CI Pathway: Parallel Computing

Assignment: Distributed Memory Parallelism

UCLA, Statistics
Hochan Son
Summer 2025
July 6, 2025

1 Introduction

Parallel computing is a critical component in modern scientific computing, enabling the solution of large-scale problems by distributing workloads across multiple processors. This report focuses on distributed memory parallelism using the Message Passing Interface (MPI) to solve the Laplace equation, a common benchmark in numerical analysis and scientific computing. The assignment explores the implementation and optimization of parallel algorithms for the Laplace solver, evaluates their performance on a high-performance computing (HPC) cluster, and analyzes the impact of various optimization strategies. Using dynamic memory allocation and non-blocking communication, the implementation aims to improve computational efficiency and scalability. The report presents a detailed performance analysis, including convergence rates, execution times, speedup factors, and parallel efficiency across different configurations. The results demonstrate the effectiveness of the optimizations and provide insights into the performance characteristics of distributed memory parallel applications.

2 Hardware Environment

2.1 System Specifications

The experiments were conducted on the NCSA Delta HPC cluster, which is a high-performance computing environment designed for parallel processing tasks. The specifications of the system are as follows:

Table 1: NCSA Delta Compute Environment

Component	**Specification**
Compute Platform	NCSA Delta HPC Cluster
Login Node	dt-login04.delta.ncsa.illinois.edu
Compute Node	cn094.delta.ncsa.illinois.edu
Operating System	Linux 4.18.0-477.95.1.el8_8.x86_64
Distribution	Red Hat Enterprise Linux 8
Architecture	x86_64
Node Interconnect	HPE Slingshot

2.2 Processor Architecture

Table 2: AMD EPYC 7763 Processor Specifications

Parameter	**Value**
CPU Model	AMD EPYC 7763 64-Core Processor
Architecture	AMD Zen 3 (Milan)
Physical Cores	64 per socket
Hardware Threads	128 (2-way SMT)
Base Clock	2.45 GHz
Boost Clock	Up to 3.5 GHz
Manufacturing Process	7nm TSMC
Socket Type	SP3

3 Exercises For This Module.

Write a code that runs on 8 PEs and does a `"circular shift."` This means that every PE sends some data to its nearest neighbor either `"up"` (one PE higher) or `"down."` To make it circular, PE 7 and PE 0 are treated as neighbors. Make sure that whatever data you send is received. The easiest way to do that is to have each PE print out its received messages.

3.1 Exercise Notes

To compile with MPI we do either:

```
mpicc laplace_mpi.c
```

```
or

mpif90 laplace_mpi.f90
```

You will have an executable called a.out. Now you need to ask for a compute node with 8 processes allocated in order to run. Similar, but not identical, to our previous Slurm command:

```
srun --account=becs-delta-cpu --partition=cpu-interactive \
  --nodes=1 --tasks=8 --tasks-per-node=8 --pty bash
```

And we wish to run using 8 processes. The command to run our a.out executable or available process is

```
mpirun -n 8 a.out
```

Submit a copy of your code. Any output might be helpful, too.

4 Solution Implementation

4.1 original code, `hw3_laplace_mpi.c`

- The original, laplace_mpi.c, is a simple MPI program that demonstrates among multiple processes. The code initializes the MPI environment, allocates an array for each process, and performs the shift by sending data to neighboring processes (PE). Each process prints the data it receives from its neighbor. The code is structured to run on multiple processors, allowing for parallel execution of the circular shift operation. However, it has fixed values for the number of processes and the size of the data array, which may not be suitable for dynamic mpi applications.

```
#define NPES            4       // number of processors (fixed)
```

4.2 1-dimension, `hw3_laplace_mpi_1d.c`

- **Non-blocking communication with computation overlap:** The code uses `MPI_Irecv` and `MPI_Isend` to allow computation and communication to overlap, improving efficiency.

```
MPI_Irecv(&Temperature[0][1], COLUMNS, MPI_DOUBLE, up_PE, 0,
    MPI_COMM_WORLD, &request);
MPI_Isend(&Temperature[my_rows][1], COLUMNS, MPI_DOUBLE, down_PE, 0,
    MPI_COMM_WORLD, &request);
```

- **Dynamic memory allocation (1D contiguous memory allocation):** The code dynamically allocates memory for the temperature arrays, allowing for flexible problem sizes.

```
// Dynamic memory allocation for 1D arrays
double **Temperature;
double **Temperature_last;
Temperature = malloc((my_rows + 2) * sizeof(double*));
Temperature_last = malloc((my_rows + 2) * sizeof(double*));
```

4.3 2-Simension, `laplace_mpi_2d.c`

- **Non-blocking communication with computation overlap:** Similar to the 1D version, but adapted for 2D arrays.

```
MPI_Irecv(&Temperature[0][1], COLUMNS, MPI_DOUBLE, up_PE, 0,
    MPI_COMM_WORLD, &request);
MPI_Isend(&Temperature[my_rows][1], COLUMNS, MPI_DOUBLE, down_PE, 0,
    MPI_COMM_WORLD, &request);
```

- **Dynamic memory allocation (2D contiguous memory allocation):** Uses pointer arithmetic to allocate a contiguous block for the 2D array.

```
// Dynamic memory allocation for 2D arrays
double (*Temperature)[COLUMNS+2];
double (*Temperature_last)[COLUMNS+2];

// Allocate dynamic memory
Temperature = (double (*)[COLUMNS+2])malloc((my_rows+2) * (COLUMNS+2) *
    sizeof(double));
Temperature_last = (double (*)[COLUMNS+2])malloc((my_rows+2) * (COLUMNS
    +2) * sizeof(double));
```

- **Dynamic swapping memory allocation:** Efficiently swaps pointers for the next iteration.

```
// Swap pointers for next iteration
double (*temp)[COLUMNS+2] = Temperature;
Temperature = Temperature_last;
Temperature_last = temp;
```

4.4 finally, laplace_mpi_2d_optimized.c

- The final optimized version of the MPI Laplace solver combines the 1D and 2D optimizations, utilizing non-blocking communication, dynamic memory allocation, and efficient data swapping techniques. This version is designed to handle larger grids and improve performance by reducing communication overhead and enhancing computational efficiency. The code structure allows for easy scalability across multiple processes, making it suitable for high-performance computing environments.

- Circular shift operation is implemented using MPI for parallel processing. Each process sends its data to its nearest neighbor, either up or down, and receives data from its neighbor. The circular nature of the shift is maintained by treating the first and last processes as neighbors. The implementation uses non-blocking communication to allow for overlap between computation and communication, improving overall performance.

```
int next_PE, prev_PE;
// Calculate ring neighbors
next_PE = (my_PE_num + 1) % npes;
prev_PE = (my_PE_num - 1 + npes) % npes;
// Dynamically Calculate local dimensions
int rows_per_process = ROWS_GLOBAL / npes;
int ghost_rows = ROWS_GLOBAL % npes;
 // for even distribution of the ghost_rows in case the rows are not
     exactly divisible by process X.
int ROWS = rows_per_process + (my_PE_num < ghost_rows ? 1 : 0);

// Circular shift operation
MPI_Sendrecv(&data[0], 1, MPI_DOUBLE, up_PE, 0, &data[my_rows-1], 1,
    MPI_DOUBLE, down_PE, 0, MPI_COMM_WORLD, &status);
MPI_Sendrecv(&data[my_rows-1], 1, MPI_DOUBLE, down_PE, 0, &data[0], 1,
    MPI_DOUBLE, up_PE, 0, MPI_COMM_WORLD, &status);
```

- Free memory after use

```
      // Free memory after all of the communication has finished
for (int i = 0; i < ROWS + 2; i++) {
    free(Temperature[i]);
    free(Temperature_last[i]);
}
free(Temperature);
free(Temperature_last);
```

5 Performance Analysis

Table 3: MPI Laplace Solver Performance Results

Implementation	Processes	Iterations	Converged	Final Error	Time (s)	Speedup	Efficiency	Iter/sec
Original	4	3372	✓	0.009995	6.039	--	--	558.412
1D Optimized	1	3372	✓	0.009995	21.855	1.000	1.000	154.293
1D Optimized	4	3372	✓	0.009995	6.079	3.595	0.899	554.667
1D Optimized	8	3372	✓	0.009995	3.667	5.960	0.745	919.589
2D Optimized	1	4000	×	20.922598	26.028	1.000	1.000	153.682
2D Optimized	4	4000	×	20.922598	7.397	3.519	0.880	540.786
2D Optimized	8	4000	×	20.922598	4.142	6.284	0.786	965.766
Final Optimized	1	3602	✓	0.009998	23.325	1.000	1.000	154.424
Final Optimized	4	3602	✓	0.009998	6.707	3.478	0.869	537.055
Final Optimized	8	3449	✓	0.009999	3.674	6.349	0.794	938.764

6 Summary Statistics

6.1 Convergence Analysis

Table 4: Convergence Analysis by Implementation

Implementation	Total Tests	Converged Tests	Avg Time (s)	Convergence Rate (%)
1D Optimized	3	3	10.534	100.0
2D Optimized	3	0	12.522	0.0
Final Optimized	3	3	11.235	100.0

6.2 Performance Analysis (Converged Tests Only)

Table 5: Performance Metrics for Successfully Converged Tests

Implementation	Min Time	Max Time	Mean Time	Max Speedup	Mean Efficiency	Max Throughput
1D Optimized	3.667	21.855	10.534	5.960	0.881	919.589
Final Optimized	3.674	23.325	11.235	6.349	0.888	938.764

7 Key Performance Insights

- **Best Overall Performance:** Final Optimized implementation with 8 processes achieved the fastest execution time of 3.674 s
- **Highest Speedup:** Final Optimized with 8 processes demonstrated a speedup factor of **6.349**× compared to single-process execution

- **Best Parallel Efficiency:** 1D Optimized with 4 processes achieved **89.9%** parallel efficiency
- **Convergence Issues:** The 2D Optimized implementation failed to converge in all test configurations, reaching the maximum iteration limit of 4000
- **Scalability Trends:** All successfully converging implementations show good scalability up to 8 processes, though efficiency decreases with higher process counts (typical parallel computing behavior)
- **Throughput Champion:** Final Optimized with 8 processes achieved the highest computational throughput of 938.764 iterations/s

8 Conclusions

The analysis reveals that both the 1D Optimized and Final Optimized implementations demonstrate excellent parallel performance characteristics. The 2D Optimized approach, while showing good speedup trends, suffers from convergence issues that prevent practical application. The study demonstrates the importance of algorithmic correctness alongside performance optimization in parallel computing applications.

9 Appendix.code

9.1 Code A: hw3_laplace_mpi_1.c

```
/**********************************************************************
 * Laplace MPI C Version
 *
 * T is initially 0.0
 * Boundaries are as follows
 *
 *                T                      4 sub-grids
 *   0  +--------------------+  0    +--------------------+
 *      |                    |       |                    |
 *      |                    |       |--------------------|
 *      |                    |       |                    |
 *   T  |                    |  T    |--------------------|
 *      |                    |       |                    |
 *      |                    |       |--------------------|
 *      |                    |       |                    |
 *   0  +--------------------+ 100   +--------------------+
 *      0         T        100
 *
 * Each PE only has a local subgrid.
 * Each PE works on a sub grid and then sends
 * its boundaries to neighbors.
 *
 *  John Urbanic, PSC 2014
 *
 **********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <mpi.h>

#define COLUMNS      1000
#define ROWS_GLOBAL  1000        // this is a "global" row count
#define NPES            4        // number of processors
#define ROWS (ROWS_GLOBAL/NPES)  // number of real local rows

// communication tags
#define DOWN     100
#define UP       101

#define MAX_TEMP_ERROR 0.01

double Temperature[ROWS+2][COLUMNS+2];
double Temperature_last[ROWS+2][COLUMNS+2]; //padding for ghost cells

void initialize(int npes, int my_PE_num);
void track_progress(int iter);
```

```
int main(int argc, char *argv[]) {

    int i, j;
    int max_iterations;
    int iteration=1;
    double dt;
    struct timeval start_time, stop_time, elapsed_time;

    int        npes;                // number of PEs
    int        my_PE_num;           // my PE number
    double     dt_global=100;       // delta t across all PEs
    MPI_Status status;              // status returned by MPI calls

    // the usual MPI startup routines
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);

    // verify only NPES PEs are being used
    if(npes != NPES) {
      if(my_PE_num==0) {
        printf("This code must be run with %d PEs\n", NPES);
      }
      MPI_Finalize();
      exit(1);
    }

    // PE 0 asks for input
    if(my_PE_num==0) {
      printf("Maximum iterations [100-4000]?\n");
      fflush(stdout); // Not always necessary, but can be helpful
      scanf("%d", &max_iterations);
    }

    // bcast max iterations to other PEs
    MPI_Bcast(&max_iterations, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (my_PE_num==0) gettimeofday(&start_time,NULL);

    initialize(npes, my_PE_num);

    while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

        // main calculation: average my four neighbors
        for(i = 1; i <= ROWS; i++) {
            for(j = 1; j <= COLUMNS; j++) {
                Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
                    Temperature_last[i-1][j] +
                                            Temperature_last[i][j+1] +
                                                Temperature_last[i][j-1]);
            }
```

```
    }

    // COMMUNICATION PHASE: send ghost rows for next iteration

    // send bottom real row down
    if(my_PE_num != npes-1){              //unless we are bottom PE
        MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, my_PE_num+1, DOWN
            , MPI_COMM_WORLD);
    }

    // receive the bottom row from above into our top ghost row
    if(my_PE_num != 0){                   //unless we are top PE
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1,
            DOWN, MPI_COMM_WORLD, &status);
    }

    // send top real row up
    if(my_PE_num != 0){                     //unless we are top PE
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP,
            MPI_COMM_WORLD);
    }

    // receive the top row from below into our bottom ghost row
    if(my_PE_num != npes-1){              //unless we are bottom PE
        MPI_Recv(&Temperature_last[ROWS+1][1], COLUMNS, MPI_DOUBLE, my_PE_num
            +1, UP, MPI_COMM_WORLD, &status);
    }

    dt = 0.0;

    for(i = 1; i <= ROWS; i++){
        for(j = 1; j <= COLUMNS; j++){
            dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }

    // find global dt
    MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // periodically print test values - only for PE in lower corner
    if((iteration % 100) == 0) {
        if (my_PE_num == npes-1){
            track_progress(iteration);
        }
    }

    iteration++;
}

// Slightly more accurate timing and cleaner output
MPI_Barrier(MPI_COMM_WORLD);
```

```
    // PE 0 finish timing and output values
    if (my_PE_num==0){
        gettimeofday(&stop_time,NULL);
        timersub(&stop_time, &start_time, &elapsed_time);

        printf("\nMax error at iteration %d was %f\n", iteration-1, dt_global);
        printf("Total time was %f seconds.\n", elapsed_time.tv_sec+elapsed_time.
            tv_usec/1000000.0);
    }

    MPI_Finalize();
}



void initialize(int npes, int my_PE_num){

    double tMin, tMax;  //Local boundary limits
    int i,j;

    for(i = 0; i <= ROWS+1; i++){
        for (j = 0; j <= COLUMNS+1; j++){
            Temperature_last[i][j] = 0.0;
        }
    }

    // Local boundry condition endpoints
    tMin = (my_PE_num)*100.0/npes;
    tMax = (my_PE_num+1)*100.0/npes;

    // Left and right boundaries
    for (i = 0; i <= ROWS+1; i++) {
      Temperature_last[i][0] = 0.0;
      Temperature_last[i][COLUMNS+1] = tMin + ((tMax-tMin)/ROWS)*i;
    }

    // Top boundary (PE 0 only)
    if (my_PE_num == 0)
      for (j = 0; j <= COLUMNS+1; j++)
        Temperature_last[0][j] = 0.0;

    // Bottom boundary (Last PE only)
    if (my_PE_num == npes-1)
      for (j=0; j<=COLUMNS+1; j++)
        Temperature_last[ROWS+1][j] = (100.0/COLUMNS) * j;

}


// only called by last PE
void track_progress(int iteration) {
```

```
    int i;

    printf("----------␣Iteration␣number:␣%d␣-----------\n", iteration);

    // output global coordinates so user doesn't␣have␣to␣understand␣decomposition
␣␣␣␣for(i␣=␣5;␣i␣>=␣0;␣i--)␣{
␣␣␣␣␣␣printf("[%d,%d]:␣%5.2f␣␣",␣ROWS_GLOBAL-i,␣COLUMNS-i,␣Temperature[ROWS-i][
    COLUMNS-i]);
␣␣␣␣}
␣␣␣␣printf("\n");
}
```

