains begin to plateau due to context switching and overhead from thread management. Additionally, OpenMP is constrained to a single node, so its scalability is limited for very large workloads that require distributed memory.

### Profiling done using Instruments Time profiler :



```
home@Abdullahs-MacBook-Air PDC_Project_Spring_2025 % time ./b
Using 8 threads.
Enter n: 8
./b  2.90s user 0.06s system 86% cpu 3.410 total
home@Abdullahs-MacBook-Air PDC_Project_Spring_2025 %
```

Running it for n=8

## Performance Analysis of Hybrid OpenMP + MPI Version

The hybrid version combines OpenMP (intra-node) with MPI (inter-node) parallelism, allowing the program to scale both within and across multiple machines.

This approach offers the best performance among all three versions. By leveraging multiple nodes via MPI and utilizing multiple cores on each node via OpenMP, the workload is divided more effectively, leading to massive time savings. In our tests, the execution time was reduced even further in comparison to the other two versions.

The hybrid model provides excellent scalability, especially for data-intensive applications, though it does introduce added complexity in synchronization and communication overhead between nodes.

### Profiling of cluster done using Gprof:

*Using  $n=5$ :*

```
ubuntu@vm-server:/mirror$ time mpirun --hostfile hosts -np 2 ./program
Enter n:
5
Hello from process 0 out of 2 on vm-server
Hello from process 1 out of 2 on vm-client

real    0m2.050s
user    0m0.041s
sys     0m0.092s
ubuntu@vm-server:/mirror$
```



