HIGH PERFORMANCE COMPUTING

Heat Diffussion Equations
MPI

ZiHan, Chen

Yang, Hao

University of Lleida

Higher Polytechnic School

Master's Degree in Informatics Engineering

Course 2024 - 2025

18 May 2025

Contents

1 Introduction

This report presents the hybrid implementation of the two-dimensional heat diffusion equation using MPI for distributed-memory parallelism and OpenMP for shared-memory threading. Building on our previous serial and OpenMP-only solvers, we now decompose the global $N \times N$ grid across MPI ranks and further accelerate each subdomain with threads. We evaluate performance on a 1000×1000 grid over 10 000 time steps, comparing against the serial baseline to measure speedup, efficiency, load imbalance, and parallelization overhead.

2 Methodology

2.1 Domain Decomposition

A one-dimensional (row-wise) static decomposition splits the global grid across P MPI processes. Each rank receives either $\lfloor N/P \rfloor$ or $\lfloor N/P \rfloor + 1$ rows so that all rows are covered.

2.2 Halo Exchange

At each time step, neighbouring ranks exchange boundary rows (halos) via non-blocking `MPI_Isend`/`MPI_Irecv`, followed by `MPI_Waitall`. Only once halos arrive do we proceed to update boundary cells.

2.3 Hybrid Parallelism

Within each rank, the interior update loops use OpenMP `#pragma omp parallel for collapse(2)` directives. Halo communication and computation are separated to respect data dependencies.

3 Implementation

The hybrid solver is implemented in `heat_mpi.c`. Key steps:

1. Initialize MPI with thread support via `MPI_Init_thread`.
2. Parse input parameters (grid size, time steps) and compute $r = \alpha\, \Delta t / \Delta x^2$.
3. Determine local row counts and offsets per rank.
4. Allocate local and temporary arrays; initialize the diagonal heat source.
5. In each time step, call `solve_heat_equation_hybrid()`:
   - Post non-blocking halo Isend/Irecv.

- Update interior points with OpenMP collapse.
- Wait for halos and update boundary rows.
- Swap grids for next iteration.

6. On rank 0, gather final grid with `MPI_Gatherv` and write a BMP image.
7. Rank 0 logs total execution time (via `MPI_Wtime`).

A shell script (`heat_mpi.sh`) automates benchmarks across 2, 4, 8, 16, and 32 ranks.

4 Results

Performance measurements were carried out on grids of size 100×100, 1000×1000 and 2000×2000 over 10 000 time steps. We compare the pure serial execution to the hybrid MPI+OpenMP implementation using 2, 4, 8, 16 and 32 MPI processes (each spawning OpenMP threads).

Execution Time Comparison

Table 1 reports the raw execution times (in seconds) for both serial and hybrid runs.

Table 1: Execution Times (s) for Serial and Hybrid MPI+OpenMP Runs

Size	Steps	Serial	2 proc	4 proc	8 proc	16 proc	32 proc
100	100	0.000000	0.069873	0.124906	0.013887	0.025570	0.903717
100	1 000	0.020000	0.098586	0.361923	0.084249	0.084800	7.218292
100	10 000	0.220000	0.887402	2.611719	0.779793	0.796152	64.846480
100	100 000	2.130000	2.458472	40.277947	7.738679	7.860102	649.839919
1 000	100	0.340000	0.408743	0.652464	0.211907	0.553936	1.050846
1 000	1 000	3.540000	4.332085	5.900651	1.004584	1.640179	7.893415
1 000	10 000	25.250000	26.999726	42.639749	7.327339	5.981825	70.589969
1 000	100 000	278.160000	406.873956	459.596351	68.896344	43.400104	714.476561
2 000	100	1.160000	2.173718	0.933115	0.808209	0.811837	2.243320
2 000	1 000	13.640000	17.777219	6.169584	3.585566	3.138372	8.769729
2 000	10 000	135.670000	171.976954	58.481932	31.427259	19.596315	79.212660
2 000	100 000	1061.740000	598.118427	576.672416	295.147282	158.346738	770.509922

Speedup Relative to Serial

We define speedup as

$$\text{Speedup} = \frac{\text{Serial Time}}{\text{Parallel Time}}.$$

Table 2 shows the computed speedups for each hybrid configuration.