9.2 Code B: hw3_laplace_mpi_2.c

```
/*************************************************************
 * Complete Working 1D Linear Laplace MPI Solver
 * Project: CI Pathway Summer 2025
 * Course: Parallel Programming
 * Assignment: Distributed Memory Parallelism
 * Original Author: John Urbanic, PSC 2014
 * Modified by: Hochan Son, UCLA, Statistics
 * Date: 2025-07-05
 *
 * COMPLETE WORKING VERSION with:
 * - Correct boundary conditions (matches original exactly)
 * - Linear topology (no circular communication)
 * - Dynamic process count support
 * - Proper convergence checking
 * - Fixed ghost cell communication
 *************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <mpi.h>

#define COLUMNS      1000
#define ROWS_GLOBAL  1000
#define MAX_TEMP_ERROR 0.01

// Communication tags
#define DOWN     100
#define UP       101

// Global variables for temperature arrays
double **Temperature;
double **Temperature_last;

// Function prototypes
void initialize(int npes, int my_PE_num, int my_rows);
void track_progress(int iteration, int my_rows, int npes, int my_PE_num);
```

```
int main(int argc, char *argv[]) {
    int i, j;
    int max_iterations;
    int iteration = 1;
    double dt;
    struct timeval start_time, stop_time, elapsed_time;

    int npes;                // number of PEs
    int my_PE_num;           // my PE number
    double dt_global = 100;  // delta t across all PEs
    MPI_Status status;       // status returned by MPI calls

    // MPI startup routines
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);

    // Calculate dynamic local dimensions
    int rows_per_process = ROWS_GLOBAL / npes;
    int ghost_rows = ROWS_GLOBAL % npes;

    // Handle uneven division
    int my_rows = rows_per_process;
    if (my_PE_num < ghost_rows) {
        my_rows++;  // First 'ghost_rows' processes get one ghost row
    }

    if (my_PE_num == 0) {
        printf("===␣Complete␣Working␣1D␣Linear␣Laplace␣MPI␣Solver␣===\n");
        printf("Running␣with␣%d␣processes\n", npes);
        printf("Grid␣size:␣%d␣x␣%d\n", ROWS_GLOBAL, COLUMNS);
        printf("Process␣0␣managing␣%d␣rows\n", my_rows);
    }

    // Dynamic memory allocation for 1D arrays
    Temperature = malloc((my_rows + 2) * sizeof(double*));
    Temperature_last = malloc((my_rows + 2) * sizeof(double*));

    for(i = 0; i < my_rows + 2; i++) {
        Temperature[i] = malloc((COLUMNS + 2) * sizeof(double));
        Temperature_last[i] = malloc((COLUMNS + 2) * sizeof(double));
    }

    // PE 0 asks for input
    if(my_PE_num == 0) {
        printf("Maximum␣iterations␣[100-4000]?\n");
        fflush(stdout);
        scanf("%d", &max_iterations);
    }

    // Broadcast max iterations to other PEs
    MPI_Bcast(&max_iterations, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
if (my_PE_num == 0) gettimeofday(&start_time, NULL);

// Initialize boundary conditions
initialize(npes, my_PE_num, my_rows);

// Main iteration loop
while (dt_global > MAX_TEMP_ERROR && iteration <= max_iterations) {

    // === MAIN CALCULATION: average my four neighbors ===
    for(i = 1; i <= my_rows; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
                Temperature_last[i-1][j] +
                                        Temperature_last[i][j+1] +
                                            Temperature_last[i][j-1]);
        }
    }

    // === COMMUNICATION PHASE: send ghost rows for next iteration ===

    // Send bottom real row down
    if(my_PE_num != npes-1) {  // unless we are bottom PE
        MPI_Send(&Temperature[my_rows][1], COLUMNS, MPI_DOUBLE, my_PE_num+1,
            DOWN, MPI_COMM_WORLD);
    }

    // Receive the bottom row from above into our top ghost row
    if(my_PE_num != 0) {  // unless we are top PE
        MPI_Recv(&Temperature_last[0][1], COLUMNS, MPI_DOUBLE, my_PE_num-1,
            DOWN, MPI_COMM_WORLD, &status);
    }

    // Send top real row up
    if(my_PE_num != 0) {  // unless we are top PE
        MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, my_PE_num-1, UP,
            MPI_COMM_WORLD);
    }

    // Receive the top row from below into our bottom ghost row
    if(my_PE_num != npes-1) {  // unless we are bottom PE
        MPI_Recv(&Temperature_last[my_rows+1][1], COLUMNS, MPI_DOUBLE,
            my_PE_num+1, UP, MPI_COMM_WORLD, &status);
    }

    // === CONVERGENCE CHECK ===
    dt = 0.0;
    for(i = 1; i <= my_rows; i++) {
        for(j = 1; j <= COLUMNS; j++) {
            dt = fmax(fabs(Temperature[i][j] - Temperature_last[i][j]), dt);
            Temperature_last[i][j] = Temperature[i][j];
        }
    }
```

```
        // Find global dt
        MPI_Reduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
        MPI_Bcast(&dt_global, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

        // Periodically print test values - only for PE in lower corner
        if((iteration % 100) == 0) {
            if (my_PE_num == npes-1) {
                track_progress(iteration, my_rows, npes, my_PE_num);
            }
            if (my_PE_num == 0) {
                printf("Iteration %d: dt_global = %f\n", iteration, dt_global);
            }
        }

        iteration++;
    }

    // Synchronize for accurate timing
    MPI_Barrier(MPI_COMM_WORLD);

    // PE 0 finish timing and output values
    if (my_PE_num == 0) {
        gettimeofday(&stop_time, NULL);
        timersub(&stop_time, &start_time, &elapsed_time);

        printf("\n============== RESULTS ==============\n");
        if (dt_global <= MAX_TEMP_ERROR) {
            printf("   CONVERGED after %d iterations\n", iteration-1);
        } else {
            printf("   Did NOT converge after %d iterations\n", iteration-1);
        }
        printf("Final error: %f\n", dt_global);
        printf("Total time: %f seconds\n", elapsed_time.tv_sec + elapsed_time.
            tv_usec/1000000.0);
        printf("=====================================\n");
    }

    // Free memory
    for(i = 0; i < my_rows + 2; i++) {
        free(Temperature[i]);
        free(Temperature_last[i]);
    }
    free(Temperature);
    free(Temperature_last);

    MPI_Finalize();
    return 0;
}

void initialize(int npes, int my_PE_num, int my_rows) {
    double tMin, tMax;  // Local boundary limits
    int i, j;
```

```
    // Initialize all points to 0.0
    for(i = 0; i <= my_rows + 1; i++) {
        for (j = 0; j <= COLUMNS + 1; j++) {
            Temperature_last[i][j] = 0.0;
        }
    }

    //     CRITICAL: Correct boundary condition calculation
    // This EXACTLY matches the original working version
    tMin = (my_PE_num) * 100.0 / npes;
    tMax = (my_PE_num + 1) * 100.0 / npes;

    // Left and right boundaries
    for (i = 0; i <= my_rows + 1; i++) {
        Temperature_last[i][0] = 0.0;  // Left boundary = 0 C
        Temperature_last[i][COLUMNS+1] = tMin + ((tMax - tMin) / my_rows) * i;  //
             Right boundary: linear gradient
    }

    // Top boundary (PE 0 only)
    if (my_PE_num == 0) {
        for (j = 0; j <= COLUMNS + 1; j++) {
            Temperature_last[0][j] = 0.0;  // Top boundary = 0 C
        }
    }

    // Bottom boundary (Last PE only)
    if (my_PE_num == npes - 1) {
        for (j = 0; j <= COLUMNS + 1; j++) {
            Temperature_last[my_rows + 1][j] = (100.0 / COLUMNS) * j;  // Bottom:
                0 C  to 100 C
        }
    }

    printf("Process␣%d:␣initialized␣boundaries␣(tMin=%.1f,␣tMax=%.1f,␣rows=%d)\n",
           my_PE_num, tMin, tMax, my_rows);
}

void track_progress(int iteration, int my_rows, int npes, int my_PE_num) {
    int i;

    printf("----------␣Iteration␣number:␣%d␣------------\n", iteration);

    // Calculate global coordinates for display
    // This matches the original algorithm exactly
    int rows_per_process = ROWS_GLOBAL / npes;
    int ghost_rows = ROWS_GLOBAL % npes;

    int global_start_row = my_PE_num * rows_per_process;
    if (my_PE_num < ghost_rows) {
        global_start_row += my_PE_num;
    } else {
```

```
        global_start_row += ghost_rows;
    }

    // Output global coordinates so user doesn't have to understand decomposition
    for(i = 5; i >= 0; i--) {
        if (my_rows - i > 0) {
            int global_row = global_start_row + (my_rows - i);
            printf("[%d,%d]: %5.2f  ", global_row, COLUMNS - i, Temperature[
    my_rows - i][COLUMNS - i]);
        }
    }
    printf("\n");
}
```

9.3 Code C: hw3_laplace_mpi_3.c

```
/*************************************************************
 * 2 Dimension Optimized Laplace MPI C Version
 *
 * Performance Optimizations:
 * - Non-blocking communication with computation overlap
 * - Pointer swapping instead of array copying
 * - Loop fusion for better cache locality
 * - AllReduce instead of Reduce+Bcast
 * - Dynamic memory allocation for scalability
 * - Removed hardcoded processor count limitation
 *
 * T is initially 0.0
 * Boundaries are as follows
 *
 *                 T                       4 sub-grids
 *   0  +--------------------+  0    +--------------------+
 *      |                    |       |                    |
 *      |                    |       |--------------------|
 *      |                    |       |                    |
 *   T  |                    |  T    |--------------------|
 *      |                    |       |                    |
 *      |                    |       |--------------------|
 *      |                    |       |                    |
 *   0  +--------------------+ 100   +--------------------+
 *      0         T       100
 *
 * Each PE only has a local subgrid.
 * Each PE works on a sub grid and then sends
 * its boundaries to neighbors.
 *
 *************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
```

```c
#include <mpi.h>

#define COLUMNS      1000
#define ROWS_GLOBAL  1000        // this is a "global" row count

// communication tags
#define DOWN     100
#define UP       101

#define MAX_TEMP_ERROR 0.01

// Global pointers for dynamic arrays
double (*Temperature)[COLUMNS+2];
double (*Temperature_last)[COLUMNS+2];

void initialize(int npes, int my_PE_num, int my_rows);
void track_progress(int iteration, int my_rows);

int main(int argc, char *argv[]) {

    int i, j;
    int max_iterations;
    int iteration=1;
    double dt;
    struct timeval start_time, stop_time, elapsed_time;

    int        npes;                // number of PEs
    int        my_PE_num;           // my PE number
    double     dt_global=100;       // delta t across all PEs
    MPI_Status status;              // status returned by MPI calls
    MPI_Request requests[4];        // for non-blocking communication
    int        req_count;           // number of active requests

    // Dynamic row distribution
    int rows_per_process;
    int extra_rows;
    int my_rows;
    int my_start_row;

    // the usual MPI startup routines
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);

    // Calculate dynamic load balancing
    rows_per_process = ROWS_GLOBAL / npes;
    extra_rows = ROWS_GLOBAL % npes;

    // Distribute extra rows to first few processes
    if (my_PE_num < extra_rows) {
        my_rows = rows_per_process + 1;
        my_start_row = my_PE_num * my_rows;
    } else {
```

```
    my_rows = rows_per_process;
    my_start_row = extra_rows * (rows_per_process + 1) +
                   (my_PE_num - extra_rows) * rows_per_process;
}

// Allocate dynamic memory
Temperature = (double (*)[COLUMNS+2])malloc((my_rows+2) * (COLUMNS+2) * sizeof
    (double));
Temperature_last = (double (*)[COLUMNS+2])malloc((my_rows+2) * (COLUMNS+2) *
    sizeof(double));

if (!Temperature || !Temperature_last) {
    printf("PE␣%d:␣Memory␣allocation␣failed\n", my_PE_num);
    MPI_Finalize();
    exit(1);
}

// PE 0 asks for input
if(my_PE_num==0) {
    printf("Maximum␣iterations␣[100-4000]?\n");
    printf("Running␣on␣%d␣processes␣with␣dynamic␣load␣balancing\n", npes);
    fflush(stdout);
    scanf("%d", &max_iterations);
}

// bcast max iterations to other PEs
MPI_Bcast(&max_iterations, 1, MPI_INT, 0, MPI_COMM_WORLD);

if (my_PE_num==0) gettimeofday(&start_time,NULL);

initialize(npes, my_PE_num, my_rows);

while ( dt_global > MAX_TEMP_ERROR && iteration <= max_iterations ) {

    // PHASE 1: Start non-blocking communication for ghost rows
    req_count = 0;

    // Send bottom real row down and receive top ghost row
    if(my_PE_num != npes-1) {
        MPI_Isend(&Temperature_last[my_rows][1], COLUMNS, MPI_DOUBLE,
                  my_PE_num+1, DOWN, MPI_COMM_WORLD, &requests[req_count++]);
    }
    if(my_PE_num != 0) {
        MPI_Irecv(&Temperature[0][1], COLUMNS, MPI_DOUBLE,
                  my_PE_num-1, DOWN, MPI_COMM_WORLD, &requests[req_count++]);
    }

    // Send top real row up and receive bottom ghost row
    if(my_PE_num != 0) {
        MPI_Isend(&Temperature_last[1][1], COLUMNS, MPI_DOUBLE,
                  my_PE_num-1, UP, MPI_COMM_WORLD, &requests[req_count++]);
    }
    if(my_PE_num != npes-1) {
```

```
            MPI_Irecv(&Temperature[my_rows+1][1], COLUMNS, MPI_DOUBLE,
                      my_PE_num+1, UP, MPI_COMM_WORLD, &requests[req_count++]);
        }

        // PHASE 2: Calculate interior points (can overlap with communication)
        // Interior points don't need ghost cells
        for(i = 2; i < my_rows; i++) {
            for(j = 1; j <= COLUMNS; j++) {
                Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
                    Temperature_last[i-1][j] +
                                            Temperature_last[i][j+1] +
                    Temperature_last[i][j-1]);
            }
        }

        // PHASE 3: Wait for communication completion
        if (req_count > 0) {
            MPI_Waitall(req_count, requests, MPI_STATUSES_IGNORE);
        }

        // PHASE 4: Calculate boundary rows that need ghost cells
        // Top boundary row (row 1)
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[1][j] = 0.25 * (Temperature_last[2][j] + Temperature[0][j]
                + Temperature_last[1][j+1] + Temperature_last
                [1][j-1]);
        }

        // Bottom boundary row (row my_rows)
        for(j = 1; j <= COLUMNS; j++) {
            Temperature[my_rows][j] = 0.25 * (Temperature[my_rows+1][j] +
                Temperature_last[my_rows-1][j] +
                                            Temperature_last[my_rows][j+1] +
                Temperature_last[my_rows][j-1]);
        }

        // PHASE 5: Calculate convergence with loop fusion and pointer swapping
        dt = 0.0;
        for(i = 1; i <= my_rows; i++){
            for(j = 1; j <= COLUMNS; j++){
                dt = fmax(fabs(Temperature[i][j] - Temperature_last[i][j]), dt);
            }
        }

        // Pointer swapping instead of array copying
        double (*temp_ptr)[COLUMNS+2] = Temperature_last;
        Temperature_last = Temperature;
        Temperature = temp_ptr;

        // find global dt using AllReduce (more efficient than Reduce+Bcast)
        MPI_Allreduce(&dt, &dt_global, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
```

```
        // periodically print test values - only for PE in lower corner
        if ((iteration % 100) == 0) {
            if (my_PE_num == npes-1){
                track_progress(iteration, my_rows);
            }
        }

        iteration++;
    }

    // Slightly more accurate timing and cleaner output
    MPI_Barrier(MPI_COMM_WORLD);

    // PE 0 finish timing and output values
    if (my_PE_num==0){
        gettimeofday(&stop_time,NULL);
        timersub(&stop_time, &start_time, &elapsed_time);

        printf("\nMax error at iteration %d was %f\n", iteration-1, dt_global);
        printf("Total time was %f seconds.\n", elapsed_time.tv_sec+elapsed_time.
            tv_usec/1000000.0);
        printf("Grid size: %dx%d, Processes: %d\n", ROWS_GLOBAL, COLUMNS, npes);
    }

    // Clean up dynamic memory
    free(Temperature);
    free(Temperature_last);

    MPI_Finalize();
    return 0;
}

void initialize(int npes, int my_PE_num, int my_rows){

    double tMin, tMax;  //Local boundary limits
    int i,j;

    // Initialize all cells to 0.0
    for(i = 0; i <= my_rows+1; i++){
        for (j = 0; j <= COLUMNS+1; j++){
            Temperature_last[i][j] = 0.0;
        }
    }

    // Calculate this PE's portion of the global boundary
    int rows_per_process = ROWS_GLOBAL / npes;
    int extra_rows = ROWS_GLOBAL % npes;
    int my_start_row;

    if (my_PE_num < extra_rows) {
        my_start_row = my_PE_num * (rows_per_process + 1);
    } else {
        my_start_row = extra_rows * (rows_per_process + 1) +
```

```
                        (my_PE_num - extra_rows) * rows_per_process;
    }

    // Local boundary condition endpoints
    tMin = my_start_row * 100.0 / ROWS_GLOBAL;
    tMax = (my_start_row + my_rows) * 100.0 / ROWS_GLOBAL;

    // Left and right boundaries
    for (i = 0; i <= my_rows+1; i++) {
        Temperature_last[i][0] = 0.0;
        Temperature_last[i][COLUMNS+1] = tMin + ((tMax-tMin)/my_rows)*i;
    }

    // Top boundary (PE 0 only)
    if (my_PE_num == 0)
        for (j = 0; j <= COLUMNS+1; j++)
            Temperature_last[0][j] = 0.0;

    // Bottom boundary (Last PE only)
    if (my_PE_num == npes-1)
        for (j=0; j<=COLUMNS+1; j++)
            Temperature_last[my_rows+1][j] = (100.0/COLUMNS) * j;
}

// only called by last PE
void track_progress(int iteration, int my_rows) {

    int i;

    printf("----------␣Iteration␣number:␣%d␣-----------\n", iteration);

    // output global coordinates so user doesn't␣have␣to␣understand␣decomposition
␣␣␣␣for(i␣=␣5;␣i␣>=␣0;␣i--)␣{
␣␣␣␣␣␣␣␣printf("[%d,%d]:␣%5.2f␣␣",␣ROWS_GLOBAL-i,␣COLUMNS-i,␣Temperature_last[
    my_rows-i][COLUMNS-i]);
␣␣␣␣}
␣␣␣␣printf("\n");
}
```

