CS633 Assignment Group Number 27

Pavani Priya - 220415
Ruchita Rani - 241110059
Sontam Deekshitha - 221075
Vraj Patel - 241110080

April 14, 2025

1 Code Description

Our program implements a parallel solution for analyzing 3D time series data to find local minima/maxima counts and global min/max values for each time step. We used a 3D domain decomposition approach to divide the work among MPI processes. Our main algorithm works as follows:

1. Calculate each process's 3D coordinates based on its rank
2. Determine local subdomain size for each process
3. Identify neighbor processes in all six directions (±x, ±y, ±z)
4. Read data using parallel I/O (MPI-IO)
5. Exchange ghost cells with neighbors
6. Compute local minima/maxima and find global extrema
7. Reduce results to rank 0 and write output

2 Code Compilation and Execution Instructions

Our code can be compiled with any MPI C compiler. We used the following command:

```
mpicc -o src src.c -lm
```

To run the program, use the following command format:

```
mpirun -np P ./src input_file PX PY PZ NX NY NZ NC output_file
```

Where:

- P is the number of processes (must equal PX×PY×PZ)
- input_file is the input data filename
- PX, PY, PZ are the process grid dimensions
- NX, NY, NZ are the data grid dimensions
- NC is the number of time steps
- output_file is the output filename

We've also included the following helper scripts:

- `job.sh` - SLURM job script for running on clusters
- `JobScriptGen.py` - Python script to generate custom job scripts

- `PlotGen.py` - Performance analysis script for visualization

For example, to run with 8 processes for a 64×64×64 grid with 3 time steps:

```
mpirun -np 8 ./src data_64_64_64_3.bin 2 2 2 64 64 64 3 output_64_64_64_3_8.txt
```

3 Code Optimizations

- We replaced the sequential file reading with parallel I/O using MPI-IO, where each process directly reads its portion of data using MPI_File_open, custom MPI datatypes with MPI_Type_create_subarray, and MPI_File_read_all for better collective operations.
- We switched to MPI_Sendrecv to combine sending and receiving operations, and reduced memory allocation overhead by reusing communication buffers.
- We implemented padded local arrays with ghost cells to enhance cache utilization and eliminate boundary checking.
- We handled global domain edge cases by using MPI_PROC_NULL for non-existent neighbors, implementing conditional ghost exchanges, and adding special checks for extrema detection at boundaries.

4 Results

We analyzed the performance of our implementation using two test cases:

- 64×64×64 grid with 3 time steps
- 64×64×96 grid with 7 time steps

For each test case, we ran the code with 1, 2, 4, and 8 processes using different process grid configurations. Following Figures Visualizes Execution Time Reduction, Speedup etc varying with number of Processes

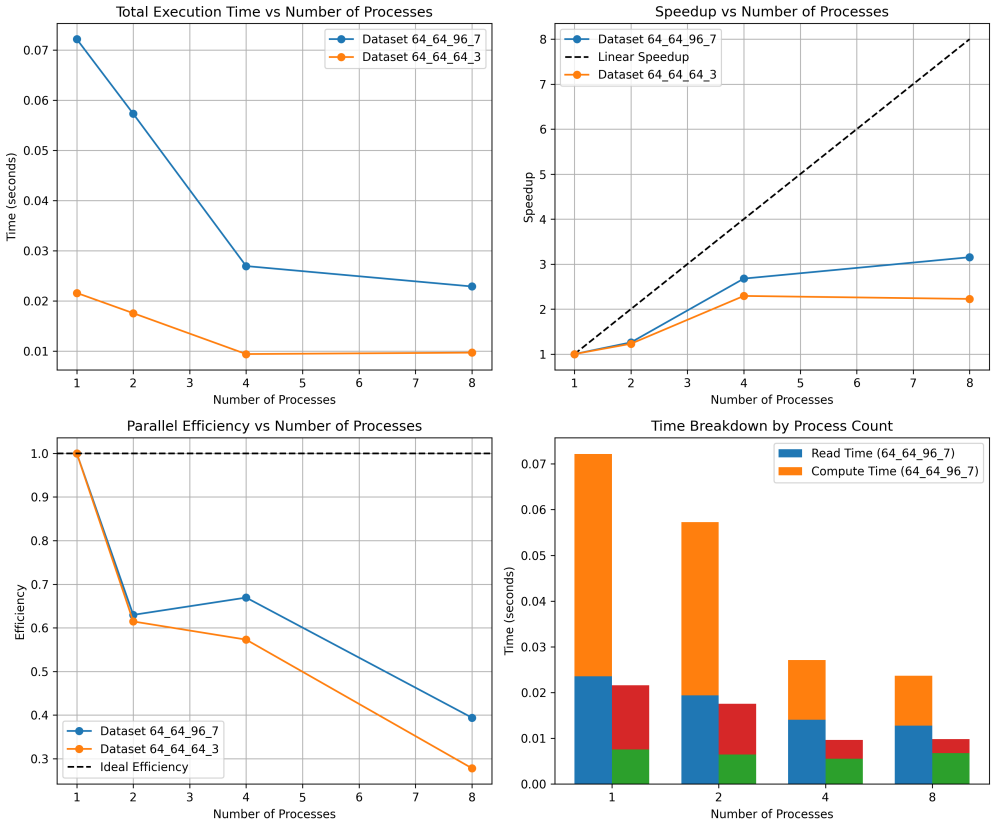

Figure 1: Performance analysis showing execution time, speedup, efficiency, and time breakdown

Following Table Shows Run Time, Compute Time, Total Time and SpeedUp for different Grid Size, Time Steps and number of processes

Table 1: Runtime Analysis for Different Grid Sizes and Process Counts

Grid Size	**Time Steps**	**Processes**	**Read Time (s)**	**Compute Time (s)**	**Total Time (s)**	**Speedup**
64×64×64	3	1	0.007552	0.014042	0.021595	1.00
		2	0.006449	0.011118	0.017567	1.23
		4	0.005506	0.004101	0.009422	2.29
		8	0.006756	0.003076	0.009704	2.23
64×64×96	7	1	0.023567	0.048581	0.072149	1.00
		2	0.019396	0.037891	0.057287	1.26
		4	0.014044	0.013092	0.026948	2.68
		8	0.012738	0.010914	0.022886	3.15

NOTE: Due to inconsistencies in Run Times on HPC data is collected on local machine.

5 Conclusions

Our implementation successfully parallelizes the detection of local minima/maxima and global extrema in 3D time series data.

The parallel I/O implementation improved performance compared to sequential I/O with data distribution. However, we observed diminishing returns beyond certain number of processes, suggesting that communication overhead and I/O contention become limiting factors at higher process counts.