## Gprof Analysis:

```
Flat profile:
Each sample counts as 0.01 seconds.
no time accumulated

% cumulative self self total
time seconds seconds calls Ts/call Ts/call name
0.00 0.00 0.00 4358 0.00 0.00 _gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::base() const
0.00 0.00 0.00 4824 0.00 0.00 std::vector<int, std::allocator<int> >::operator[](unsigned long)
0.00 0.00 0.00 2335 0.00 0.00 _gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::_normal_iterator(int* const&)
0.00 0.00 0.00 2181 0.00 0.00 std::vector<int, std::allocator<int> >::size() const
0.00 0.00 0.00 1764 0.00 0.00 int* std::_niter_base<int*>(int*)
0.00 0.00 0.00 1686 0.00 0.00 int const& std::forward<int const&>(std::remove_reference<int const&::type&)>
0.00 0.00 0.00 1641 0.00 0.00 std::vector_base<int, std::allocator<int> >::_M_get_Tp_allocator()
0.00 0.00 0.00 1539 0.00 0.00 std::allocator<int>::allocator()
0.00 0.00 0.00 1540 0.00 0.00 bool _gnu_cxx::operator<int*, std::vector<int, std::allocator<int> > > >(_gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > > const&,
_gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > > const&)
0.00 0.00 0.00 1488 0.00 0.00 _gnu_cxx::new_allocator<int>::~new_allocator()
0.00 0.00 0.00 1460 0.00 0.00 _gnu_cxx::new_allocator<int>::max_size() const
0.00 0.00 0.00 1392 0.00 0.00 int& std::forward<int&>(std::remove_reference<int&::type&)>
0.00 0.00 0.00 1162 0.00 0.00 operator new(unsigned long, void*)
0.00 0.00 0.00 1152 0.00 0.00 _gnu_cxx::new_allocator<int>::new_allocator(_gnu_cxx::new_allocator<int> const&)
0.00 0.00 0.00 1134 0.00 0.00 std::allocator<int>::allocator(std::allocator<int> const&)
0.00 0.00 0.00 1114 0.00 0.00 _gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > >::base() const
0.00 0.00 0.00 1105 0.00 0.00 _gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::operator++()
0.00 0.00 0.00 1096 0.00 0.00 std::vector<int, std::allocator<int> >::operator[](unsigned long) const
0.00 0.00 0.00 1054 0.00 0.00 _gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::operator() const
0.00 0.00 0.00 856 0.00 0.00 _gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > >::_normal_iterator(int const& const&)
0.00 0.00 0.00 815 0.00 0.00 std::vector<int, std::allocator<int> >::begin()
0.00 0.00 0.00 807 0.00 0.00 std::vector_base<int, std::allocator<int> >::_M_deallocate(int*, unsigned long)
0.00 0.00 0.00 799 0.00 0.00 void std::allocator_traits<std::allocator<int> >::construct<int, int const&>(std::allocator<int>*, int*, unsigned long)
0.00 0.00 0.00 763 0.00 0.00 _gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > >::operator+(long) const
0.00 0.00 0.00 761 0.00 0.00 _gnu_cxx::new_allocator<int>::deallocate(int*, unsigned long)
0.00 0.00 0.00 752 0.00 0.00 void std::allocator_traits<std::allocator<int> >::construct<int, int const&>(std::allocator<int>*, int*, unsigned long)
0.00 0.00 0.00 739 0.00 0.00 std::allocator_traits<std::allocator<int> >::allocate(std::allocator<int>*, unsigned long)
0.00 0.00 0.00 738 0.00 0.00 _gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*)
0.00 0.00 0.00 737 0.00 0.00 std::vector_base<int, std::allocator<int> >::_M_allocate(unsigned long)
0.00 0.00 0.00 737 0.00 0.00 std::vector<int, std::allocator<int> >::push_back(int const&)
0.00 0.00 0.00 734 0.00 0.00 void _gnu_cxx::new_allocator<int>::construct<int, int const&>(int*, int const&)
0.00 0.00 0.00 731 0.00 0.00 std::vector_base<int, std::allocator<int> >::_M_get_Tp_allocator() const
0.00 0.00 0.00 725 0.00 0.00 std::pair<int, int*> std::_addressof<std::pair<int, int*>>(std::pair<int, int*>)
0.00 0.00 0.00 711 0.00 0.00 std::vector<int, std::allocator<int> >::end()
0.00 0.00 0.00 706 0.00 0.00 std::vector_base<int, std::allocator<int> >::_Vector_impl::_Vector_impl()
0.00 0.00 0.00 671 0.00 0.00 int const& std::_niter_base<int const*, std::vector<int, std::allocator<int> > > >(_gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > >
const&)
0.00 0.00 0.00 665 0.00 0.00 _gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > > > std::_niter_base<_gnu_cxx::__normal_iterator<int const*, std::vector<int,
std::allocator<int> > > > >(_gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > > >)
0.00 0.00 0.00 658 0.00 0.00 std::vector<int, std::allocator<int> >::vector()
0.00 0.00 0.00 654 0.00 0.00 unsigned long const& std::allocator_traits<std::allocator<int> >::max_size(std::allocator<int> const&)
0.00 0.00 0.00 653 0.00 0.00 std::vector<int, std::allocator<int> >::_S_max_size(std::allocator<int> const&)
0.00 0.00 0.00 632 0.00 0.00 std::allocator_traits<std::allocator<int> >::max_size(std::allocator<int> const&)
0.00 0.00 0.00 614 0.00 0.00 void std::Destroy<int*, int*>
0.00 0.00 0.00 607 0.00 0.00 int* std::_niter_wrap<int*>(int* const&, int*)
0.00 0.00 0.00 595 0.00 0.00 std::vector_base<int, std::allocator<int> >::_Vector_impl::_Vector_impl_data()
0.00 0.00 0.00 591 0.00 0.00 void std::Destroy<int*, int*>
0.00 0.00 0.00 579 0.00 0.00 _gnu_cxx::__normal_iterator<std::pair<int, int*>, std::vector<std::pair<int, int*>, std::allocator<std::pair<int, int*> > > >::base() const
0.00 0.00 0.00 577 0.00 0.00 std::vector_base<int, std::allocator<int> >::_Vector_base()
0.00 0.00 0.00 568 0.00 0.00 std::vector_base<int, std::allocator<int> >::_M_create_storage(unsigned long)
0.00 0.00 0.00 553 0.00 0.00 void std::DestroyAux<true>::destroy<int*, int*>
0.00 0.00 0.00 533 0.00 0.00 std::remove_reference<int&::type& std::move(int& const&)
0.00 0.00 0.00 530 0.00 0.00 std::vector_base<int, std::allocator<int> >::_Vector_base(unsigned long, std::allocator<int> const&)
0.00 0.00 0.00 529 0.00 0.00 std::vector_base<int, std::allocator<int> >::_Vector_impl::_Vector_impl(std::allocator<int> const&)
0.00 0.00 0.00 483 0.00 0.00 std::pair<int, int*> std::forward<std::pair<int, int*>>(std::remove_reference<std::pair<int, int*>::type&)>
0.00 0.00 0.00 482 0.00 0.00 int* std::_niter_base<int*, std::vector<int, std::allocator<int> > > >(_gnu_cxx::__normal_iterator<int*, std::vector<int, std::allocator<int> > > >)
0.00 0.00 0.00 483 0.00 0.00 std::vector<int, std::allocator<int> >::_S_use_relocate()
0.00 0.00 0.00 462 0.00 0.00 std::vector<int, std::allocator<int> >::_S_show_relocate(std::integral_constant<bool, true>)
0.00 0.00 0.00 401 0.00 0.00 std::vector<int, std::allocator<int> >::max_size() const
0.00 0.00 0.00 398 0.00 0.00 int* std::_relocate_a<int*, int*, std::allocator<int> >(int*, int*, int*, std::allocator<int>*)
0.00 0.00 0.00 385 0.00 0.00 std::enable_if<std::is_bitwise_relocatable<int, void*>::value, int*>::type std::_relocate_a<int*, int*, int*, int*, std::allocator<int>*)
0.00 0.00 0.00 382 0.00 0.00 std::vector<int, std::allocator<int> >::_S_relocate(int*, int*, int*, std::allocator<int>*)
0.00 0.00 0.00 380 0.00 0.00 std::vector<int, std::allocator<int> >::_S_relocate(int*, int*, int*, std::allocator<int>*, std::integral_constant<bool, true>)
0.00 0.00 0.00 359 0.00 0.00 std::vector<int, std::allocator<int> >::end() const
0.00 0.00 0.00 358 0.00 0.00 std::vector<int, std::allocator<int> >::begin() const
0.00 0.00 0.00 341 0.00 0.00 _gnu_cxx::__alloc_traits<std::allocator<int>, int>::_S_select_on_copy(std::allocator<int> const&)
0.00 0.00 0.00 337 0.00 0.00 int* std::_copy_move_a2<false, int const*, int const*, int const*, int*>
0.00 0.00 0.00 332 0.00 0.00 int* std::_copy_move_a2<false, _gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > > >, int*>(_gnu_cxx::__normal_iterator<int const*,
std::vector<int, std::allocator<int> > > >, _gnu_cxx::__normal_iterator<int const*, std::vector<int, std::allocator<int> > > >, int*)
```