9.4 Code D: hw3_laplace_mpi_4.c

```
/*************************************************************
 * Project: CI Pathway Summer 2025
 * Course: Parallel Programing
 * Assignment: Distributed Memory Parallelism
 * Original Author: John Urbanic, PSC 2014
 * Author: Hochan Son, UCLA, Statistics
 * Date : 2025-06-30

 * Note:
  - Implemented robust circular (ring) communication: Each PE exchanges boundary
  rows with its neighbors using a deadlock-free pattern, supporting any number of
```

```
  processes.
  - Improved boundary initialization: The initialze_circular function sets left/
      right
  boundaries for all PEs, and top/bottom boundaries for the first/last PE,
      ensuring
  proper boundary conditions.
  - Added periodic progress reporting: The code prints progress every 100
      iterations
   from the last PE, showing representative grid values.
  - Added timing and result summary: The code measures and reports total runtime
  and final error on PE 0.
  - Memory management: Dynamic allocation and cleanup of 2D arrays for temperature
  grids.
  - Verbose control: Communication messages are printed only if the verbose flag
      is
  enabled.
 ****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <mpi.h>


#define COLUMNS      1000
#define ROWS_GLOBAL  1000        // this is a "global" row count

// communication tags
#define DOWN     100
#define UP       101

#define MAX_TEMP_ERROR 0.01
#define verbose 0

void initialze_circular(int ROWS, int npes, int my_PE_num, double **
    Temperature_last) {
    //All: generic boundary
    for (int i = 0; i <= ROWS + 1; i++) {
        Temperature_last[i][0] = 0.0;           // Left boundary
        Temperature_last[i][COLUMNS+1] = 100.0; // Right boundary
    }

    //PE_0: set top boundary
    if (my_PE_num == 0) {
        for (int j = 0; j <= COLUMNS + 1; j++)
            Temperature_last[0][j] = 0.0; // Top boundary
    }

    //PE_7: set bottom boundary
    if (my_PE_num == npes-1) {
        for (int j = 0; j <= COLUMNS + 1; j++)
            Temperature_last[ROWS+1][j] = 100.0; // Bottom boundary
```

```
    }
}

// only called by last PE
void track_progress(int iteration, int ROWS, double **Temperature) {

    printf("----------␣Iteration␣number:␣%d␣------------\n", iteration);
    // output global coordinates so user doesn't␣have␣to␣understand␣decomposition
␣␣␣␣for(int␣i␣=␣5;␣i␣>=␣0;␣i--)␣{
␣␣␣␣␣␣printf("[%d,%d]:␣%5.2f␣␣",␣ROWS-i,␣COLUMNS-i,␣Temperature[ROWS-i][COLUMNS-i
    ]);
␣␣␣␣}
␣␣␣␣printf("\n");
}

int␣main(int␣argc,␣char**␣argv)␣{
␣␣␣␣int␣my_PE_num;␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣Current␣PE
␣␣␣␣int␣npes;␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣number␣of␣PEs
␣␣␣␣int␣next_PE,␣prev_PE;
␣␣␣␣double␣dt,␣dt_global␣=␣100;
␣␣␣␣struct␣timeval␣start_time,␣stop_time,␣elapsed_time;
␣␣␣␣int␣max_iterations␣=␣4000;
␣␣␣␣int␣iteration␣=␣1;
␣␣␣␣MPI_Status␣status;␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣status␣returned␣by␣MPI␣calls

␣␣␣␣//␣Initialize␣MPI
␣␣␣␣MPI_Init(&argc,␣&argv);
␣␣␣␣MPI_Comm_rank(MPI_COMM_WORLD,␣&my_PE_num);
␣␣␣␣MPI_Comm_size(MPI_COMM_WORLD,␣&npes);


␣␣␣␣//␣Calculate␣ring␣neighbors
␣␣␣␣next_PE␣=␣(my_PE_num␣+␣1)␣%␣npes;
␣␣␣␣prev_PE␣=␣(my_PE_num␣-␣1␣+␣npes)␣%␣npes;

␣␣␣␣//␣PE␣0␣asks␣for␣input,␣default␣has␣been␣set␣to␣4000
␣␣␣␣//␣if(my_PE_num␣==␣0)␣{
␣␣␣␣//␣␣␣␣␣␣printf("Maximum␣iterations␣[100-4000]?\n");
␣␣␣␣//␣␣␣␣␣␣fflush(stdout);␣//␣Not␣always␣necessary,␣but␣can␣be␣helpful
␣␣␣␣//␣␣␣␣␣␣scanf("%d",␣&max_iterations);
␣␣␣␣//␣}
␣␣␣␣MPI_Bcast(&max_iterations,␣1,␣MPI_INT,␣0,␣MPI_COMM_WORLD);

␣␣␣␣if␣(my_PE_num==0)␣gettimeofday(&start_time,NULL);

␣␣␣␣int␣rows_per_process␣=␣ROWS_GLOBAL␣/␣npes;␣//␣Dynamically␣Calculate␣local␣
    dimensions
␣␣␣␣int␣ghost_rows␣=␣ROWS_GLOBAL␣%␣npes;
␣␣␣␣int␣ROWS␣=␣rows_per_process␣+␣(my_PE_num␣<␣ghost_rows␣?␣1␣:␣0);␣//␣for␣even␣
    distribution␣of␣the␣ghost_rows␣in␣case␣the␣rows␣are␣not␣exactly␣divisible␣by␣
    process␣X.

␣␣␣␣//␣Dynamically␣allocate␣memory␣space
```

```
    double **Temperature = (double **)malloc((ROWS + 2) * sizeof(double *));
    double **Temperature_last = (double **)malloc((ROWS + 2) * sizeof(double *));
    for (int i = 0; i < ROWS + 2; i++) {
        Temperature[i] = (double *)malloc((COLUMNS + 2) * sizeof(double));
        Temperature_last[i] = (double *)malloc((COLUMNS + 2) * sizeof(double));
    }
    initialze_circular(ROWS, npes, my_PE_num, Temperature_last);

    while (dt_global > MAX_TEMP_ERROR && iteration <= max_iterations) {
        // Main calculation: average four neighbors
        for (int i = 1; i <= ROWS; i++) {
            for (int j = 1; j <= COLUMNS; j++) {
                Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
                    Temperature_last[i-1][j] +
                                            Temperature_last[i][j+1] +
                    Temperature_last[i][j-1]);
            }
        }

        //only send/recv more than 1 processors
        if (npes > 1) {
            // COMMUNICATION PHASE: a circular ring communication (avoiding
                deadlock)
            // Each PE sends its bottom row to next_PE and receives top ghost row
                from prev_PE
            if (my_PE_num % 2 == 0) {
                if (verbose) printf("PE %d sending bottom row to PE %d\n",
                    my_PE_num, next_PE);
                MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, next_PE, DOWN
                    , MPI_COMM_WORLD);
                if (verbose) printf("PE %d receiving top ghost row from PE %d\n",
                    my_PE_num, prev_PE);
                MPI_Recv(&Temperature[0][1], COLUMNS, MPI_DOUBLE, prev_PE, DOWN,
                    MPI_COMM_WORLD, &status);
            } else {
                if (verbose) printf("PE %d receiving top ghost row from PE %d\n",
                    my_PE_num, prev_PE);
                MPI_Recv(&Temperature[0][1], COLUMNS, MPI_DOUBLE, prev_PE, DOWN,
                    MPI_COMM_WORLD, &status);
                if (verbose) printf("PE %d sending bottom row to PE %d\n",
                    my_PE_num, next_PE);
                MPI_Send(&Temperature[ROWS][1], COLUMNS, MPI_DOUBLE, next_PE, DOWN
                    , MPI_COMM_WORLD);
            }

            // Each PE sends its top row to prev_PE and receives bottom ghost row
                from next_PE
            if (my_PE_num % 2 == 0) {
                if (verbose) printf("PE %d sending top row to PE %d\n", my_PE_num,
                    prev_PE);
                MPI_Send(&Temperature[1][1], COLUMNS, MPI_DOUBLE, prev_PE, UP,
                    MPI_COMM_WORLD);
                if (verbose) printf("PE %d receiving bottom ghost row from PE %d\n
```

```
    ",␣my_PE_num,␣next_PE);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣MPI_Recv(&Temperature[ROWS+1][1],␣COLUMNS,␣MPI_DOUBLE,␣next_PE,␣UP
    ,␣MPI_COMM_WORLD,␣&status);
␣␣␣␣␣␣␣␣␣␣␣␣}␣else␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣(verbose)␣printf("PE␣%d␣receiving␣bottom␣ghost␣row␣from␣PE␣%d\n
    ",␣my_PE_num,␣next_PE);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣MPI_Recv(&Temperature[ROWS+1][1],␣COLUMNS,␣MPI_DOUBLE,␣next_PE,␣UP
    ,␣MPI_COMM_WORLD,␣&status);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣(verbose)␣printf("PE␣%d␣sending␣top␣row␣to␣PE␣%d\n",␣my_PE_num,
    ␣prev_PE);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣MPI_Send(&Temperature[1][1],␣COLUMNS,␣MPI_DOUBLE,␣prev_PE,␣UP,␣
    MPI_COMM_WORLD);
␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣}

␣␣␣␣␣␣␣␣//␣Compute␣local␣max␣difference␣and␣update␣Temperature_last
␣␣␣␣␣␣␣␣dt␣=␣0.0;
␣␣␣␣␣␣␣␣for␣(int␣i␣=␣1;␣i␣<=␣ROWS;␣i++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣(int␣j␣=␣1;␣j␣<=␣COLUMNS;␣j++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣dt␣=␣fmax(fabs(Temperature[i][j]␣-␣Temperature_last[i][j]),␣dt);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Temperature_last[i][j]␣=␣Temperature[i][j];
␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣}

␣␣␣␣␣␣␣␣//␣Find␣global␣dt
␣␣␣␣␣␣␣␣MPI_Reduce(&dt,␣&dt_global,␣1,␣MPI_DOUBLE,␣MPI_MAX,␣0,␣MPI_COMM_WORLD);
␣␣␣␣␣␣␣␣MPI_Bcast(&dt_global,␣1,␣MPI_DOUBLE,␣0,␣MPI_COMM_WORLD);

␣␣␣␣␣␣␣␣//␣periodically␣print␣test␣values␣-␣only␣for␣PE␣in␣lower␣corner
␣␣␣␣␣␣␣␣if((iteration␣%␣100)␣==␣0)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣(my_PE_num␣==␣npes-1){
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣track_progress(iteration,␣ROWS,␣Temperature);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣}

␣␣␣␣␣␣␣␣iteration++;
␣␣␣␣}
␣␣␣␣//␣PE␣0␣finish␣timing␣and␣output␣values
␣␣␣␣if␣(my_PE_num==0){
␣␣␣␣␣␣␣␣gettimeofday(&stop_time,NULL);
␣␣␣␣␣␣␣␣timersub(&stop_time,␣&start_time,␣&elapsed_time);
␣␣␣␣␣␣␣␣printf("\n=================================␣result␣
    ==================================");
␣␣␣␣␣␣␣␣printf("\nMax␣error␣at␣iteration␣%d␣was␣%f\n",␣iteration-1,␣dt_global);
␣␣␣␣␣␣␣␣printf("Total␣time␣was␣%f␣seconds.\n",␣elapsed_time.tv_sec+elapsed_time.
    tv_usec/1000000.0);
␣␣␣␣␣␣␣␣printf
    ("============================================================================\
    n");
␣␣␣␣}

␣␣␣␣//␣Free␣memory␣after␣all␣of␣the␣communication␣has␣finished
␣␣␣␣for␣(int␣i␣=␣0;␣i␣<␣ROWS␣+␣2;␣i++)␣{
```

```
        free(Temperature[i]);
        free(Temperature_last[i]);
    }
    free(Temperature);
    free(Temperature_last);

    MPI_Finalize();
    return 0;
}
```

