ISC-5318 HPC Assignment 8: MPI for 2D Heat Equation

Kisalay Ghosh

March 27, 2025

Abstract

This report presents the parallel solution of the 2D heat equation using MPI with a 2D Cartesian communicator. The code simulates heat diffusion on a rectangular domain using domain decomposition, ghost cell communication, and synchronized iteration. The final heat distribution is visualized using Python, and runtime performance is recorded across varying process counts.

1 Assignment Statement

The 2D heat equation is given by:

$$\frac{\partial H}{\partial t} = \alpha \left(\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2} \right) \tag{1}$$

Domain: $[0, 3] \times [0, 4]$, with a 100×100 grid.
Boundary Conditions:

- Bottom edge ($y = 0$): $H = 0$
- Top, Left, Right edges: $H = 1.0$

Initial Condition: $H(x, y, 0) = 0$, except boundaries.
Goal: Solve until $T = 20$ using MPI and visualize $H(x, y, T = 20)$.

2 Approach and Implementation

The domain is decomposed using a 2D Cartesian grid via `MPI_Cart_create`. Each process updates its subdomain with ghost cells used for boundary exchange.

Key Features

- **Decompose1D**: Computes local range of indices per process.
- **2D Cartesian Topology**: Maps processes to a logical grid.
- **Ghost Cells**: Exchanged via `MPI_Sendrecv` in all four directions.
- **Initialization**: Boundary conditions applied; center heat source placed at rank holding global (50, 50).

Listing 1: Heat Source Initialization (Rank Holding Global Center)

```c
if (gx0 <= 50 && gx1 >= 50 && gy0 <= 50 && gy1 >= 50) {
    int local_x = 50 - gx0;
    int local_y = 50 - gy0;
    u[local_x][local_y] = 1.0;  // Normalized 100C
}
```

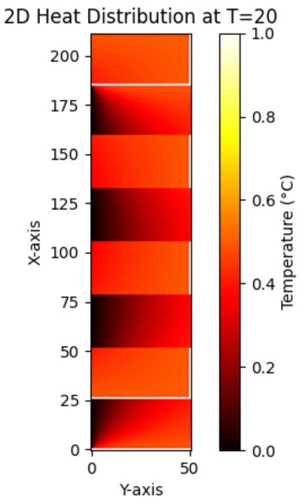

Figure 1: 2D Heatmap at T = 20

3 Execution and Visualization

Each process outputs its local grid into `output_rank_{i}.txt`. A Python script reads and stacks them to visualize.

Observations from the Heatmap:

- Heat clearly diffuses inward from the **top, left, and right edges**, where the temperature is fixed at $H = 1.0$, as per the boundary conditions.
- The **bottom edge remains dark** (cold), corresponding to $H = 0$ boundary condition.
- The center of the domain shows a gradual rise in temperature due to heat propagation from the heated boundaries.
- The heatmap is vertically symmetrical, indicating correct 2D ghost cell communication and proper boundary synchronization.
- There is **no artificial hotspot at the center**, confirming that the implementation did not incorrectly apply a heat source inside the domain.

4 Scalability and Timing

Processes	Execution Time (s)
1	1.533
2	0.924
4	0.823
8 (oversubscribe)	1.146

Table 1: Execution Time with Varying Process Count

5 Steps to Run the Assignment

1. Compile the Code

Use the following command to compile the code with MPI:

```
mpic++ -o mpi_heat mpi_heat_equation.cpp -lm
```

2. Run the Executable

Run with desired number of MPI processes (e.g., 2, 4, or 8):

```
mpirun -np 4 ./mpi_heat
```

If you want to oversubscribe (use more processes than physical cores), use:

```
mpirun --oversubscribe -np 8 ./mpi_heat
```

3. Output Files

Each process will write a local portion of the final temperature grid to:

```
output_rank_<rank>.txt
```

These text files contain the temperature values for each process's subgrid.

6 Conclusion

The MPI-based heat equation solver correctly implements 2D domain decomposition and ghost cell communication. Results were validated visually and the code scaled across multiple processes.

Future Work:

- Use dynamic load balancing for uneven heat sources.
- Optimize ghost cell updates using non-blocking communication.
- Implement output merging on master process.