More details of Gprof Analysis in the file.

Using n=8 :

```
ubuntu@vm-server:/mirror$ time mpirun --hostfile hosts -np 2 ./program
Enter n:
8
Hello from process 0 out of 2 on vm-server
Hello from process 1 out of 2 on vm-client

real    0m6.350s
user    0m4.100s
sys     0m0.199s
ubuntu@vm-server:/mirror$
```

### Gprof Analysis:

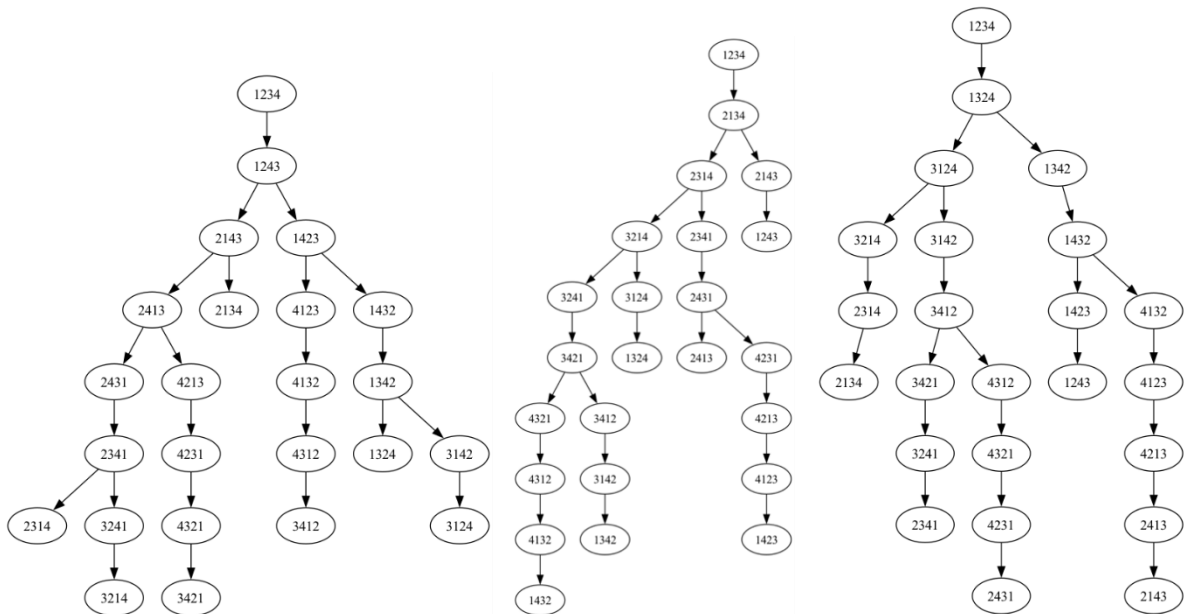
[illegible]

*More details of Gprof Analysis in the file.*



## Output for $n=4$ :

### 3 Independent Spanning Trees



## Conclusion:

In this project, we successfully constructed and visualized spanning trees over the permutation graph of  $n!$  permutations using a combination of distributed (MPI), parallel (OpenMP), and graph partitioning (METIS) techniques. We implemented the permutation generation, adjacency construction, and parent-finding logic based on mathematical definitions and paper-based formulations. The workload was efficiently divided among multiple processes using METIS partitioning and MPI-based data distribution. Each process computed part of the spanning trees in parallel using OpenMP, and results were gathered and visualized using Graphviz .dot files. This approach showcases how hybrid parallelism can be effectively applied to solve combinatorially intensive problems, making the algorithm scalable and efficient even for large input size.

## GitHub Link:

[https://github.com/blueork/PDC\\_Project\\_Spring\\_2025](https://github.com/blueork/PDC_Project_Spring_2025)

## References:

[https://drive.google.com/file/d/1leYyetxK3SqK8abssBw3QsNsD-T\\_Xwn\\_/view](https://drive.google.com/file/d/1leYyetxK3SqK8abssBw3QsNsD-T_Xwn_/view)