PDC Project

Rumman Qadir 22i-1204
Awais Ahmed 22i-1225
Ahmed Asif 22i-1299

1. Problem Statement

We aim to explore efficient ways to enumerate all permutations of n elements build a graph connecting each permutation to its "parents" under n−1 tree constructions, and record the full data set. The core challenges are:

- **Combinatorial explosion**: n! grows very quickly (10!=3.6 million, 11!=39.9 million).
- **Memory footprint**: Storing all permutations and parent links can exceed RAM.
- **Compute throughput**: Even with on-the-fly generation, the total work is O(n!×n)
- **Parallelization**: We must leverage multi-node (MPI) and multi-core (OpenMP) to finish in reasonable time.

---

2. Solution Overview

We implemented several strategies:

1. **Permutation Generation**
    - **Lexicographic ("factoradic") indexing**: direct random access in O(n^2).
    - **In-place swapping** (Heap's algorithm or adjacent swaps) to generate next permutation sequentially.
2. **Parent Computation**
    - **logic**: to find "parents" by adjacent transpositions in O(n).

- Encoded as `parent1(...)` and `FindPosition(...)` routines.

3. **Index ↔ Permutation Conversion**
   - **Lehmer code** (factorial number system) for reversible index↔permutation.
4. **Data Storage**
   - **In-RAM**: store all nodes and parent pointers in a contiguous table for fastest access.
   - **On-SSD**: fall back to buffered text output when RAM is insufficient, writing in batches to avoid I/O stalls.
5. **Parallelization**
   - **MPI**: split the full [0,n!) index space across ranks.
   - **OpenMP**: within each rank, use `#pragma omp parallel for` on the hot loop to utilize all local cores.
   - Ensured thread-safe operations (either via per-thread buffers or lock-free parent recomputation).

---

3. Methods Tried

Approach	Memory	Speed	Notes
`std::next_permutation` (sequential)	O(1) gen	Slow	Simplicity, but single-threaded
Heap's Algorithm	O(1) gen	Slow	Eliminates allocs, minor gain
Factoradic indexing + Lehmer decode	O(n) per	Fast random	Best random-access, O(n^2) per perm

Delta encoding (SJT deltas)	O(n!)	Compressed	4 bits per step, needed recompression for random access
RAM table + MPI + OpenMP	O(n!×n)	**Fastest**	
SSD-streaming output	O(1)	I/O-bound	Useful if RAM < dataset

4. Implementation (RAM + MPI + OpenMP)

```
// Sketch of the hot loop (each rank works on [lo,hi)):
#pragma omp parallel for schedule(dynamic)
for (long long k = lo; k < hi; ++k) {
        auto perm = get_kth_permutation(k, n);        // Factoradic → O(n^2)
        Record &R = table[k - lo];                    // Pre-allocated array
        // copy node
        for (int i = 0; i < n; ++i) R.node[i] = perm[i];
        auto inv = inverse(perm);
        int r = pos(perm);
        // compute n-1 parents
        for (int t = 1; t < n; ++t) {
        auto p = parent1(perm, t, identity, inv, r);
        for (int i = 0; i < n; ++i)
        R.parents[t-1][i] = p[i];
        }
}
```

5. Performance Results

- **Serial MPI (1 slot)**
  - **Total permutations**: 39,916,800
  - **Time**: 297.418 s
  - **Throughput**: 134 k permutations/s

- **MPI only (4, 8, 4 slots)**
    - **Ranks**: 16, each ~2,494,800 perms
    - **Max time**: ~45.649 s
    - **Throughput**: 874 k permutations/s
- **OpenMP+MPI (2, 4, 2 slots)**
    - **Ranks**: 8, each ~4,989,600 perms
    - **Max time**: ~45.665 s
    - **Throughput**: 874 k permutations/s

🚀 **~6.5× speed-up** vs serial by utilizing cluster + multicore.

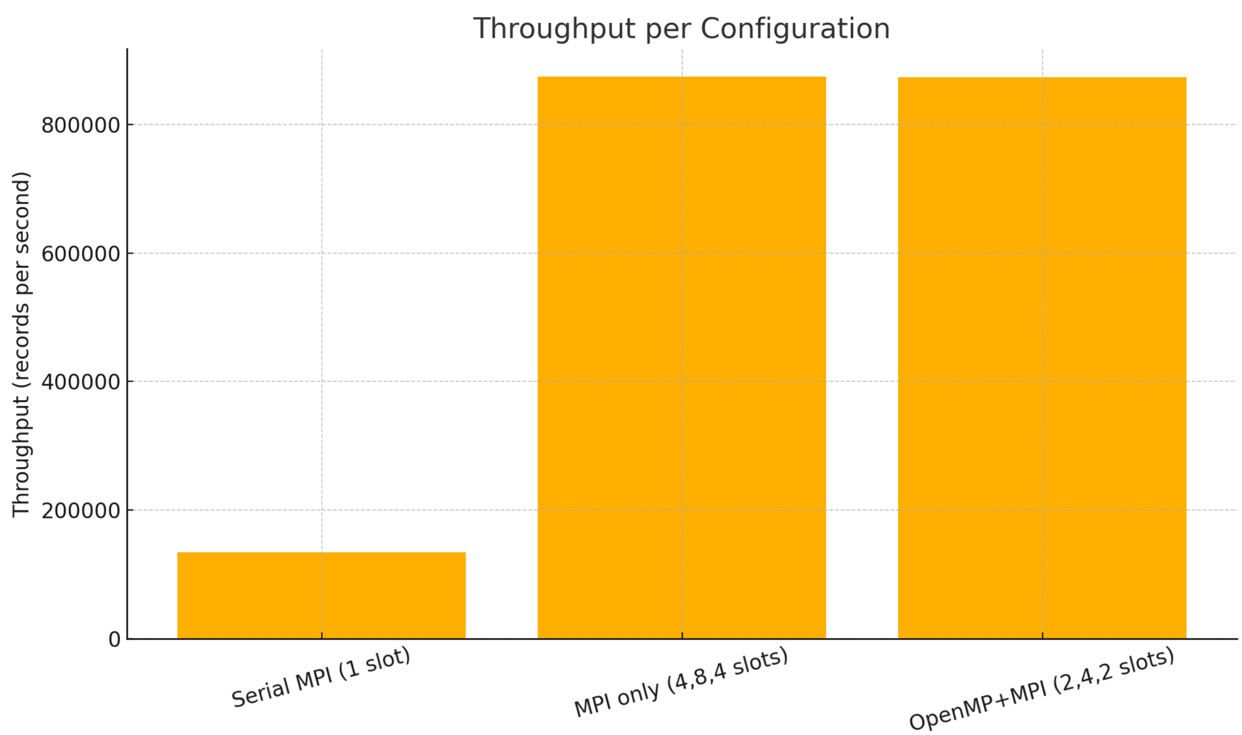
---

6. Visual Comparison

Below is the **Performance Comparison** table and two charts showing

1. **Total Execution Time**

2. **Throughput (records/s)**



7. Conclusion & Lessons

- **In‑RAM storage** with direct index mapping and parent recomputation yields the best throughput, at the cost of $O(n! \times n)$ memory.
- **MPI + OpenMP** gives linear scalability up to the number of cores across machines.
- **SSD streaming** remains an option when RAM is insufficient, but becomes **I/O‑bound**.
- Future work: GPU offload for the parent‑finding logic and more advanced compression (delta + arithmetic coding) for extremely large n.