Table 2: Speedup of Hybrid MPI+OpenMP over Serial Baseline

Size	**Steps**	**2 proc**	**4 proc**	**8 proc**	**16 proc**	**32 proc**
100	100	N/A	N/A	N/A	N/A	N/A
100	1 000	0.203	0.055	0.237	0.236	0.003
100	10 000	0.248	0.084	0.282	0.276	0.003
100	100 000	0.866	0.053	0.275	0.271	0.003
1 000	100	0.832	0.521	1.604	0.614	0.324
1 000	1 000	0.817	0.600	3.524	2.158	0.448
1 000	10 000	0.935	0.592	3.446	4.221	0.358
1 000	100 000	0.684	0.605	4.037	6.409	0.389
2 000	100	0.534	1.243	1.435	1.429	0.517
2 000	1 000	0.767	2.211	3.804	4.346	1.555
2 000	10 000	0.789	2.320	4.317	6.923	1.713
2 000	100 000	1.775	1.841	3.597	6.705	1.378

Parallel Efficiency

Parallel efficiency is defined as

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processes}} \times 100\%.$$

Table 3 summarises the efficiency for each configuration.

Table 3: Parallel Efficiency (%) of Hybrid MPI+OpenMP

Size	**Steps**	**2 proc**	**4 proc**	**8 proc**	**16 proc**	**32 proc**
100	100	N/A	N/A	N/A	N/A	N/A
100	1 000	10.1	1.4	3.0	1.5	0.01
100	10 000	12.4	2.1	3.5	1.7	0.01
100	100 000	43.3	1.3	3.4	1.7	0.01
1 000	100	41.6	13.0	20.1	3.8	1.0
1 000	1 000	41.7	15.0	44.0	13.5	1.4
1 000	10 000	46.8	14.8	43.1	26.4	1.1
1 000	100 000	34.2	15.1	50.5	40.1	1.2
2 000	100	26.7	31.1	17.9	8.9	1.6
2 000	1 000	38.4	55.3	47.6	27.2	4.9
2 000	10 000	39.4	58.0	54.0	43.3	5.4
2 000	100 000	88.8	46.0	45.0	41.9	4.3

Parallelization Overhead

We define overhead as

$$\text{Overhead} = \frac{T_{\text{measured}} - \frac{T_{\text{serial}}}{\text{Processes}}}{\frac{T_{\text{serial}}}{\text{Processes}}} \times 100\%.$$

Table 4 reports the percentage overhead for each hybrid configuration.

Table 4: Parallelization Overhead (%) for Hybrid MPI+OpenMP over Serial Baseline

Size	Steps	2 proc	4 proc	8 proc	16 proc	32 proc
100	100	N/A	N/A	N/A	N/A	N/A
100	1 000	885.9	7138.5	3270.0	6684.0	1154827.0
100	10 000	707.6	4640.4	27366.1	56904.2	942171.0
100	100 000	130.8	7467.4	2806.9	5807.4	975868.0
1 000	100	140.4	667.6	398.6	25063.0	9799.0
1 000	1 000	144.7	566.5	127.0	641.1	7036.0
1 000	10 000	113.9	575.5	132.2	279.0	8846.0
1 000	100 000	192.6	561.1	98.2	149.6	8119.0
2 000	100	274.8	221.8	457.0	1019.1	6081.0
2 000	1 000	160.6	80.9	110.3	268.2	1958.0
2 000	10 000	153.5	72.4	85.3	131.1	1768.0
2 000	100 000	12.7	117.3	122.4	138.6	2224.0

5 Discussion

The hybrid MPI+OpenMP solver shows:

- **Startup overhead at 2 ranks:** Speedup of 0.94×, indicating that MPI setup and minimal halo traffic slightly outweigh partitioned computation.
- **Peak at 16 ranks:** Maximum speedup of 4.22× (26% efficiency) demonstrates useful parallel gain, though sub-linear.
- **Inefficiency at 32 ranks:** Speedup collapses to 0.36×. Each rank's subdomain is too small, so halo exchange dominates compute time.
- **Parallel overhead:** Even at 8 ranks, ideal time is $25.25/8 = 3.16\,\text{s}$ vs. measured 7.33 s, an overhead of $(7.33 - 3.16)/3.16 \approx 132\%$ due to communication and threading costs.

These results highlight a "sweet spot" around 8–16 ranks. Beyond that, the communication-to-compute ratio grows too large. A 2D decomposition or dynamic load balancing could improve scaling.

6 Conclusion

We extended our OpenMP solver into a hybrid MPI+OpenMP model. While speedup peaks at 16 ranks, overall efficiency remains modest due to communication and synchronization overheads. For optimal performance on this problem size, 8–16 ranks are recommended. Future work will investigate multi-dimensional domain decomposition and adaptive load distribution to lower parallel overheads.