10 Appendix.hw3_result.txt

10.1 result.txt

```
=== Basic System Info ===
Date: Sun Jul  6 16:18:53 CDT 2025
System: Linux cn093.delta.ncsa.illinois.edu 4.18.0-477.95.1.el8_8.x86_64 #1 SMP
    Fri Apr 11 09:50:48 EDT 2025 x86_64 x86_64 x86_64 GNU/Linux
CPU Info: AMD EPYC 7763 64-Core Processor
==========================
!!!!STARTING MPI PROCESS TEST - original !!!!
=== Test with  pe ===
Start time: 2025-07-06 16:18:53.106651311

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
Maximum iterations [100-4000]?
---------- Iteration number: 100 ------------
[995,995]: 63.33  [996,996]: 72.67  [997,997]: 81.40  [998,998]: 88.97  [999,999]:
     94.86  [1000,1000]: 98.67
---------- Iteration number: 200 ------------
[995,995]: 79.11  [996,996]: 84.86  [997,997]: 89.91  [998,998]: 94.10  [999,999]:
     97.26  [1000,1000]: 99.28
---------- Iteration number: 300 ------------
[995,995]: 85.25  [996,996]: 89.39  [997,997]: 92.96  [998,998]: 95.88  [999,999]:
     98.07  [1000,1000]: 99.49
---------- Iteration number: 400 ------------
[995,995]: 88.50  [996,996]: 91.75  [997,997]: 94.52  [998,998]: 96.78  [999,999]:
     98.48  [1000,1000]: 99.59
---------- Iteration number: 500 ------------
[995,995]: 90.52  [996,996]: 93.19  [997,997]: 95.47  [998,998]: 97.33  [999,999]:
     98.73  [1000,1000]: 99.66
---------- Iteration number: 600 ------------
[995,995]: 91.88  [996,996]: 94.17  [997,997]: 96.11  [998,998]: 97.69  [999,999]:
     98.89  [1000,1000]: 99.70
---------- Iteration number: 700 ------------
[995,995]: 92.87  [996,996]: 94.87  [997,997]: 96.57  [998,998]: 97.95  [999,999]:
     99.01  [1000,1000]: 99.73
---------- Iteration number: 800 ------------
```

```
[995,995]: 93.62  [996,996]: 95.40  [997,997]: 96.91  [998,998]: 98.15  [999,999]:
    99.10  [1000,1000]: 99.75
---------- Iteration number: 900 ------------
[995,995]: 94.21  [996,996]: 95.81  [997,997]: 97.18  [998,998]: 98.30  [999,999]:
    99.17  [1000,1000]: 99.77
---------- Iteration number: 1000 ------------
[995,995]: 94.68  [996,996]: 96.15  [997,997]: 97.40  [998,998]: 98.42  [999,999]:
    99.22  [1000,1000]: 99.78
---------- Iteration number: 1100 ------------
[995,995]: 95.06  [996,996]: 96.42  [997,997]: 97.57  [998,998]: 98.52  [999,999]:
    99.27  [1000,1000]: 99.79
---------- Iteration number: 1200 ------------
[995,995]: 95.39  [996,996]: 96.64  [997,997]: 97.72  [998,998]: 98.61  [999,999]:
    99.30  [1000,1000]: 99.80
---------- Iteration number: 1300 ------------
[995,995]: 95.66  [996,996]: 96.84  [997,997]: 97.84  [998,998]: 98.68  [999,999]:
    99.33  [1000,1000]: 99.81
---------- Iteration number: 1400 ------------
[995,995]: 95.90  [996,996]: 97.00  [997,997]: 97.95  [998,998]: 98.74  [999,999]:
    99.36  [1000,1000]: 99.82
---------- Iteration number: 1500 ------------
[995,995]: 96.10  [996,996]: 97.15  [997,997]: 98.04  [998,998]: 98.79  [999,999]:
    99.38  [1000,1000]: 99.82
---------- Iteration number: 1600 ------------
[995,995]: 96.28  [996,996]: 97.27  [997,997]: 98.12  [998,998]: 98.84  [999,999]:
    99.40  [1000,1000]: 99.83
---------- Iteration number: 1700 ------------
[995,995]: 96.44  [996,996]: 97.38  [997,997]: 98.20  [998,998]: 98.88  [999,999]:
    99.42  [1000,1000]: 99.83
---------- Iteration number: 1800 ------------
[995,995]: 96.58  [996,996]: 97.48  [997,997]: 98.26  [998,998]: 98.91  [999,999]:
    99.44  [1000,1000]: 99.83
---------- Iteration number: 1900 ------------
[995,995]: 96.70  [996,996]: 97.57  [997,997]: 98.32  [998,998]: 98.94  [999,999]:
    99.45  [1000,1000]: 99.84
---------- Iteration number: 2000 ------------
[995,995]: 96.81  [996,996]: 97.65  [997,997]: 98.37  [998,998]: 98.97  [999,999]:
    99.47  [1000,1000]: 99.84
---------- Iteration number: 2100 ------------
[995,995]: 96.92  [996,996]: 97.72  [997,997]: 98.41  [998,998]: 99.00  [999,999]:
    99.48  [1000,1000]: 99.84
---------- Iteration number: 2200 ------------
[995,995]: 97.01  [996,996]: 97.78  [997,997]: 98.45  [998,998]: 99.02  [999,999]:
    99.49  [1000,1000]: 99.85
---------- Iteration number: 2300 ------------
[995,995]: 97.09  [996,996]: 97.84  [997,997]: 98.49  [998,998]: 99.05  [999,999]:
    99.50  [1000,1000]: 99.85
---------- Iteration number: 2400 ------------
[995,995]: 97.17  [996,996]: 97.90  [997,997]: 98.53  [998,998]: 99.06  [999,999]:
    99.51  [1000,1000]: 99.85
---------- Iteration number: 2500 ------------
[995,995]: 97.24  [996,996]: 97.95  [997,997]: 98.56  [998,998]: 99.08  [999,999]:
    99.51  [1000,1000]: 99.85
```

```
---------- Iteration number: 2600 ------------
[995,995]: 97.31  [996,996]: 97.99  [997,997]: 98.59  [998,998]: 99.10  [999,999]:
     99.52  [1000,1000]: 99.86
---------- Iteration number: 2700 ------------
[995,995]: 97.37  [996,996]: 98.04  [997,997]: 98.62  [998,998]: 99.12  [999,999]:
     99.53  [1000,1000]: 99.86
---------- Iteration number: 2800 ------------
[995,995]: 97.43  [996,996]: 98.08  [997,997]: 98.64  [998,998]: 99.13  [999,999]:
     99.54  [1000,1000]: 99.86
---------- Iteration number: 2900 ------------
[995,995]: 97.48  [996,996]: 98.11  [997,997]: 98.67  [998,998]: 99.14  [999,999]:
     99.54  [1000,1000]: 99.86
---------- Iteration number: 3000 ------------
[995,995]: 97.53  [996,996]: 98.15  [997,997]: 98.69  [998,998]: 99.16  [999,999]:
     99.55  [1000,1000]: 99.86
---------- Iteration number: 3100 ------------
[995,995]: 97.58  [996,996]: 98.18  [997,997]: 98.71  [998,998]: 99.17  [999,999]:
     99.55  [1000,1000]: 99.86
---------- Iteration number: 3200 ------------
[995,995]: 97.62  [996,996]: 98.21  [997,997]: 98.73  [998,998]: 99.18  [999,999]:
     99.56  [1000,1000]: 99.86
---------- Iteration number: 3300 ------------
[995,995]: 97.66  [996,996]: 98.24  [997,997]: 98.75  [998,998]: 99.19  [999,999]:
     99.56  [1000,1000]: 99.87

Max error at iteration 3372 was 0.009995
Total time was 6.038548 seconds.
End time: 2025-07-06 16:18:59.451181586
Total wall clock time: 6.341 seconds
----------------------------------------

!!!!STARTING MPI PROCESS TEST - 1d optimized!!!!
=== Test with 1 pe ===
Start time: 2025-07-06 16:19:00.460036338

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
=== Complete Working 1D Linear Laplace MPI Solver ===
Running with 1 processes
Grid size: 1000 x 1000
Process 0 managing 1000 rows
Maximum iterations [100-4000]?
Process 0: initialized boundaries (tMin=0.0, tMax=100.0, rows=1000)
---------- Iteration number: 100 ------------
[995,995]: 63.33  [996,996]: 72.67  [997,997]: 81.40  [998,998]: 88.97  [999,999]:
     94.86  [1000,1000]: 98.67
Iteration 100: dt_global = 0.355752
---------- Iteration number: 200 ------------
[995,995]: 79.11  [996,996]: 84.86  [997,997]: 89.91  [998,998]: 94.10  [999,999]:
     97.26  [1000,1000]: 99.28
Iteration 200: dt_global = 0.177398
---------- Iteration number: 300 ------------
```

```
[995,995]: 85.25  [996,996]: 89.39  [997,997]: 92.96  [998,998]: 95.88  [999,999]:
    98.07  [1000,1000]: 99.49
Iteration 300: dt_global = 0.117823
---------- Iteration number: 400 ------------
[995,995]: 88.50  [996,996]: 91.75  [997,997]: 94.52  [998,998]: 96.78  [999,999]:
    98.48  [1000,1000]: 99.59
Iteration 400: dt_global = 0.088080
---------- Iteration number: 500 ------------
[995,995]: 90.52  [996,996]: 93.19  [997,997]: 95.47  [998,998]: 97.33  [999,999]:
    98.73  [1000,1000]: 99.66
Iteration 500: dt_global = 0.070262
---------- Iteration number: 600 ------------
[995,995]: 91.88  [996,996]: 94.17  [997,997]: 96.11  [998,998]: 97.69  [999,999]:
    98.89  [1000,1000]: 99.70
Iteration 600: dt_global = 0.058427
---------- Iteration number: 700 ------------
[995,995]: 92.87  [996,996]: 94.87  [997,997]: 96.57  [998,998]: 97.95  [999,999]:
    99.01  [1000,1000]: 99.73
Iteration 700: dt_global = 0.049974
---------- Iteration number: 800 ------------
[995,995]: 93.62  [996,996]: 95.40  [997,997]: 96.91  [998,998]: 98.15  [999,999]:
    99.10  [1000,1000]: 99.75
Iteration 800: dt_global = 0.043625
---------- Iteration number: 900 ------------
[995,995]: 94.21  [996,996]: 95.81  [997,997]: 97.18  [998,998]: 98.30  [999,999]:
    99.17  [1000,1000]: 99.77
Iteration 900: dt_global = 0.038710
---------- Iteration number: 1000 ------------
[995,995]: 94.68  [996,996]: 96.15  [997,997]: 97.40  [998,998]: 98.42  [999,999]:
    99.22  [1000,1000]: 99.78
Iteration 1000: dt_global = 0.034767
---------- Iteration number: 1100 ------------
[995,995]: 95.06  [996,996]: 96.42  [997,997]: 97.57  [998,998]: 98.52  [999,999]:
    99.27  [1000,1000]: 99.79
Iteration 1100: dt_global = 0.031554
---------- Iteration number: 1200 ------------
[995,995]: 95.39  [996,996]: 96.64  [997,997]: 97.72  [998,998]: 98.61  [999,999]:
    99.30  [1000,1000]: 99.80
Iteration 1200: dt_global = 0.028876
---------- Iteration number: 1300 ------------
[995,995]: 95.66  [996,996]: 96.84  [997,997]: 97.84  [998,998]: 98.68  [999,999]:
    99.33  [1000,1000]: 99.81
Iteration 1300: dt_global = 0.026607
---------- Iteration number: 1400 ------------
[995,995]: 95.90  [996,996]: 97.00  [997,997]: 97.95  [998,998]: 98.74  [999,999]:
    99.36  [1000,1000]: 99.82
Iteration 1400: dt_global = 0.024668
---------- Iteration number: 1500 ------------
[995,995]: 96.10  [996,996]: 97.15  [997,997]: 98.04  [998,998]: 98.79  [999,999]:
    99.38  [1000,1000]: 99.82
Iteration 1500: dt_global = 0.022988
---------- Iteration number: 1600 ------------
[995,995]: 96.28  [996,996]: 97.27  [997,997]: 98.12  [998,998]: 98.84  [999,999]:
```

```
    99.40  [1000,1000]: 99.83
Iteration 1600: dt_global = 0.021521
---------- Iteration number: 1700 ------------
[995,995]: 96.44  [996,996]: 97.38  [997,997]: 98.20  [998,998]: 98.88  [999,999]:
    99.42  [1000,1000]: 99.83
Iteration 1700: dt_global = 0.020227
---------- Iteration number: 1800 ------------
[995,995]: 96.58  [996,996]: 97.48  [997,997]: 98.26  [998,998]: 98.91  [999,999]:
    99.44  [1000,1000]: 99.83
Iteration 1800: dt_global = 0.019076
---------- Iteration number: 1900 ------------
[995,995]: 96.70  [996,996]: 97.57  [997,997]: 98.32  [998,998]: 98.94  [999,999]:
    99.45  [1000,1000]: 99.84
Iteration 1900: dt_global = 0.018048
---------- Iteration number: 2000 ------------
[995,995]: 96.81  [996,996]: 97.65  [997,997]: 98.37  [998,998]: 98.97  [999,999]:
    99.47  [1000,1000]: 99.84
Iteration 2000: dt_global = 0.017123
---------- Iteration number: 2100 ------------
[995,995]: 96.92  [996,996]: 97.72  [997,997]: 98.41  [998,998]: 99.00  [999,999]:
    99.48  [1000,1000]: 99.84
Iteration 2100: dt_global = 0.016286
---------- Iteration number: 2200 ------------
[995,995]: 97.01  [996,996]: 97.78  [997,997]: 98.45  [998,998]: 99.02  [999,999]:
    99.49  [1000,1000]: 99.85
Iteration 2200: dt_global = 0.015526
---------- Iteration number: 2300 ------------
[995,995]: 97.09  [996,996]: 97.84  [997,997]: 98.49  [998,998]: 99.05  [999,999]:
    99.50  [1000,1000]: 99.85
Iteration 2300: dt_global = 0.014832
---------- Iteration number: 2400 ------------
[995,995]: 97.17  [996,996]: 97.90  [997,997]: 98.53  [998,998]: 99.06  [999,999]:
    99.51  [1000,1000]: 99.85
Iteration 2400: dt_global = 0.014196
---------- Iteration number: 2500 ------------
[995,995]: 97.24  [996,996]: 97.95  [997,997]: 98.56  [998,998]: 99.08  [999,999]:
    99.51  [1000,1000]: 99.85
Iteration 2500: dt_global = 0.013612
---------- Iteration number: 2600 ------------
[995,995]: 97.31  [996,996]: 97.99  [997,997]: 98.59  [998,998]: 99.10  [999,999]:
    99.52  [1000,1000]: 99.86
Iteration 2600: dt_global = 0.013073
---------- Iteration number: 2700 ------------
[995,995]: 97.37  [996,996]: 98.04  [997,997]: 98.62  [998,998]: 99.12  [999,999]:
    99.53  [1000,1000]: 99.86
Iteration 2700: dt_global = 0.012574
---------- Iteration number: 2800 ------------
[995,995]: 97.43  [996,996]: 98.08  [997,997]: 98.64  [998,998]: 99.13  [999,999]:
    99.54  [1000,1000]: 99.86
Iteration 2800: dt_global = 0.012112
---------- Iteration number: 2900 ------------
[995,995]: 97.48  [996,996]: 98.11  [997,997]: 98.67  [998,998]: 99.14  [999,999]:
    99.54  [1000,1000]: 99.86
```

```
Iteration 2900: dt_global = 0.011682
---------- Iteration number: 3000 ------------
[995,995]: 97.53  [996,996]: 98.15  [997,997]: 98.69  [998,998]: 99.16  [999,999]:
    99.55  [1000,1000]: 99.86
Iteration 3000: dt_global = 0.011279
---------- Iteration number: 3100 ------------
[995,995]: 97.58  [996,996]: 98.18  [997,997]: 98.71  [998,998]: 99.17  [999,999]:
    99.55  [1000,1000]: 99.86
Iteration 3100: dt_global = 0.010904
---------- Iteration number: 3200 ------------
[995,995]: 97.62  [996,996]: 98.21  [997,997]: 98.73  [998,998]: 99.18  [999,999]:
    99.56  [1000,1000]: 99.86
Iteration 3200: dt_global = 0.010551
---------- Iteration number: 3300 ------------
[995,995]: 97.66  [996,996]: 98.24  [997,997]: 98.75  [998,998]: 99.19  [999,999]:
    99.56  [1000,1000]: 99.87
Iteration 3300: dt_global = 0.010222

============== RESULTS ==============
CONVERGED after 3372 iterations
Final error: 0.009995
Total time: 21.854539 seconds
====================================
End time: 2025-07-06 16:19:22.590312517
Total wall clock time: 22.127 seconds
----------------------------------------

=== Test with 4 pe ===
Start time: 2025-07-06 16:19:23.598638230

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
