Parallelizing Kruskal’s Algorithm for Finding Minimum Spanning Trees

By A.Abhiramee & Vaishnavi Bhuvanagiri

Summary:

We parallelized the minimum spanning tree algorithm by parallelizing Kruskal’s algorithm. In implementing Kruskal’s algorithm, we parallelized the sorting step (a divide and conquer merge sort algorithm). We used C++ and OpenMP to conduct parallelization across different numbers of threads in an attempt to achieve speedup.

Overview:

- Edge-Centric Parallelization - Distribute the edges across multiple processors.
- Merge sort to parallelize sorting after being distributed into multiple processors

Parallel Kruskal's Algorithm Implementation:

- **Input Graph:**
  Both Kruskal’s take in connected, undirected graphs as input.
- **Edge Distribution:**
  Distribute edges among different processors to facilitate parallel processing.
- **Parallel Sorting:(merge sort)**
  Each processor independently sorts its local set of edges by weight.
- **Local MST Construction:**
  Processors independently construct local fragments of the MST by adding edges that do not create cycles.
  Concurrently perform cycle detection and update disjoint-set data structures.
- **Communication and Merging:**
  Processors communicate to exchange information about edges and merge local MST fragments into a global MST.
  Coordinate to handle concurrent updates and synchronisation.
- **Final MST Construction:**

Repeat the process until all vertices are connected, and a global MST is constructed.

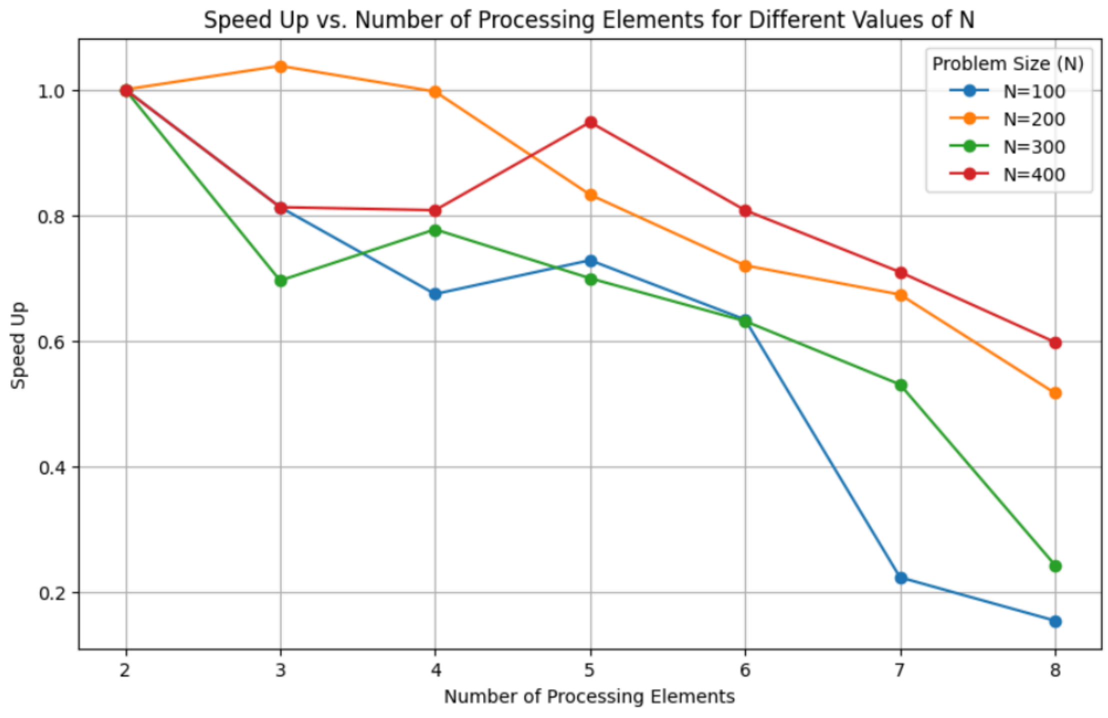
- **Output MST:**
  output a set of edges that directly represent the MST along with the total computation time.

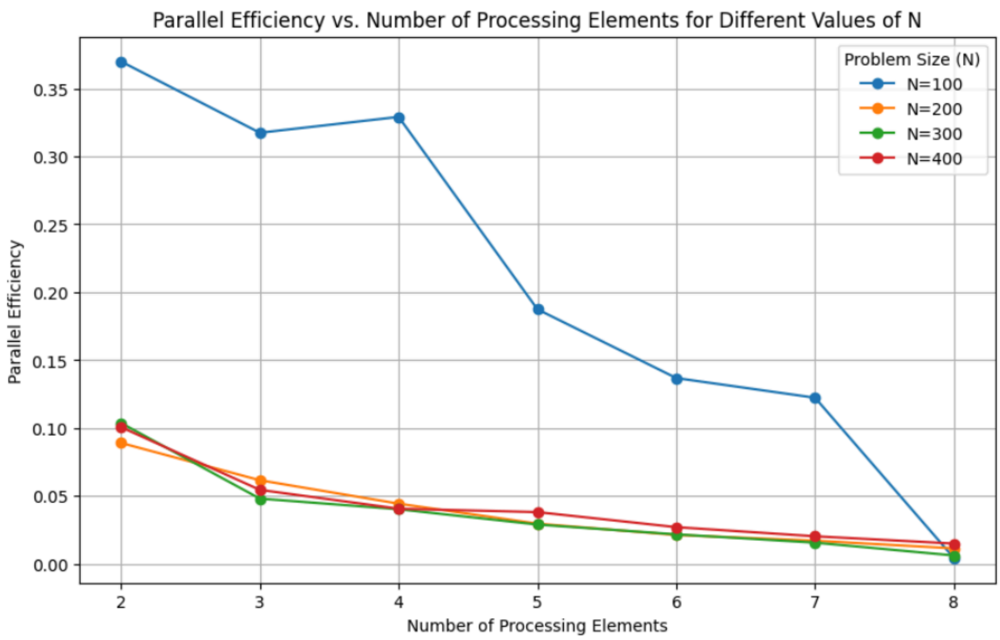
Parallelization Strategy:

