Parallel and Distributed Computing

Project on HPC using MPI

Muhammad Anser Sohaib

367628

Part 1:

Table 1: HPC Cluster Node Map and Availability Ranking

Node	**Specs** **(cores, threads, Speed in MHz)**	**Usage Level** **(users, load, active tasks, longest simulation (hours))**	**Ranking Availability**
afrit	8, 16, 2266	5, 20.51, 593, 2	Should avoid for simulations
Compute-0-0	8, 16, 1600	1, 0.04, 366, 5	1
Compute-0-2	down	down	n/a
Compute-0-4	down	down	n/a
Compute-0-6	8, 16, 2267	1, 15, 371,4	2
Compute-0-16	8, 16, 2267	2, 12, 366,21	4
Compute-0-28	8, 16, 2267	1, 12, 367, 15	3
…	…	…	…

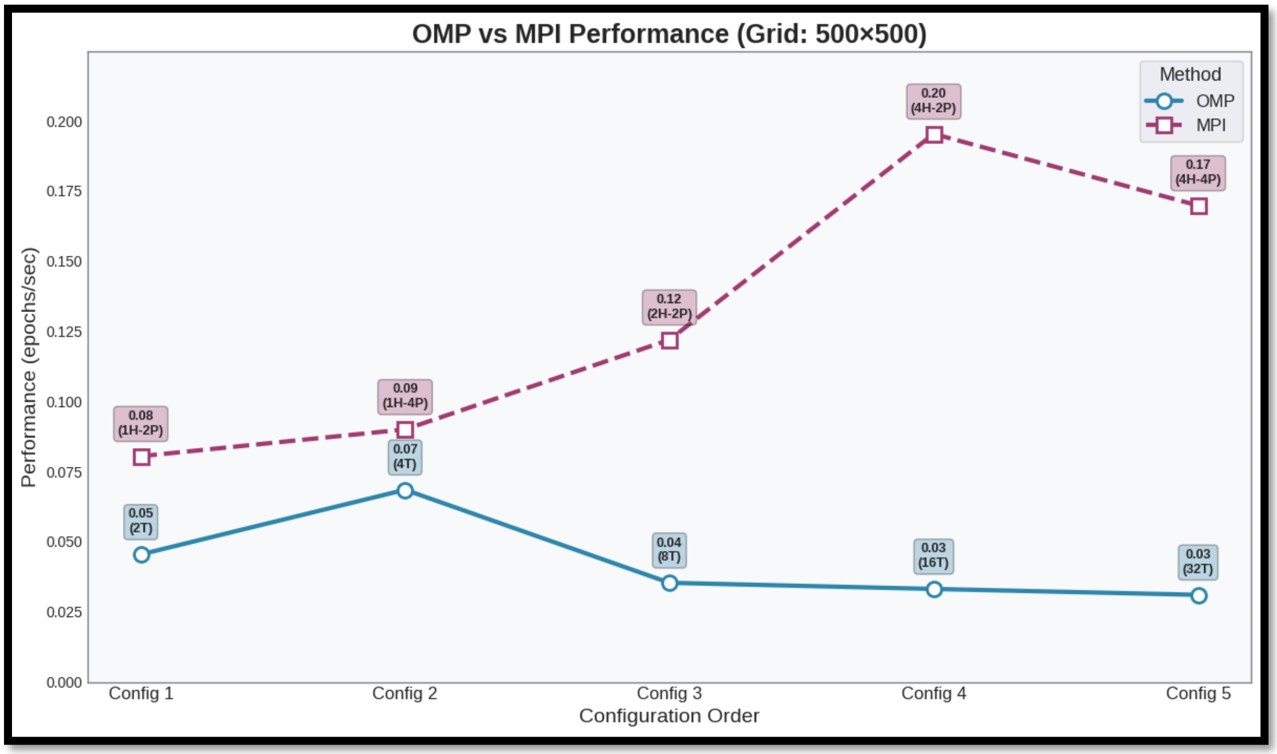
Since compute-0-0 is the best compute node for now. I'll continue using it for the next part.

Part 2:

I am using the following Compute Nodes based on availability:

Compute-0-5, Compute-0-6, Compute-0-8, Compute-0-9,

Performance:



Experimental Procedure and Data Collection

We evaluated the performance of a parallel Laplace solver using two parallelization models: **OpenMP (OMP)** for shared-memory and **MPI** for distributed-memory architectures. The solver was executed on a fixed grid size of **500×500** to ensure consistency across experiments.

Procedure:

1. **Environment Setup**

- All experiments were run on a high-performance cluster supporting both multi-threading (OpenMP) and multi-processing (MPI).
- Compiler optimizations (-O3) and proper affinity settings were ensured.

2. **OpenMP Execution**
   - The solver was compiled with OpenMP flags.
   - Experiments were run with **2T, 4T, 8T, 16T, and 32T** threads.
   - Execution time and convergence behavior were logged.
   - Performance was computed as epochs/sec = total_epochs / time.
3. **MPI Execution**
   - The solver was compiled with MPI support.
   - Experiments were conducted with **1H-2P, 1H-4P, 2H-2P, 4H-2P, and 4H-4P** configurations.
   - Data included total time, convergence status, and output at key grid points.
   - Performance was similarly computed as epochs/sec.
4. **Data Logging**
   - For each run, the following were recorded:
     - Thread/process config
     - Total time

        - Max difference (for convergence)
        - Whether convergence was achieved
        - Epochs per second (performance)

5. **Postprocessing**
    - Results were extracted, normalized, and visualized using pandas, matplotlib, and seaborn.
    - Performance trends were analyzed across thread/process scales.

This setup allowed a fair and direct comparison between shared memory and distributed-memory strategies under controlled conditions.

Problems Faced During Implementation

Initially, setting up the HPC environment was challenging. Configuring **CMake**, enabling **OpenMP**, and ensuring proper compiler support took significant effort, especially for someone using it for the first time. Understanding thread pinning, memory affinity, and debugging parallel execution added to the learning curve. However, with trial and error, it became manageable over time.

Transferring files between local and remote systems using scp and rsync also posed difficulties in the beginning due to path issues and permission errors. Once the workflow was established, it became straightforward.

Despite successful setup and execution, some **unsolved issues** remained. The **Laplace solver failed to converge** even after increasing the number of epochs to **2000**. This indicates potential issues with the update logic or boundary conditions. Further debugging is needed to identify and fix the convergence problem.

All the code is hereby attached and available on my github.