=== Complete Working 1D Linear Laplace MPI Solver ===
Running with 4 processes
Grid size: 1000 x 1000
Process 0 managing 250 rows
Maximum iterations [100-4000]?
Process 0: initialized boundaries (tMin=0.0, tMax=25.0, rows=250)
Process 1: initialized boundaries (tMin=25.0, tMax=50.0, rows=250)
Process 2: initialized boundaries (tMin=50.0, tMax=75.0, rows=250)
Process 3: initialized boundaries (tMin=75.0, tMax=100.0, rows=250)
---------- Iteration number: 100 ------------
[995,995]: 63.33  [996,996]: 72.67  [997,997]: 81.40  [998,998]: 88.97  [999,999]:
    94.86  [1000,1000]: 98.67
Iteration 100: dt_global = 0.355752
Iteration 200: dt_global = 0.177398
---------- Iteration number: 200 ------------
[995,995]: 79.11  [996,996]: 84.86  [997,997]: 89.91  [998,998]: 94.10  [999,999]:
    97.26  [1000,1000]: 99.28
---------- Iteration number: 300 ------------
[995,995]: 85.25  [996,996]: 89.39  [997,997]: 92.96  [998,998]: 95.88  [999,999]:
    98.07  [1000,1000]: 99.49
```

```
Iteration 300: dt_global = 0.117823
Iteration 400: dt_global = 0.088080
---------- Iteration number: 400 ------------
[995,995]: 88.50  [996,996]: 91.75  [997,997]: 94.52  [998,998]: 96.78  [999,999]:
    98.48  [1000,1000]: 99.59
---------- Iteration number: 500 ------------
[995,995]: 90.52  [996,996]: 93.19  [997,997]: 95.47  [998,998]: 97.33  [999,999]:
    98.73  [1000,1000]: 99.66
Iteration 500: dt_global = 0.070262
Iteration 600: dt_global = 0.058427
---------- Iteration number: 600 ------------
[995,995]: 91.88  [996,996]: 94.17  [997,997]: 96.11  [998,998]: 97.69  [999,999]:
    98.89  [1000,1000]: 99.70
Iteration 700: dt_global = 0.049974
---------- Iteration number: 700 ------------
[995,995]: 92.87  [996,996]: 94.87  [997,997]: 96.57  [998,998]: 97.95  [999,999]:
    99.01  [1000,1000]: 99.73
Iteration 800: dt_global = 0.043625
---------- Iteration number: 800 ------------
[995,995]: 93.62  [996,996]: 95.40  [997,997]: 96.91  [998,998]: 98.15  [999,999]:
    99.10  [1000,1000]: 99.75
Iteration 900: dt_global = 0.038710
---------- Iteration number: 900 ------------
[995,995]: 94.21  [996,996]: 95.81  [997,997]: 97.18  [998,998]: 98.30  [999,999]:
    99.17  [1000,1000]: 99.77
Iteration 1000: dt_global = 0.034767
---------- Iteration number: 1000 ------------
[995,995]: 94.68  [996,996]: 96.15  [997,997]: 97.40  [998,998]: 98.42  [999,999]:
    99.22  [1000,1000]: 99.78
---------- Iteration number: 1100 ------------
[995,995]: 95.06  [996,996]: 96.42  [997,997]: 97.57  [998,998]: 98.52  [999,999]:
    99.27  [1000,1000]: 99.79
Iteration 1100: dt_global = 0.031554
---------- Iteration number: 1200 ------------
[995,995]: 95.39  [996,996]: 96.64  [997,997]: 97.72  [998,998]: 98.61  [999,999]:
    99.30  [1000,1000]: 99.80
Iteration 1200: dt_global = 0.028876
Iteration 1300: dt_global = 0.026607
---------- Iteration number: 1300 ------------
[995,995]: 95.66  [996,996]: 96.84  [997,997]: 97.84  [998,998]: 98.68  [999,999]:
    99.33  [1000,1000]: 99.81
Iteration 1400: dt_global = 0.024668
---------- Iteration number: 1400 ------------
[995,995]: 95.90  [996,996]: 97.00  [997,997]: 97.95  [998,998]: 98.74  [999,999]:
    99.36  [1000,1000]: 99.82
---------- Iteration number: 1500 ------------
[995,995]: 96.10  [996,996]: 97.15  [997,997]: 98.04  [998,998]: 98.79  [999,999]:
    99.38  [1000,1000]: 99.82
Iteration 1500: dt_global = 0.022988
Iteration 1600: dt_global = 0.021521
---------- Iteration number: 1600 ------------
[995,995]: 96.28  [996,996]: 97.27  [997,997]: 98.12  [998,998]: 98.84  [999,999]:
    99.40  [1000,1000]: 99.83
```

```
Iteration 1700: dt_global = 0.020227
---------- Iteration number: 1700 ------------
[995,995]: 96.44  [996,996]: 97.38  [997,997]: 98.20  [998,998]: 98.88  [999,999]:
    99.42  [1000,1000]: 99.83
Iteration 1800: dt_global = 0.019076
---------- Iteration number: 1800 ------------
[995,995]: 96.58  [996,996]: 97.48  [997,997]: 98.26  [998,998]: 98.91  [999,999]:
    99.44  [1000,1000]: 99.83
Iteration 1900: dt_global = 0.018048
---------- Iteration number: 1900 ------------
[995,995]: 96.70  [996,996]: 97.57  [997,997]: 98.32  [998,998]: 98.94  [999,999]:
    99.45  [1000,1000]: 99.84
---------- Iteration number: 2000 ------------
[995,995]: 96.81  [996,996]: 97.65  [997,997]: 98.37  [998,998]: 98.97  [999,999]:
    99.47  [1000,1000]: 99.84
Iteration 2000: dt_global = 0.017123
---------- Iteration number: 2100 ------------
[995,995]: 96.92  [996,996]: 97.72  [997,997]: 98.41  [998,998]: 99.00  [999,999]:
    99.48  [1000,1000]: 99.84
Iteration 2100: dt_global = 0.016286
---------- Iteration number: 2200 ------------
[995,995]: 97.01  [996,996]: 97.78  [997,997]: 98.45  [998,998]: 99.02  [999,999]:
    99.49  [1000,1000]: 99.85
Iteration 2200: dt_global = 0.015526
---------- Iteration number: 2300 ------------
[995,995]: 97.09  [996,996]: 97.84  [997,997]: 98.49  [998,998]: 99.05  [999,999]:
    99.50  [1000,1000]: 99.85
Iteration 2300: dt_global = 0.014832
---------- Iteration number: 2400 ------------
[995,995]: 97.17  [996,996]: 97.90  [997,997]: 98.53  [998,998]: 99.06  [999,999]:
    99.51  [1000,1000]: 99.85
Iteration 2400: dt_global = 0.014196
---------- Iteration number: 2500 ------------
[995,995]: 97.24  [996,996]: 97.95  [997,997]: 98.56  [998,998]: 99.08  [999,999]:
    99.51  [1000,1000]: 99.85
Iteration 2500: dt_global = 0.013612
---------- Iteration number: 2600 ------------
[995,995]: 97.31  [996,996]: 97.99  [997,997]: 98.59  [998,998]: 99.10  [999,999]:
    99.52  [1000,1000]: 99.86
Iteration 2600: dt_global = 0.013073
---------- Iteration number: 2700 ------------
[995,995]: 97.37  [996,996]: 98.04  [997,997]: 98.62  [998,998]: 99.12  [999,999]:
    99.53  [1000,1000]: 99.86
Iteration 2700: dt_global = 0.012574
Iteration 2800: dt_global = 0.012112
---------- Iteration number: 2800 ------------
[995,995]: 97.43  [996,996]: 98.08  [997,997]: 98.64  [998,998]: 99.13  [999,999]:
    99.54  [1000,1000]: 99.86
---------- Iteration number: 2900 ------------
[995,995]: 97.48  [996,996]: 98.11  [997,997]: 98.67  [998,998]: 99.14  [999,999]:
    99.54  [1000,1000]: 99.86
Iteration 2900: dt_global = 0.011682
---------- Iteration number: 3000 ------------
```

```
[995,995]: 97.53  [996,996]: 98.15  [997,997]: 98.69  [998,998]: 99.16  [999,999]:
    99.55  [1000,1000]: 99.86
Iteration 3000: dt_global = 0.011279
---------- Iteration number: 3100 ------------
[995,995]: 97.58  [996,996]: 98.18  [997,997]: 98.71  [998,998]: 99.17  [999,999]:
    99.55  [1000,1000]: 99.86
Iteration 3100: dt_global = 0.010904
Iteration 3200: dt_global = 0.010551
---------- Iteration number: 3200 ------------
[995,995]: 97.62  [996,996]: 98.21  [997,997]: 98.73  [998,998]: 99.18  [999,999]:
    99.56  [1000,1000]: 99.86
Iteration 3300: dt_global = 0.010222
---------- Iteration number: 3300 ------------
[995,995]: 97.66  [996,996]: 98.24  [997,997]: 98.75  [998,998]: 99.19  [999,999]:
    99.56  [1000,1000]: 99.87

============== RESULTS ==============
CONVERGED after 3372 iterations
Final error: 0.009995
Total time: 6.079321 seconds
====================================
End time: 2025-07-06 16:19:30.000456318
Total wall clock time: 6.399 seconds
----------------------------------------

=== Test with 8 pe ===
Start time: 2025-07-06 16:19:31.007778315

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
=== Complete Working 1D Linear Laplace MPI Solver ===
Running with 8 processes
Grid size: 1000 x 1000
Process 0 managing 125 rows
Maximum iterations [100-4000]?
Process 1: initialized boundaries (tMin=12.5, tMax=25.0, rows=125)
Process 2: initialized boundaries (tMin=25.0, tMax=37.5, rows=125)
Process 3: initialized boundaries (tMin=37.5, tMax=50.0, rows=125)
Process 4: initialized boundaries (tMin=50.0, tMax=62.5, rows=125)
Process 5: initialized boundaries (tMin=62.5, tMax=75.0, rows=125)
Process 6: initialized boundaries (tMin=75.0, tMax=87.5, rows=125)
Process 7: initialized boundaries (tMin=87.5, tMax=100.0, rows=125)
Process 0: initialized boundaries (tMin=0.0, tMax=12.5, rows=125)
---------- Iteration number: 100 ------------
[995,995]: 63.33  [996,996]: 72.67  [997,997]: 81.40  [998,998]: 88.97  [999,999]:
    94.86  [1000,1000]: 98.67
Iteration 100: dt_global = 0.355752
Iteration 200: dt_global = 0.177398
---------- Iteration number: 200 ------------
[995,995]: 79.11  [996,996]: 84.86  [997,997]: 89.91  [998,998]: 94.10  [999,999]:
    97.26  [1000,1000]: 99.28
---------- Iteration number: 300 ------------
```

```
[995,995]: 85.25  [996,996]: 89.39  [997,997]: 92.96  [998,998]: 95.88  [999,999]:
    98.07  [1000,1000]: 99.49
Iteration 300: dt_global = 0.117823
---------- Iteration number: 400 ------------
[995,995]: 88.50  [996,996]: 91.75  [997,997]: 94.52  [998,998]: 96.78  [999,999]:
    98.48  [1000,1000]: 99.59
Iteration 400: dt_global = 0.088080
---------- Iteration number: 500 ------------
[995,995]: 90.52  [996,996]: 93.19  [997,997]: 95.47  [998,998]: 97.33  [999,999]:
    98.73  [1000,1000]: 99.66
Iteration 500: dt_global = 0.070262
Iteration 600: dt_global = 0.058427
---------- Iteration number: 600 ------------
[995,995]: 91.88  [996,996]: 94.17  [997,997]: 96.11  [998,998]: 97.69  [999,999]:
    98.89  [1000,1000]: 99.70
---------- Iteration number: 700 ------------
[995,995]: 92.87  [996,996]: 94.87  [997,997]: 96.57  [998,998]: 97.95  [999,999]:
    99.01  [1000,1000]: 99.73
Iteration 700: dt_global = 0.049974
Iteration 800: dt_global = 0.043625
---------- Iteration number: 800 ------------
[995,995]: 93.62  [996,996]: 95.40  [997,997]: 96.91  [998,998]: 98.15  [999,999]:
    99.10  [1000,1000]: 99.75
---------- Iteration number: 900 ------------
[995,995]: 94.21  [996,996]: 95.81  [997,997]: 97.18  [998,998]: 98.30  [999,999]:
    99.17  [1000,1000]: 99.77
Iteration 900: dt_global = 0.038710
Iteration 1000: dt_global = 0.034767
---------- Iteration number: 1000 ------------
[995,995]: 94.68  [996,996]: 96.15  [997,997]: 97.40  [998,998]: 98.42  [999,999]:
    99.22  [1000,1000]: 99.78
---------- Iteration number: 1100 ------------
[995,995]: 95.06  [996,996]: 96.42  [997,997]: 97.57  [998,998]: 98.52  [999,999]:
    99.27  [1000,1000]: 99.79
Iteration 1100: dt_global = 0.031554
Iteration 1200: dt_global = 0.028876
---------- Iteration number: 1200 ------------
[995,995]: 95.39  [996,996]: 96.64  [997,997]: 97.72  [998,998]: 98.61  [999,999]:
    99.30  [1000,1000]: 99.80
---------- Iteration number: 1300 ------------
[995,995]: 95.66  [996,996]: 96.84  [997,997]: 97.84  [998,998]: 98.68  [999,999]:
    99.33  [1000,1000]: 99.81
Iteration 1300: dt_global = 0.026607
---------- Iteration number: 1400 ------------
[995,995]: 95.90  [996,996]: 97.00  [997,997]: 97.95  [998,998]: 98.74  [999,999]:
    99.36  [1000,1000]: 99.82
Iteration 1400: dt_global = 0.024668
Iteration 1500: dt_global = 0.022988
---------- Iteration number: 1500 ------------
[995,995]: 96.10  [996,996]: 97.15  [997,997]: 98.04  [998,998]: 98.79  [999,999]:
    99.38  [1000,1000]: 99.82
Iteration 1600: dt_global = 0.021521
---------- Iteration number: 1600 ------------
```

```
[995,995]: 96.28  [996,996]: 97.27  [997,997]: 98.12  [998,998]: 98.84  [999,999]:
    99.40  [1000,1000]: 99.83
---------- Iteration number: 1700 ------------
[995,995]: 96.44  [996,996]: 97.38  [997,997]: 98.20  [998,998]: 98.88  [999,999]:
    99.42  [1000,1000]: 99.83
Iteration 1700: dt_global = 0.020227
---------- Iteration number: 1800 ------------
[995,995]: 96.58  [996,996]: 97.48  [997,997]: 98.26  [998,998]: 98.91  [999,999]:
    99.44  [1000,1000]: 99.83
Iteration 1800: dt_global = 0.019076
---------- Iteration number: 1900 ------------
[995,995]: 96.70  [996,996]: 97.57  [997,997]: 98.32  [998,998]: 98.94  [999,999]:
    99.45  [1000,1000]: 99.84
Iteration 1900: dt_global = 0.018048
Iteration 2000: dt_global = 0.017123
---------- Iteration number: 2000 ------------
[995,995]: 96.81  [996,996]: 97.65  [997,997]: 98.37  [998,998]: 98.97  [999,999]:
    99.47  [1000,1000]: 99.84
---------- Iteration number: 2100 ------------
[995,995]: 96.92  [996,996]: 97.72  [997,997]: 98.41  [998,998]: 99.00  [999,999]:
    99.48  [1000,1000]: 99.84
Iteration 2100: dt_global = 0.016286
Iteration 2200: dt_global = 0.015526
---------- Iteration number: 2200 ------------
[995,995]: 97.01  [996,996]: 97.78  [997,997]: 98.45  [998,998]: 99.02  [999,999]:
    99.49  [1000,1000]: 99.85
---------- Iteration number: 2300 ------------
[995,995]: 97.09  [996,996]: 97.84  [997,997]: 98.49  [998,998]: 99.05  [999,999]:
    99.50  [1000,1000]: 99.85
Iteration 2300: dt_global = 0.014832
Iteration 2400: dt_global = 0.014196
---------- Iteration number: 2400 ------------
[995,995]: 97.17  [996,996]: 97.90  [997,997]: 98.53  [998,998]: 99.06  [999,999]:
    99.51  [1000,1000]: 99.85
---------- Iteration number: 2500 ------------
[995,995]: 97.24  [996,996]: 97.95  [997,997]: 98.56  [998,998]: 99.08  [999,999]:
    99.51  [1000,1000]: 99.85
Iteration 2500: dt_global = 0.013612
---------- Iteration number: 2600 ------------
[995,995]: 97.31  [996,996]: 97.99  [997,997]: 98.59  [998,998]: 99.10  [999,999]:
    99.52  [1000,1000]: 99.86
Iteration 2600: dt_global = 0.013073
Iteration 2700: dt_global = 0.012574
---------- Iteration number: 2700 ------------