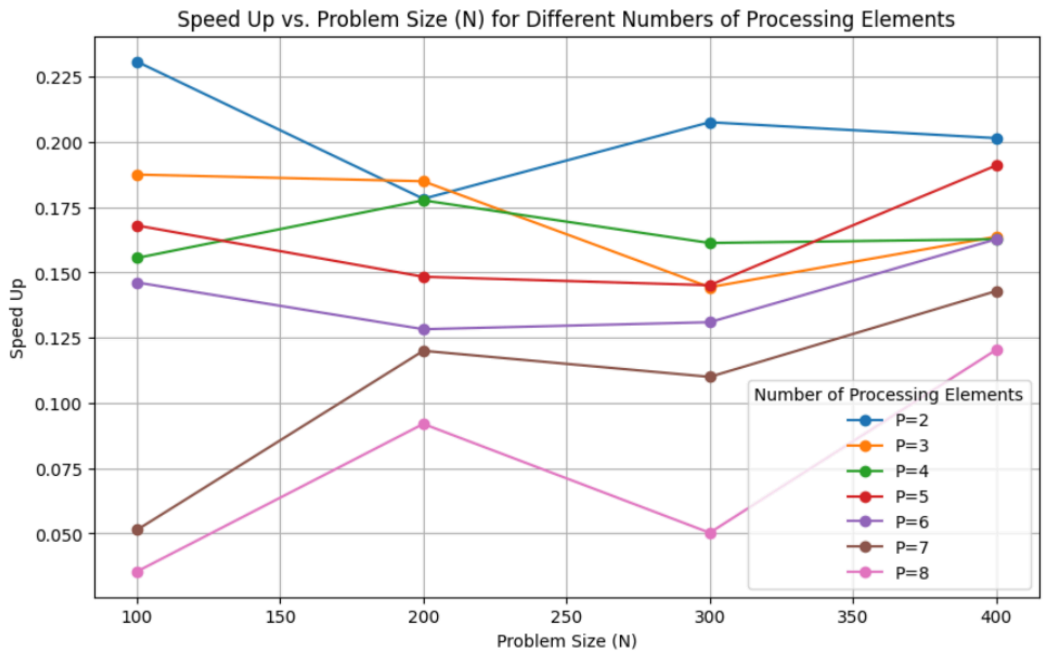
- Scatter edge list to different MPI processes.
- Sort edges in parallel using merge sort.
- Merge sorted lists to find MST in parallel.

Performance Analysis:

- speed up = (Running time of your program in 1 processing element)/(Running time of your program in p processing elements)
- Parallel Efficiency = (Speed up/p) x 100%







Advantages of HPC:

- Speed: HPC architectures offer immense computational power, speeding up complex algorithms like Kruskal’s.
- Parallelism: HPC systems excel in parallel processing, allowing simultaneous execution of tasks, perfect for Kruskal’s Algorithm.
- Scalability: Easily scale up resources on HPC clusters, accommodating growing data sizes efficiently.

- Resource Optimization: HPC efficiently utilizes hardware resources, reducing execution time and costs.
- Real-Time Insights: Faster computation means quicker insights, vital for time-sensitive applications.

Conclusion:

- Encourage further exploration and experimentation with parallel algorithms.

Future Work:

- Utilising CUDA for Parallel MST Computation:
  GPU Acceleration:Investigate the use of CUDA (Compute Unified Device Architecture) to harness the parallel processing power of GPUs for accelerating MST computations.

- Exploring Alternative Parallel MST Algorithms:
  - Prim's Algorithm: Investigate the parallelization of Prim's algorithm for MST construction, focusing on efficient data structures and parallel traversal strategies to exploit concurrency.
  - Borůvka's Algorithm: Explore the parallelization of Borůvka's algorithm, which iteratively merges clusters of vertices to construct the MST, and assess its performance compared to Kruskal's algorithm.

References:

1. Parallelization of Minimum Spanning Tree Algorithms Using Distributed Memory Architectures - Springer(http://www.scl.rs/papers/Loncar-TET-Springer.pdf)

2. Parallel Minimum Spanning Tree Algorithm - York University(https://wiki.eecs.yorku.ca/course_archive/2010-11/W/6490A/_media/public:xiwen.pdf)
3. A parallel Algorithm for computing minimum spanning tree (https://cs.wellesley.edu/~pmetaxas/mst.pdf)