[995,995]: 97.37  [996,996]: 98.04  [997,997]: 98.62  [998,998]: 99.12  [999,999]:
    99.53  [1000,1000]: 99.86
---------- Iteration number: 2800 ------------
[995,995]: 97.43  [996,996]: 98.08  [997,997]: 98.64  [998,998]: 99.13  [999,999]:
    99.54  [1000,1000]: 99.86
Iteration 2800: dt_global = 0.012112
---------- Iteration number: 2900 ------------
[995,995]: 97.48  [996,996]: 98.11  [997,997]: 98.67  [998,998]: 99.14  [999,999]:
    99.54  [1000,1000]: 99.86
```

```
Iteration 2900: dt_global = 0.011682
Iteration 3000: dt_global = 0.011279
---------- Iteration number: 3000 ------------
[995,995]: 97.53  [996,996]: 98.15  [997,997]: 98.69  [998,998]: 99.16  [999,999]:
    99.55  [1000,1000]: 99.86
Iteration 3100: dt_global = 0.010904
---------- Iteration number: 3100 ------------
[995,995]: 97.58  [996,996]: 98.18  [997,997]: 98.71  [998,998]: 99.17  [999,999]:
    99.55  [1000,1000]: 99.86
---------- Iteration number: 3200 ------------
[995,995]: 97.62  [996,996]: 98.21  [997,997]: 98.73  [998,998]: 99.18  [999,999]:
    99.56  [1000,1000]: 99.86
Iteration 3200: dt_global = 0.010551
---------- Iteration number: 3300 ------------
[995,995]: 97.66  [996,996]: 98.24  [997,997]: 98.75  [998,998]: 99.19  [999,999]:
    99.56  [1000,1000]: 99.87
Iteration 3300: dt_global = 0.010222

============== RESULTS ==============
CONVERGED after 3372 iterations
Final error: 0.009995
Total time: 3.666854 seconds
====================================
End time: 2025-07-06 16:19:35.061872580
Total wall clock time: 4.051 seconds
----------------------------------------

!!!!STARTING MPI PROCESS TEST - 2d optimized!!!!
=== Test with 1 pe ===
Start time: 2025-07-06 16:19:36.072479751

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
Maximum iterations [100-4000]?
Running on 1 processes with dynamic load balancing
---------- Iteration number: 100 ------------
[995,995]: 31.66  [996,996]: 36.34  [997,997]: 40.70  [998,998]: 44.48  [999,999]:
    47.43  [1000,1000]: 49.33
---------- Iteration number: 200 ------------
[995,995]: 39.55  [996,996]: 42.43  [997,997]: 44.96  [998,998]: 47.05  [999,999]:
    48.63  [1000,1000]: 49.64
---------- Iteration number: 300 ------------
[995,995]: 42.62  [996,996]: 44.69  [997,997]: 46.48  [998,998]: 47.94  [999,999]:
    49.04  [1000,1000]: 49.75
---------- Iteration number: 400 ------------
[995,995]: 44.25  [996,996]: 45.87  [997,997]: 47.26  [998,998]: 48.39  [999,999]:
    49.24  [1000,1000]: 49.80
---------- Iteration number: 500 ------------
[995,995]: 45.26  [996,996]: 46.60  [997,997]: 47.74  [998,998]: 48.66  [999,999]:
    49.36  [1000,1000]: 49.83
---------- Iteration number: 600 ------------
[995,995]: 45.94  [996,996]: 47.08  [997,997]: 48.06  [998,998]: 48.85  [999,999]:
```

```
     49.45  [1000,1000]: 49.85
---------- Iteration number: 700 ------------
[995,995]: 46.44  [996,996]: 47.44  [997,997]: 48.28  [998,998]: 48.98  [999,999]:
     49.50  [1000,1000]: 49.86
---------- Iteration number: 800 ------------
[995,995]: 46.81  [996,996]: 47.70  [997,997]: 48.46  [998,998]: 49.07  [999,999]:
     49.55  [1000,1000]: 49.87
---------- Iteration number: 900 ------------
[995,995]: 47.10  [996,996]: 47.91  [997,997]: 48.59  [998,998]: 49.15  [999,999]:
     49.58  [1000,1000]: 49.88
---------- Iteration number: 1000 ------------
[995,995]: 47.34  [996,996]: 48.07  [997,997]: 48.70  [998,998]: 49.21  [999,999]:
     49.61  [1000,1000]: 49.89
---------- Iteration number: 1100 ------------
[995,995]: 47.53  [996,996]: 48.21  [997,997]: 48.79  [998,998]: 49.26  [999,999]:
     49.63  [1000,1000]: 49.90
---------- Iteration number: 1200 ------------
[995,995]: 47.69  [996,996]: 48.32  [997,997]: 48.86  [998,998]: 49.30  [999,999]:
     49.65  [1000,1000]: 49.90
---------- Iteration number: 1300 ------------
[995,995]: 47.83  [996,996]: 48.42  [997,997]: 48.92  [998,998]: 49.34  [999,999]:
     49.67  [1000,1000]: 49.90
---------- Iteration number: 1400 ------------
[995,995]: 47.95  [996,996]: 48.50  [997,997]: 48.98  [998,998]: 49.37  [999,999]:
     49.68  [1000,1000]: 49.91
---------- Iteration number: 1500 ------------
[995,995]: 48.05  [996,996]: 48.57  [997,997]: 49.02  [998,998]: 49.40  [999,999]:
     49.69  [1000,1000]: 49.91
---------- Iteration number: 1600 ------------
[995,995]: 48.14  [996,996]: 48.64  [997,997]: 49.06  [998,998]: 49.42  [999,999]:
     49.70  [1000,1000]: 49.91
---------- Iteration number: 1700 ------------
[995,995]: 48.22  [996,996]: 48.69  [997,997]: 49.10  [998,998]: 49.44  [999,999]:
     49.71  [1000,1000]: 49.92
---------- Iteration number: 1800 ------------
[995,995]: 48.29  [996,996]: 48.74  [997,997]: 49.13  [998,998]: 49.46  [999,999]:
     49.72  [1000,1000]: 49.92
---------- Iteration number: 1900 ------------
[995,995]: 48.35  [996,996]: 48.78  [997,997]: 49.16  [998,998]: 49.47  [999,999]:
     49.73  [1000,1000]: 49.92
---------- Iteration number: 2000 ------------
[995,995]: 48.41  [996,996]: 48.82  [997,997]: 49.18  [998,998]: 49.49  [999,999]:
     49.73  [1000,1000]: 49.92
---------- Iteration number: 2100 ------------
[995,995]: 48.46  [996,996]: 48.86  [997,997]: 49.21  [998,998]: 49.50  [999,999]:
     49.74  [1000,1000]: 49.92
---------- Iteration number: 2200 ------------
[995,995]: 48.51  [996,996]: 48.89  [997,997]: 49.23  [998,998]: 49.51  [999,999]:
     49.74  [1000,1000]: 49.92
---------- Iteration number: 2300 ------------
[995,995]: 48.55  [996,996]: 48.92  [997,997]: 49.25  [998,998]: 49.52  [999,999]:
     49.75  [1000,1000]: 49.92
---------- Iteration number: 2400 ------------
```

```
[995,995]: 48.59  [996,996]: 48.95  [997,997]: 49.26  [998,998]: 49.53  [999,999]:
     49.75  [1000,1000]: 49.93
---------- Iteration number: 2500 ------------
[995,995]: 48.62  [996,996]: 48.97  [997,997]: 49.28  [998,998]: 49.54  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2600 ------------
[995,995]: 48.66  [996,996]: 49.00  [997,997]: 49.29  [998,998]: 49.55  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2700 ------------
[995,995]: 48.69  [996,996]: 49.02  [997,997]: 49.31  [998,998]: 49.56  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2800 ------------
[995,995]: 48.71  [996,996]: 49.04  [997,997]: 49.32  [998,998]: 49.56  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 2900 ------------
[995,995]: 48.74  [996,996]: 49.06  [997,997]: 49.33  [998,998]: 49.57  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 3000 ------------
[995,995]: 48.77  [996,996]: 49.07  [997,997]: 49.34  [998,998]: 49.58  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 3100 ------------
[995,995]: 48.79  [996,996]: 49.09  [997,997]: 49.35  [998,998]: 49.58  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3200 ------------
[995,995]: 48.81  [996,996]: 49.10  [997,997]: 49.36  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3300 ------------
[995,995]: 48.83  [996,996]: 49.12  [997,997]: 49.37  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3400 ------------
[995,995]: 48.85  [996,996]: 49.13  [997,997]: 49.38  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3500 ------------
[995,995]: 48.87  [996,996]: 49.14  [997,997]: 49.39  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3600 ------------
[995,995]: 48.88  [996,996]: 49.16  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3700 ------------
[995,995]: 48.90  [996,996]: 49.17  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3800 ------------
[995,995]: 48.92  [996,996]: 49.18  [997,997]: 49.41  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
---------- Iteration number: 3900 ------------
[995,995]: 48.93  [996,996]: 49.19  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
---------- Iteration number: 4000 ------------
[995,995]: 48.94  [996,996]: 49.20  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94

Max error at iteration 4000 was 20.922598
Total time was 26.027700 seconds.
```

```
Grid size: 1000x1000, Processes: 1
End time: 2025-07-06 16:20:02.390110775
Total wall clock time: 26.310 seconds
----------------------------------------

=== Test with 4 pe ===
Start time: 2025-07-06 16:20:03.397525608

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
Maximum iterations [100-4000]?
Running on 4 processes with dynamic load balancing
---------- Iteration number: 100 ------------
[995,995]: 31.66  [996,996]: 36.34  [997,997]: 40.70  [998,998]: 44.48  [999,999]:
     47.43  [1000,1000]: 49.33
---------- Iteration number: 200 ------------
[995,995]: 39.55  [996,996]: 42.43  [997,997]: 44.96  [998,998]: 47.05  [999,999]:
     48.63  [1000,1000]: 49.64
---------- Iteration number: 300 ------------
[995,995]: 42.62  [996,996]: 44.69  [997,997]: 46.48  [998,998]: 47.94  [999,999]:
     49.04  [1000,1000]: 49.75
---------- Iteration number: 400 ------------
[995,995]: 44.25  [996,996]: 45.87  [997,997]: 47.26  [998,998]: 48.39  [999,999]:
     49.24  [1000,1000]: 49.80
---------- Iteration number: 500 ------------
[995,995]: 45.26  [996,996]: 46.60  [997,997]: 47.74  [998,998]: 48.66  [999,999]:
     49.36  [1000,1000]: 49.83
---------- Iteration number: 600 ------------
[995,995]: 45.94  [996,996]: 47.08  [997,997]: 48.06  [998,998]: 48.85  [999,999]:
     49.45  [1000,1000]: 49.85
---------- Iteration number: 700 ------------
[995,995]: 46.44  [996,996]: 47.44  [997,997]: 48.28  [998,998]: 48.98  [999,999]:
     49.50  [1000,1000]: 49.86
---------- Iteration number: 800 ------------
[995,995]: 46.81  [996,996]: 47.70  [997,997]: 48.46  [998,998]: 49.07  [999,999]:
     49.55  [1000,1000]: 49.87
---------- Iteration number: 900 ------------
[995,995]: 47.10  [996,996]: 47.91  [997,997]: 48.59  [998,998]: 49.15  [999,999]:
     49.58  [1000,1000]: 49.88
---------- Iteration number: 1000 ------------
[995,995]: 47.34  [996,996]: 48.07  [997,997]: 48.70  [998,998]: 49.21  [999,999]:
     49.61  [1000,1000]: 49.89
---------- Iteration number: 1100 ------------
[995,995]: 47.53  [996,996]: 48.21  [997,997]: 48.79  [998,998]: 49.26  [999,999]:
     49.63  [1000,1000]: 49.90
---------- Iteration number: 1200 ------------
[995,995]: 47.69  [996,996]: 48.32  [997,997]: 48.86  [998,998]: 49.30  [999,999]:
     49.65  [1000,1000]: 49.90
---------- Iteration number: 1300 ------------
[995,995]: 47.83  [996,996]: 48.42  [997,997]: 48.92  [998,998]: 49.34  [999,999]:
     49.67  [1000,1000]: 49.90
---------- Iteration number: 1400 ------------
```

```
[995,995]: 47.95  [996,996]: 48.50  [997,997]: 48.98  [998,998]: 49.37  [999,999]:
    49.68  [1000,1000]: 49.91
---------- Iteration number: 1500 ------------
[995,995]: 48.05  [996,996]: 48.57  [997,997]: 49.02  [998,998]: 49.40  [999,999]:
    49.69  [1000,1000]: 49.91
---------- Iteration number: 1600 ------------
[995,995]: 48.14  [996,996]: 48.64  [997,997]: 49.06  [998,998]: 49.42  [999,999]:
    49.70  [1000,1000]: 49.91
---------- Iteration number: 1700 ------------
[995,995]: 48.22  [996,996]: 48.69  [997,997]: 49.10  [998,998]: 49.44  [999,999]:
    49.71  [1000,1000]: 49.92
---------- Iteration number: 1800 ------------
[995,995]: 48.29  [996,996]: 48.74  [997,997]: 49.13  [998,998]: 49.46  [999,999]:
    49.72  [1000,1000]: 49.92
---------- Iteration number: 1900 ------------
[995,995]: 48.35  [996,996]: 48.78  [997,997]: 49.16  [998,998]: 49.47  [999,999]:
    49.73  [1000,1000]: 49.92
---------- Iteration number: 2000 ------------
[995,995]: 48.41  [996,996]: 48.82  [997,997]: 49.18  [998,998]: 49.49  [999,999]:
    49.73  [1000,1000]: 49.92
---------- Iteration number: 2100 ------------
[995,995]: 48.46  [996,996]: 48.86  [997,997]: 49.21  [998,998]: 49.50  [999,999]:
    49.74  [1000,1000]: 49.92
---------- Iteration number: 2200 ------------
[995,995]: 48.51  [996,996]: 48.89  [997,997]: 49.23  [998,998]: 49.51  [999,999]:
    49.74  [1000,1000]: 49.92
---------- Iteration number: 2300 ------------
[995,995]: 48.55  [996,996]: 48.92  [997,997]: 49.25  [998,998]: 49.52  [999,999]:
    49.75  [1000,1000]: 49.92
---------- Iteration number: 2400 ------------
[995,995]: 48.59  [996,996]: 48.95  [997,997]: 49.26  [998,998]: 49.53  [999,999]:
    49.75  [1000,1000]: 49.93
---------- Iteration number: 2500 ------------
[995,995]: 48.62  [996,996]: 48.97  [997,997]: 49.28  [998,998]: 49.54  [999,999]:
    49.76  [1000,1000]: 49.93
---------- Iteration number: 2600 ------------
[995,995]: 48.66  [996,996]: 49.00  [997,997]: 49.29  [998,998]: 49.55  [999,999]:
    49.76  [1000,1000]: 49.93
---------- Iteration number: 2700 ------------
[995,995]: 48.69  [996,996]: 49.02  [997,997]: 49.31  [998,998]: 49.56  [999,999]:
    49.76  [1000,1000]: 49.93
---------- Iteration number: 2800 ------------
[995,995]: 48.71  [996,996]: 49.04  [997,997]: 49.32  [998,998]: 49.56  [999,999]:
    49.77  [1000,1000]: 49.93
---------- Iteration number: 2900 ------------
[995,995]: 48.74  [996,996]: 49.06  [997,997]: 49.33  [998,998]: 49.57  [999,999]:
    49.77  [1000,1000]: 49.93
---------- Iteration number: 3000 ------------
[995,995]: 48.77  [996,996]: 49.07  [997,997]: 49.34  [998,998]: 49.58  [999,999]:
    49.77  [1000,1000]: 49.93
---------- Iteration number: 3100 ------------
[995,995]: 48.79  [996,996]: 49.09  [997,997]: 49.35  [998,998]: 49.58  [999,999]:
    49.78  [1000,1000]: 49.93
```

```
---------- Iteration number: 3200 ------------
[995,995]: 48.81  [996,996]: 49.10  [997,997]: 49.36  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3300 ------------
[995,995]: 48.83  [996,996]: 49.12  [997,997]: 49.37  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3400 ------------
[995,995]: 48.85  [996,996]: 49.13  [997,997]: 49.38  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3500 ------------
[995,995]: 48.87  [996,996]: 49.14  [997,997]: 49.39  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3600 ------------
[995,995]: 48.88  [996,996]: 49.16  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3700 ------------
[995,995]: 48.90  [996,996]: 49.17  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3800 ------------
[995,995]: 48.92  [996,996]: 49.18  [997,997]: 49.41  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
---------- Iteration number: 3900 ------------
[995,995]: 48.93  [996,996]: 49.19  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
---------- Iteration number: 4000 ------------
[995,995]: 48.94  [996,996]: 49.20  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94

Max error at iteration 4000 was 20.922598
Total time was 7.396640 seconds.
Grid size: 1000x1000, Processes: 4
End time: 2025-07-06 16:20:11.123943861
Total wall clock time: 7.723 seconds
----------------------------------------

=== Test with 8 pe ===
Start time: 2025-07-06 16:20:12.132072715

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
Maximum iterations [100-4000]?
Running on 8 processes with dynamic load balancing
---------- Iteration number: 100 ------------
[995,995]: 31.66  [996,996]: 36.34  [997,997]: 40.70  [998,998]: 44.48  [999,999]:
     47.43  [1000,1000]: 49.33
---------- Iteration number: 200 ------------
[995,995]: 39.55  [996,996]: 42.43  [997,997]: 44.96  [998,998]: 47.05  [999,999]:
     48.63  [1000,1000]: 49.64
---------- Iteration number: 300 ------------
[995,995]: 42.62  [996,996]: 44.69  [997,997]: 46.48  [998,998]: 47.94  [999,999]:
     49.04  [1000,1000]: 49.75
---------- Iteration number: 400 ------------
```

```
[995,995]: 44.25  [996,996]: 45.87  [997,997]: 47.26  [998,998]: 48.39  [999,999]:
    49.24  [1000,1000]: 49.80
---------- Iteration number: 500 ------------
[995,995]: 45.26  [996,996]: 46.60  [997,997]: 47.74  [998,998]: 48.66  [999,999]:
    49.36  [1000,1000]: 49.83
---------- Iteration number: 600 ------------
[995,995]: 45.94  [996,996]: 47.08  [997,997]: 48.06  [998,998]: 48.85  [999,999]:
    49.45  [1000,1000]: 49.85
---------- Iteration number: 700 ------------
[995,995]: 46.44  [996,996]: 47.44  [997,997]: 48.28  [998,998]: 48.98  [999,999]:
    49.50  [1000,1000]: 49.86
---------- Iteration number: 800 ------------
[995,995]: 46.81  [996,996]: 47.70  [997,997]: 48.46  [998,998]: 49.07  [999,999]:
    49.55  [1000,1000]: 49.87
---------- Iteration number: 900 ------------
[995,995]: 47.10  [996,996]: 47.91  [997,997]: 48.59  [998,998]: 49.15  [999,999]:
    49.58  [1000,1000]: 49.88
---------- Iteration number: 1000 -----------
[995,995]: 47.34  [996,996]: 48.07  [997,997]: 48.70  [998,998]: 49.21  [999,999]:
    49.61  [1000,1000]: 49.89
---------- Iteration number: 1100 -----------
[995,995]: 47.53  [996,996]: 48.21  [997,997]: 48.79  [998,998]: 49.26  [999,999]:
    49.63  [1000,1000]: 49.90
---------- Iteration number: 1200 -----------
[995,995]: 47.69  [996,996]: 48.32  [997,997]: 48.86  [998,998]: 49.30  [999,999]:
    49.65  [1000,1000]: 49.90
---------- Iteration number: 1300 -----------
[995,995]: 47.83  [996,996]: 48.42  [997,997]: 48.92  [998,998]: 49.34  [999,999]:
    49.67  [1000,1000]: 49.90
---------- Iteration number: 1400 -----------
[995,995]: 47.95  [996,996]: 48.50  [997,997]: 48.98  [998,998]: 49.37  [999,999]:
    49.68  [1000,1000]: 49.91
---------- Iteration number: 1500 -----------
[995,995]: 48.05  [996,996]: 48.57  [997,997]: 49.02  [998,998]: 49.40  [999,999]:
    49.69  [1000,1000]: 49.91
---------- Iteration number: 1600 -----------
[995,995]: 48.14  [996,996]: 48.64  [997,997]: 49.06  [998,998]: 49.42  [999,999]:
    49.70  [1000,1000]: 49.91
---------- Iteration number: 1700 -----------
[995,995]: 48.22  [996,996]: 48.69  [997,997]: 49.10  [998,998]: 49.44  [999,999]:
    49.71  [1000,1000]: 49.92
---------- Iteration number: 1800 -----------
[995,995]: 48.29  [996,996]: 48.74  [997,997]: 49.13  [998,998]: 49.46  [999,999]:
    49.72  [1000,1000]: 49.92
---------- Iteration number: 1900 -----------
[995,995]: 48.35  [996,996]: 48.78  [997,997]: 49.16  [998,998]: 49.47  [999,999]:
    49.73  [1000,1000]: 49.92
---------- Iteration number: 2000 -----------
[995,995]: 48.41  [996,996]: 48.82  [997,997]: 49.18  [998,998]: 49.49  [999,999]:
    49.73  [1000,1000]: 49.92
---------- Iteration number: 2100 -----------
[995,995]: 48.46  [996,996]: 48.86  [997,997]: 49.21  [998,998]: 49.50  [999,999]:
    49.74  [1000,1000]: 49.92
```

```
---------- Iteration number: 2200 ------------
[995,995]: 48.51  [996,996]: 48.89  [997,997]: 49.23  [998,998]: 49.51  [999,999]:
     49.74  [1000,1000]: 49.92
---------- Iteration number: 2300 ------------
[995,995]: 48.55  [996,996]: 48.92  [997,997]: 49.25  [998,998]: 49.52  [999,999]:
     49.75  [1000,1000]: 49.92
---------- Iteration number: 2400 ------------
[995,995]: 48.59  [996,996]: 48.95  [997,997]: 49.26  [998,998]: 49.53  [999,999]:
     49.75  [1000,1000]: 49.93
---------- Iteration number: 2500 ------------
[995,995]: 48.62  [996,996]: 48.97  [997,997]: 49.28  [998,998]: 49.54  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2600 ------------
[995,995]: 48.66  [996,996]: 49.00  [997,997]: 49.29  [998,998]: 49.55  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2700 ------------
[995,995]: 48.69  [996,996]: 49.02  [997,997]: 49.31  [998,998]: 49.56  [999,999]:
     49.76  [1000,1000]: 49.93
---------- Iteration number: 2800 ------------
[995,995]: 48.71  [996,996]: 49.04  [997,997]: 49.32  [998,998]: 49.56  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 2900 ------------
[995,995]: 48.74  [996,996]: 49.06  [997,997]: 49.33  [998,998]: 49.57  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 3000 ------------
[995,995]: 48.77  [996,996]: 49.07  [997,997]: 49.34  [998,998]: 49.58  [999,999]:
     49.77  [1000,1000]: 49.93
---------- Iteration number: 3100 ------------
[995,995]: 48.79  [996,996]: 49.09  [997,997]: 49.35  [998,998]: 49.58  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3200 ------------
[995,995]: 48.81  [996,996]: 49.10  [997,997]: 49.36  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3300 ------------
[995,995]: 48.83  [996,996]: 49.12  [997,997]: 49.37  [998,998]: 49.59  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3400 ------------
[995,995]: 48.85  [996,996]: 49.13  [997,997]: 49.38  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3500 ------------
[995,995]: 48.87  [996,996]: 49.14  [997,997]: 49.39  [998,998]: 49.60  [999,999]:
     49.78  [1000,1000]: 49.93
---------- Iteration number: 3600 ------------
[995,995]: 48.88  [996,996]: 49.16  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3700 ------------
[995,995]: 48.90  [996,996]: 49.17  [997,997]: 49.40  [998,998]: 49.61  [999,999]:
     49.79  [1000,1000]: 49.93
---------- Iteration number: 3800 ------------
[995,995]: 48.92  [996,996]: 49.18  [997,997]: 49.41  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
---------- Iteration number: 3900 ------------
[995,995]: 48.93  [996,996]: 49.19  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
```

```
     49.79  [1000,1000]: 49.94

Max error at iteration 4000 was 20.922598
Total time was 4.141792 seconds.
Grid size: 1000x1000, Processes: 8
---------- Iteration number: 4000 ------------
[995,995]: 48.94  [996,996]: 49.20  [997,997]: 49.42  [998,998]: 49.62  [999,999]:
     49.79  [1000,1000]: 49.94
End time: 2025-07-06 16:20:16.632110165
Total wall clock time: 4.497 seconds
----------------------------------------

!!!!STARTING MPI PROCESS TEST -final optimized !!!!
=== Test with 1 pe ===
Start time: 2025-07-06 16:20:17.641053226

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
---------- Iteration number: 100 ------------
[995,995]: 63.74  [996,996]: 73.08  [997,997]: 81.78  [998,998]: 89.28  [999,999]:
     95.08  [1000,1000]: 98.75
---------- Iteration number: 200 ------------
[995,995]: 79.68  [996,996]: 85.39  [997,997]: 90.37  [998,998]: 94.46  [999,999]:
     97.50  [1000,1000]: 99.37
---------- Iteration number: 300 ------------
[995,995]: 85.91  [996,996]: 89.98  [997,997]: 93.46  [998,998]: 96.27  [999,999]:
     98.32  [1000,1000]: 99.58
---------- Iteration number: 400 ------------
[995,995]: 89.22  [996,996]: 92.38  [997,997]: 95.05  [998,998]: 97.18  [999,999]:
     98.74  [1000,1000]: 99.68
---------- Iteration number: 500 ------------
[995,995]: 91.27  [996,996]: 93.85  [997,997]: 96.02  [998,998]: 97.74  [999,999]:
     98.99  [1000,1000]: 99.75
---------- Iteration number: 600 ------------
[995,995]: 92.67  [996,996]: 94.85  [997,997]: 96.67  [998,998]: 98.11  [999,999]:
     99.16  [1000,1000]: 99.79
---------- Iteration number: 700 ------------
[995,995]: 93.68  [996,996]: 95.56  [997,997]: 97.14  [998,998]: 98.38  [999,999]:
     99.28  [1000,1000]: 99.82
---------- Iteration number: 800 ------------
[995,995]: 94.44  [996,996]: 96.11  [997,997]: 97.49  [998,998]: 98.58  [999,999]:
     99.37  [1000,1000]: 99.84
---------- Iteration number: 900 ------------
[995,995]: 95.04  [996,996]: 96.53  [997,997]: 97.77  [998,998]: 98.74  [999,999]:
     99.44  [1000,1000]: 99.86
---------- Iteration number: 1000 ------------
[995,995]: 95.53  [996,996]: 96.87  [997,997]: 97.99  [998,998]: 98.86  [999,999]:
     99.49  [1000,1000]: 99.87
---------- Iteration number: 1100 ------------
[995,995]: 95.93  [996,996]: 97.15  [997,997]: 98.17  [998,998]: 98.96  [999,999]:
     99.54  [1000,1000]: 99.88
---------- Iteration number: 1200 ------------
```

```
[995,995]: 96.26  [996,996]: 97.39  [997,997]: 98.32  [998,998]: 99.05  [999,999]:
    99.58  [1000,1000]: 99.89
---------- Iteration number: 1300 ------------
[995,995]: 96.54  [996,996]: 97.58  [997,997]: 98.45  [998,998]: 99.12  [999,999]:
    99.61  [1000,1000]: 99.90
---------- Iteration number: 1400 ------------
[995,995]: 96.78  [996,996]: 97.75  [997,997]: 98.56  [998,998]: 99.19  [999,999]:
    99.64  [1000,1000]: 99.91
---------- Iteration number: 1500 ------------
[995,995]: 96.99  [996,996]: 97.90  [997,997]: 98.65  [998,998]: 99.24  [999,999]:
    99.66  [1000,1000]: 99.92
---------- Iteration number: 1600 ------------
[995,995]: 97.18  [996,996]: 98.03  [997,997]: 98.74  [998,998]: 99.29  [999,999]:
    99.68  [1000,1000]: 99.92
---------- Iteration number: 1700 ------------
[995,995]: 97.34  [996,996]: 98.15  [997,997]: 98.81  [998,998]: 99.33  [999,999]:
    99.70  [1000,1000]: 99.93
---------- Iteration number: 1800 ------------
[995,995]: 97.49  [996,996]: 98.25  [997,997]: 98.88  [998,998]: 99.37  [999,999]:
    99.72  [1000,1000]: 99.93
---------- Iteration number: 1900 ------------
[995,995]: 97.62  [996,996]: 98.34  [997,997]: 98.93  [998,998]: 99.40  [999,999]:
    99.73  [1000,1000]: 99.93
---------- Iteration number: 2000 ------------
[995,995]: 97.74  [996,996]: 98.42  [997,997]: 98.99  [998,998]: 99.43  [999,999]:
    99.75  [1000,1000]: 99.94
---------- Iteration number: 2100 ------------
[995,995]: 97.84  [996,996]: 98.50  [997,997]: 99.04  [998,998]: 99.46  [999,999]:
    99.76  [1000,1000]: 99.94
---------- Iteration number: 2200 ------------
[995,995]: 97.94  [996,996]: 98.56  [997,997]: 99.08  [998,998]: 99.48  [999,999]:
    99.77  [1000,1000]: 99.94
---------- Iteration number: 2300 ------------
[995,995]: 98.03  [996,996]: 98.63  [997,997]: 99.12  [998,998]: 99.50  [999,999]:
    99.78  [1000,1000]: 99.94
---------- Iteration number: 2400 ------------
[995,995]: 98.11  [996,996]: 98.68  [997,997]: 99.16  [998,998]: 99.52  [999,999]:
    99.79  [1000,1000]: 99.95
---------- Iteration number: 2500 ------------
[995,995]: 98.18  [996,996]: 98.74  [997,997]: 99.19  [998,998]: 99.54  [999,999]:
    99.80  [1000,1000]: 99.95
---------- Iteration number: 2600 ------------
[995,995]: 98.25  [996,996]: 98.78  [997,997]: 99.22  [998,998]: 99.56  [999,999]:
    99.80  [1000,1000]: 99.95
---------- Iteration number: 2700 ------------
[995,995]: 98.32  [996,996]: 98.83  [997,997]: 99.25  [998,998]: 99.58  [999,999]:
    99.81  [1000,1000]: 99.95
---------- Iteration number: 2800 ------------
[995,995]: 98.38  [996,996]: 98.87  [997,997]: 99.28  [998,998]: 99.59  [999,999]:
    99.82  [1000,1000]: 99.95
---------- Iteration number: 2900 ------------
[995,995]: 98.43  [996,996]: 98.91  [997,997]: 99.30  [998,998]: 99.61  [999,999]:
    99.82  [1000,1000]: 99.96
```

```
---------- Iteration number: 3000 ------------
[995,995]: 98.48  [996,996]: 98.95  [997,997]: 99.32  [998,998]: 99.62  [999,999]:
     99.83  [1000,1000]: 99.96
---------- Iteration number: 3100 ------------
[995,995]: 98.53  [996,996]: 98.98  [997,997]: 99.35  [998,998]: 99.63  [999,999]:
     99.84  [1000,1000]: 99.96
---------- Iteration number: 3200 ------------
[995,995]: 98.58  [996,996]: 99.01  [997,997]: 99.37  [998,998]: 99.64  [999,999]:
     99.84  [1000,1000]: 99.96
---------- Iteration number: 3300 ------------
[995,995]: 98.62  [996,996]: 99.04  [997,997]: 99.38  [998,998]: 99.65  [999,999]:
     99.85  [1000,1000]: 99.96
---------- Iteration number: 3400 ------------
[995,995]: 98.66  [996,996]: 99.07  [997,997]: 99.40  [998,998]: 99.66  [999,999]:
     99.85  [1000,1000]: 99.96
---------- Iteration number: 3500 ------------
[995,995]: 98.70  [996,996]: 99.10  [997,997]: 99.42  [998,998]: 99.67  [999,999]:
     99.85  [1000,1000]: 99.96
---------- Iteration number: 3600 ------------
[995,995]: 98.74  [996,996]: 99.12  [997,997]: 99.44  [998,998]: 99.68  [999,999]:
     99.86  [1000,1000]: 99.96

================================ result ===================================
Max error at iteration 3602 was 0.009998
Total time was 23.325403 seconds.
============================================================================
End time: 2025-07-06 16:20:41.233464220
Total wall clock time: 23.589 seconds
----------------------------------------

=== Test with 4 pe ===
Start time: 2025-07-06 16:20:42.241139637

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
================================================================
---------- Iteration number: 100 ------------
[245,995]: 63.74  [246,996]: 73.08  [247,997]: 81.78  [248,998]: 89.28  [249,999]:
     95.08  [250,1000]: 98.75
---------- Iteration number: 200 ------------
[245,995]: 79.68  [246,996]: 85.39  [247,997]: 90.37  [248,998]: 94.46  [249,999]:
     97.50  [250,1000]: 99.37
---------- Iteration number: 300 ------------
[245,995]: 85.91  [246,996]: 89.98  [247,997]: 93.46  [248,998]: 96.27  [249,999]:
     98.32  [250,1000]: 99.58
---------- Iteration number: 400 ------------
[245,995]: 89.22  [246,996]: 92.38  [247,997]: 95.05  [248,998]: 97.18  [249,999]:
     98.74  [250,1000]: 99.68
---------- Iteration number: 500 ------------
[245,995]: 91.27  [246,996]: 93.85  [247,997]: 96.02  [248,998]: 97.74  [249,999]:
     98.99  [250,1000]: 99.75
---------- Iteration number: 600 ------------
[245,995]: 92.67  [246,996]: 94.85  [247,997]: 96.67  [248,998]: 98.11  [249,999]:
```

```
     99.16  [250,1000]: 99.79
---------- Iteration number: 700 ------------
[245,995]: 93.68  [246,996]: 95.56  [247,997]: 97.14  [248,998]: 98.38  [249,999]:
     99.28  [250,1000]: 99.82
---------- Iteration number: 800 ------------
[245,995]: 94.44  [246,996]: 96.11  [247,997]: 97.49  [248,998]: 98.58  [249,999]:
     99.37  [250,1000]: 99.84
---------- Iteration number: 900 ------------
[245,995]: 95.04  [246,996]: 96.53  [247,997]: 97.77  [248,998]: 98.74  [249,999]:
     99.44  [250,1000]: 99.86
---------- Iteration number: 1000 ------------
[245,995]: 95.53  [246,996]: 96.87  [247,997]: 97.99  [248,998]: 98.86  [249,999]:
     99.49  [250,1000]: 99.87
---------- Iteration number: 1100 ------------
[245,995]: 95.93  [246,996]: 97.15  [247,997]: 98.17  [248,998]: 98.96  [249,999]:
     99.54  [250,1000]: 99.88
---------- Iteration number: 1200 ------------
[245,995]: 96.26  [246,996]: 97.39  [247,997]: 98.32  [248,998]: 99.05  [249,999]:
     99.58  [250,1000]: 99.89
---------- Iteration number: 1300 ------------
[245,995]: 96.54  [246,996]: 97.58  [247,997]: 98.45  [248,998]: 99.12  [249,999]:
     99.61  [250,1000]: 99.90
---------- Iteration number: 1400 ------------
[245,995]: 96.78  [246,996]: 97.75  [247,997]: 98.56  [248,998]: 99.19  [249,999]:
     99.64  [250,1000]: 99.91
---------- Iteration number: 1500 ------------
[245,995]: 96.99  [246,996]: 97.90  [247,997]: 98.65  [248,998]: 99.24  [249,999]:
     99.66  [250,1000]: 99.92
---------- Iteration number: 1600 ------------
[245,995]: 97.18  [246,996]: 98.03  [247,997]: 98.74  [248,998]: 99.29  [249,999]:
     99.68  [250,1000]: 99.92
---------- Iteration number: 1700 ------------
[245,995]: 97.34  [246,996]: 98.15  [247,997]: 98.81  [248,998]: 99.33  [249,999]:
     99.70  [250,1000]: 99.93
---------- Iteration number: 1800 ------------
[245,995]: 97.49  [246,996]: 98.25  [247,997]: 98.88  [248,998]: 99.37  [249,999]:
     99.72  [250,1000]: 99.93
---------- Iteration number: 1900 ------------
[245,995]: 97.62  [246,996]: 98.34  [247,997]: 98.93  [248,998]: 99.40  [249,999]:
     99.73  [250,1000]: 99.93
---------- Iteration number: 2000 ------------
[245,995]: 97.74  [246,996]: 98.42  [247,997]: 98.99  [248,998]: 99.43  [249,999]:
     99.75  [250,1000]: 99.94
---------- Iteration number: 2100 ------------
[245,995]: 97.84  [246,996]: 98.50  [247,997]: 99.04  [248,998]: 99.46  [249,999]:
     99.76  [250,1000]: 99.94
---------- Iteration number: 2200 ------------
[245,995]: 97.94  [246,996]: 98.56  [247,997]: 99.08  [248,998]: 99.48  [249,999]:
     99.77  [250,1000]: 99.94
---------- Iteration number: 2300 ------------
[245,995]: 98.03  [246,996]: 98.63  [247,997]: 99.12  [248,998]: 99.50  [249,999]:
     99.78  [250,1000]: 99.94
---------- Iteration number: 2400 ------------
```

```
[245,995]: 98.11  [246,996]: 98.68  [247,997]: 99.16  [248,998]: 99.52  [249,999]:
    99.79  [250,1000]: 99.95
---------- Iteration number: 2500 ------------
[245,995]: 98.18  [246,996]: 98.74  [247,997]: 99.19  [248,998]: 99.54  [249,999]:
    99.80  [250,1000]: 99.95
---------- Iteration number: 2600 ------------
[245,995]: 98.25  [246,996]: 98.78  [247,997]: 99.22  [248,998]: 99.56  [249,999]:
    99.80  [250,1000]: 99.95
---------- Iteration number: 2700 ------------
[245,995]: 98.32  [246,996]: 98.83  [247,997]: 99.25  [248,998]: 99.58  [249,999]:
    99.81  [250,1000]: 99.95
---------- Iteration number: 2800 ------------
[245,995]: 98.38  [246,996]: 98.87  [247,997]: 99.28  [248,998]: 99.59  [249,999]:
    99.82  [250,1000]: 99.95
---------- Iteration number: 2900 ------------
[245,995]: 98.43  [246,996]: 98.91  [247,997]: 99.30  [248,998]: 99.61  [249,999]:
    99.82  [250,1000]: 99.96
---------- Iteration number: 3000 ------------
[245,995]: 98.48  [246,996]: 98.95  [247,997]: 99.32  [248,998]: 99.62  [249,999]:
    99.83  [250,1000]: 99.96
---------- Iteration number: 3100 ------------
[245,995]: 98.53  [246,996]: 98.98  [247,997]: 99.35  [248,998]: 99.63  [249,999]:
    99.84  [250,1000]: 99.96
---------- Iteration number: 3200 ------------
[245,995]: 98.58  [246,996]: 99.01  [247,997]: 99.37  [248,998]: 99.64  [249,999]:
    99.84  [250,1000]: 99.96
---------- Iteration number: 3300 ------------
[245,995]: 98.62  [246,996]: 99.04  [247,997]: 99.38  [248,998]: 99.65  [249,999]:
    99.85  [250,1000]: 99.96
---------- Iteration number: 3400 ------------
[245,995]: 98.66  [246,996]: 99.07  [247,997]: 99.40  [248,998]: 99.66  [249,999]:
    99.85  [250,1000]: 99.96
---------- Iteration number: 3500 ------------
[245,995]: 98.70  [246,996]: 99.10  [247,997]: 99.42  [248,998]: 99.67  [249,999]:
    99.85  [250,1000]: 99.96
---------- Iteration number: 3600 ------------
[245,995]: 98.74  [246,996]: 99.12  [247,997]: 99.44  [248,998]: 99.68  [249,999]:
    99.86  [250,1000]: 99.96

================================ result ==================================
Max error at iteration 3602 was 0.009998
Total time was 6.706944 seconds.
==========================================================================
End time: 2025-07-06 16:20:49.240479334
Total wall clock time: 6.996 seconds
--------------------------------------

=== Test with 8 pe ===
Start time: 2025-07-06 16:20:50.248903255

======================   ALLOCATED NODES   ======================
        cn093: flags=0x11 slots=8 max_slots=0 slots_inuse=0 state=UP
================================================================
```

```
---------- Iteration number: 100 ------------
[120,995]: 63.74  [121,996]: 73.08  [122,997]: 81.78  [123,998]: 89.28  [124,999]:
     95.08  [125,1000]: 98.75
---------- Iteration number: 200 ------------
[120,995]: 79.68  [121,996]: 85.39  [122,997]: 90.37  [123,998]: 94.46  [124,999]:
     97.50  [125,1000]: 99.37
---------- Iteration number: 300 ------------
[120,995]: 85.91  [121,996]: 89.98  [122,997]: 93.46  [123,998]: 96.27  [124,999]:
     98.32  [125,1000]: 99.58
---------- Iteration number: 400 ------------
[120,995]: 89.22  [121,996]: 92.38  [122,997]: 95.05  [123,998]: 97.18  [124,999]:
     98.74  [125,1000]: 99.68
---------- Iteration number: 500 ------------
[120,995]: 91.27  [121,996]: 93.85  [122,997]: 96.02  [123,998]: 97.74  [124,999]:
     98.99  [125,1000]: 99.75
---------- Iteration number: 600 ------------
[120,995]: 92.67  [121,996]: 94.85  [122,997]: 96.67  [123,998]: 98.11  [124,999]:
     99.16  [125,1000]: 99.79
---------- Iteration number: 700 ------------
[120,995]: 93.68  [121,996]: 95.56  [122,997]: 97.14  [123,998]: 98.38  [124,999]:
     99.28  [125,1000]: 99.82
---------- Iteration number: 800 ------------
[120,995]: 94.44  [121,996]: 96.11  [122,997]: 97.49  [123,998]: 98.58  [124,999]:
     99.37  [125,1000]: 99.84
---------- Iteration number: 900 ------------
[120,995]: 95.04  [121,996]: 96.53  [122,997]: 97.77  [123,998]: 98.74  [124,999]:
     99.44  [125,1000]: 99.86
---------- Iteration number: 1000 ------------
[120,995]: 95.53  [121,996]: 96.87  [122,997]: 97.99  [123,998]: 98.86  [124,999]:
     99.49  [125,1000]: 99.87
---------- Iteration number: 1100 ------------
[120,995]: 95.93  [121,996]: 97.15  [122,997]: 98.17  [123,998]: 98.96  [124,999]:
     99.54  [125,1000]: 99.88
---------- Iteration number: 1200 ------------
[120,995]: 96.26  [121,996]: 97.39  [122,997]: 98.32  [123,998]: 99.05  [124,999]:
     99.58  [125,1000]: 99.89
---------- Iteration number: 1300 ------------
[120,995]: 96.54  [121,996]: 97.58  [122,997]: 98.45  [123,998]: 99.12  [124,999]:
     99.61  [125,1000]: 99.90
---------- Iteration number: 1400 ------------
[120,995]: 96.78  [121,996]: 97.75  [122,997]: 98.56  [123,998]: 99.19  [124,999]:
     99.64  [125,1000]: 99.91
---------- Iteration number: 1500 ------------
[120,995]: 96.99  [121,996]: 97.90  [122,997]: 98.65  [123,998]: 99.24  [124,999]:
     99.66  [125,1000]: 99.92
---------- Iteration number: 1600 ------------
[120,995]: 97.18  [121,996]: 98.03  [122,997]: 98.74  [123,998]: 99.29  [124,999]:
     99.68  [125,1000]: 99.92
---------- Iteration number: 1700 ------------
[120,995]: 97.34  [121,996]: 98.15  [122,997]: 98.81  [123,998]: 99.33  [124,999]:
     99.70  [125,1000]: 99.93
---------- Iteration number: 1800 ------------
[120,995]: 97.49  [121,996]: 98.25  [122,997]: 98.88  [123,998]: 99.37  [124,999]:
```

```
     99.72  [125,1000]: 99.93
---------- Iteration number: 1900 ------------
[120,995]: 97.62  [121,996]: 98.34  [122,997]: 98.93  [123,998]: 99.40  [124,999]:
     99.73  [125,1000]: 99.93
---------- Iteration number: 2000 ------------
[120,995]: 97.74  [121,996]: 98.42  [122,997]: 98.99  [123,998]: 99.43  [124,999]:
     99.75  [125,1000]: 99.94
---------- Iteration number: 2100 ------------
[120,995]: 97.84  [121,996]: 98.50  [122,997]: 99.04  [123,998]: 99.46  [124,999]:
     99.76  [125,1000]: 99.94
---------- Iteration number: 2200 ------------
[120,995]: 97.94  [121,996]: 98.56  [122,997]: 99.08  [123,998]: 99.48  [124,999]:
     99.77  [125,1000]: 99.94
---------- Iteration number: 2300 ------------
[120,995]: 98.03  [121,996]: 98.63  [122,997]: 99.12  [123,998]: 99.50  [124,999]:
     99.78  [125,1000]: 99.94
---------- Iteration number: 2400 ------------
[120,995]: 98.11  [121,996]: 98.68  [122,997]: 99.16  [123,998]: 99.52  [124,999]:
     99.79  [125,1000]: 99.95
---------- Iteration number: 2500 ------------
[120,995]: 98.18  [121,996]: 98.74  [122,997]: 99.19  [123,998]: 99.54  [124,999]:
     99.80  [125,1000]: 99.95
---------- Iteration number: 2600 ------------
[120,995]: 98.25  [121,996]: 98.78  [122,997]: 99.22  [123,998]: 99.56  [124,999]:
     99.80  [125,1000]: 99.95
---------- Iteration number: 2700 ------------
[120,995]: 98.32  [121,996]: 98.83  [122,997]: 99.25  [123,998]: 99.58  [124,999]:
     99.81  [125,1000]: 99.95
---------- Iteration number: 2800 ------------
[120,995]: 98.38  [121,996]: 98.87  [122,997]: 99.27  [123,998]: 99.59  [124,999]:
     99.82  [125,1000]: 99.95
---------- Iteration number: 2900 ------------
[120,995]: 98.43  [121,996]: 98.91  [122,997]: 99.30  [123,998]: 99.61  [124,999]:
     99.82  [125,1000]: 99.96
---------- Iteration number: 3000 ------------
[120,995]: 98.48  [121,996]: 98.94  [122,997]: 99.32  [123,998]: 99.62  [124,999]:
     99.83  [125,1000]: 99.96
---------- Iteration number: 3100 ------------
[120,995]: 98.53  [121,996]: 98.98  [122,997]: 99.34  [123,998]: 99.63  [124,999]:
     99.84  [125,1000]: 99.96
---------- Iteration number: 3200 ------------
[120,995]: 98.58  [121,996]: 99.01  [122,997]: 99.36  [123,998]: 99.64  [124,999]:
     99.84  [125,1000]: 99.96
---------- Iteration number: 3300 ------------
[120,995]: 98.62  [121,996]: 99.04  [122,997]: 99.38  [123,998]: 99.65  [124,999]:
     99.85  [125,1000]: 99.96
---------- Iteration number: 3400 ------------
[120,995]: 98.66  [121,996]: 99.07  [122,997]: 99.40  [123,998]: 99.66  [124,999]:
     99.85  [125,1000]: 99.96

================================ result ==================================
Max error at iteration 3449 was 0.009999
Total time was 3.673978 seconds.
```

```
================================================================================
End time: 2025-07-06 16:20:54.272173932
Total wall clock time: 4.020 seconds
----------------------------------------
```